PENGENDALIAN KESEIMBANGAN PER-AXIS PADA QUADROCOPTER MENGGUNAKAN KONTROLER PROPORSIONAL INTEGRAL DEFERENSIAL (PID) BERBASIS MIKROKONTROLER ATMEGA 168-20AU

SKRIPSI

KONSENTRASI TEKNIK KONTROL

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

Disusun oleh :

RIO ROSDIANTO

NIM. 0810630086 - 63

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN

UNIVERSITAS BRAWIJAYA

FAKULTAS TEKNIK

MALANG

2012

LEMBAR PERSETUJUAN

PENGENDALIAN KESEIMBANGAN PER-AXIS PADA QUADROCOPTER MENGGUNAKAN KONTROLER PROPORSIONAL INTEGRAL DEFERENSIAL (PID) BERBASIS MIKROKONTROLER ATMEGA 168-20AU

SKRIPSI

JURUSAN TEKNIK ELEKTRO

Diajukan untuk memenuhi persyaratan

memperoleh gelar Sarjana Teknik

Disusun oleh :

RIO ROSDIANTO

NIM. 0810630086 - 63

Telah diperiksa dan disetujui oleh

Dosen Pembimbing

Pembimbing I	Pembimbing II
Ir. Purwanto, MT	**Fitriana Suhartati.,ST.,MT**
NIP. 19540424 198601 1 001	**NIP. 19741017 199802 2 001**

LEMBAR PENGESAHAN

PENGENDALIAN KESEIMBANGAN PER-AXIS PADA QUADROCOPTER MENGGUNAKAN KONTROLER PROPORSIONAL INTEGRAL DEFERENSIAL (PID) BERBASIS MIKROKONTROLER ATMEGA 168-20AU

Disusun Oleh :

RIO ROSDIANTO

NIM : 0810630086 – 63

Skripsi ini telah diuji dan dinyatakan lulus

pada tanggal 1 Agustus 2012

Majelis Penguji :

Dr. Ir. Erni Yudaningtyas, MT.
NIP. 19650913 199002 2 001

Muhammad Aziz Muslim,ST.,MT.,Ph.D
NIP. 19741203 200012 1 001

Goegoes Dwi Nusantoro, ST., MT
NIP. 19711013 200604 1 001

Mengetahui :

Ketua Jurusan Teknik Elektro

Dr. Ir. Sholeh Hadi Pramono, MS
NIP. 19580728 198201 1 001

PENGANTAR

Puji dan syukur penulis panjatkan kepada Allah SWT, karena dengan rahmat, taufik dan hidayah-Nya lah skripsi ini dapat diselesaikan. Skripsi berjudul "Pengendalian Keseimbangan Per-*Axis* Pada *Quadrocopter* Menggunakan Kontroler Proporsional Integral Deferensial (PID) Berbasis Mikrokontroler ATMEGA 168-20AU" ini disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Teknik di Jurusan Teknik Elektro Universitas Brawijaya.

Penulis menyadari bahwa penyusunan skripsi ini tidak terlepas dari bantuan berbagai pihak. Oleh karena itu, dengan ketulusan dan kerendahan hati penulis menyampaikan terima kasih kepada:

1. Allah SWT atas rahmat dan hidayah yang diberikan,
2. Rasulullah Muhammad SAW semoga sholawat dan salam tetap tercurah kepada beliau,
3. Ibu Roosline Lawandy, Ayah H.Priyadi atas segala nasehat, kasih sayang, perhatian dan kesabarannya di dalam membesarkan dan mendidik penulis, serta telah banyak mendoakan kelancaran penulis hingga terselesaikannya skripsi ini,
4. Kedua adik penulis Rivo Adiputra dan Luthfiyah Rachmah,
5. Bapak Dr. Ir. Sholeh Hadi Pramono, MS selaku Ketua Jurusan Teknik Elektro Universitas Brawijaya,
6. Bapak Aziz Muslim, ST,.MT.,Ph.D selaku Sekretaris Jurusan Teknik Elektro Universitas Brawijaya,
7. Bapak Moch.Rif'an.,ST.,MT selaku Ketua Prodi Strata Satu Jurusan Teknik Elektro Universitas Brawijaya dan juga Dosen Pembimbing akademik penulis atas segala nasehat dan bimbingan yang telah diberikan,
8. Bapak Ir. Purwanto.,MT selaku Ketua Kelompok Dosen Keahlian Sistem Kontrol Jurusan Teknik Elektro Universitas Brawijaya dan juga Dosen Pembimbing I atas segala ilmu, bimbingan, nasehat, gagasan, ide, saran, motivasi dan bantuan yang telah diberikan,
9. Ibu Fitriana Suhartati.,ST.,MT selaku Dosen Pembimbing II atas segala bimbingan, nasehat, gagasan, ide, saran, motivasi dan masukan yang telah diberikan,

10. Staff Recording Jurusan Teknik Elektro,
11. Adek Herastri Nandarini atas segala celoteh, semangat dan doa yang diberikan kepada penulis,
12. Rekan-rekan staff SOI-ASIA Universitas Brawijaya yang mendukung dalam pengerjaan skripsi ini,
13. Rekan-rekan pengerjaan skripsi di Laboratorium Sistem Kontrol, Arif, Mahendra, Wahyu, Seif, Shidqi, Irfan, Mas Aldo,
14. Seluruh Keluarga Besar Asisten Laboratorium Sistem Kontrol Jurusan Teknik Elektro atas segala masukan-masukannya,
15. Seluruh teman-teman, senior dan junior serta semua pihak yang tidak mungkin untuk dicantumkan namanya satu per satu, terima kasih banyak atas bantuan dan dukungannya,

Penulis menyadari bahwa skripsi ini masih belum sempurna, oleh karena itu penulis mengharapkan kritik dan saran yang membangun. Penulis berharap seomga skripsi ini dapat bermanfaat bagi pengembangan ilmu pengetahuan dan teknologi.

Malang 20 Juli 2012

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

Quadrocopter merupakan helikopter dengan empat buah lengan yang masing-masing berjarak 90˚. Pada skripsi ini yang menjadi fokus bahasan adalah pengontrolan per-*axis* pada *quadrocopter*. Pengontrolan per-*axis* pada *quadrocopter* dimaksudkan agar *quadrocopter* mampu menyeimbangkan posisi pada sudut kemiringan 0˚ sebelum melakukan *hovering* (mengambang di udara).

Kontroler PID merupakan kontroler yang memiliki tingkat *error* dan *overshoot* yang kecil. Kontroler PID diprogram pada sebuah mikrokontroler keluaran ATMEL dengan tipe ATMEGA168-20AU. Sensor yang digunakan adalah *gyroscope*. Pemograman dilakukan pada *software* CodeVisionAVR. *Tuning* kontroler PID dilakukan dengan metode *hand tuning* dan didapatkan nilai Kp=3.4, Kd=0.4 dan Ki=4. Dengan nilai Kp, Kd dan Ki tersebut, *quadrocopter* mampu mempertahankan posisi seimbang ketiga *axis* pada sudut 0 derajat.

Kata Kunci : Keseimbangan, *Quadrocopter*, Kontroler PID, *Gyroscope*

BAB I
PENDAHULUAN

1.1 Latar Belakang

Salah satu pekerjaan yang membutuhkan keakuratan tingkat tinggi adalah pencitraan geografis. Pencitraan geografis biasanya dilakukan untuk kepentingan observasi terhadap daerah tertentu. Observasi bertujuan untuk analisa awal kondisi suatu daerah. Maka dari itu pencitraan geografis sangatlah penting. Metode yang paling canggih yang sering digunakan adalah dengan menggunakan pencitraan satelit. Penggunaan metode ini memang memiliki keakuratan tingkat tinggi, akan tetapi penggunaan satelit dan kamera yang harus beresolusi tinggi membuat metode ini merupakan metode dengan biaya termahal. Metode lain yang bisa digunakan adalah dengan menggunakan pesawat terbang. Metode ini pun masih tergolong mahal, karena biaya avtur dan kameranya juga harus beresolusi tinggi (www.pertamina.com).

Alternatif yang paling menguntungkan adalah dengan menggunakan *drone* berupa UAV (*Unmanned Aerial Vehicles*). UAV biasanya merupakan sebuah prototipe pesawat yang dikendalikan oleh *remote control* atau terbang secara otomatis berdasarkan program yang sudah diintegrasikan. Salah satu UAV yang sering digunakan adalah *quadrocopter*. *Quadrocopter* merupakan helikopter yang memiliki daya dorong melalui keempat rotornya. Keempat rotor tersebut diletakkan pada setiap lengan-lengannya dengan posisi sudut antar lengan adalah 90˚. Karena merupakan sebuah miniatur, daya jelajah dan ketahanan keseimbangan terhadap *disturbance* tergolong rendah. Sehingga diperlukan sebuah sistem kontrol yang tepat untuk mempertahankan keseimbangan dari *disturbance*.

Ada tiga hal yang mempengaruhi tingkat keseimbangan sebuah quadrocopter yaitu, *roll axis*, *yaw axis* dan *pitch axis*. Keseimbangan ketiga *axis* ini akan berpengaruh pada kecepatan masing-masing aktuatornya. Kemiringan per kecepatan anguler dapat dideteksi melalui sensor *gyro* sehingga mempengaruhi kecepatan motor

brushless. Pengendalian dengan menggunakan kontroler PID (Proporsional Integral Deferensial) merupakan kontroler berumpan balik yang sederhana dan terbukti dapat memberikan performa kontrol yang optimal. Hal ini dikarenakan penentuan kekuatan *gain* untuk tiap-tiap nilai variabel kontrol P, I dan D cukup mudah.

1.2 Rumusan Masalah

1. Bagaimana merancang kontroler PID (Proporsional Integral Deferensial) untuk mempertahankan keseimbangan *quadrocopter* per-axis?
2. Bagaimana merancang kondisi fisik *quadrocopter* dengan pemilihan komponen-komponen yang tepat?

1.3 Batasan Masalah

Untuk menekankan pada objek pembahasan yang ada, maka pada penelitian ini diberikan batasan masalah sebagai berikut:

1. *Quadrocopter* yang dibuat adalah merupakan sebuah miniatur.
2. Pembahasan ditekankan pada pengendalian keseimbangan dan respons sistem *quadrocopter* per-axis dan diberi *distubance*, kinerja driver dan elektronika tidak dibahas mendalam.
3. Gangguan berupa distribusi beban yang tidak merata perubahan aerodinamis secara alami.
4. Pencitraan geografis dan proses terbang tidak dibahas mendalam.

1.4 Tujuan

Menciptakan sebuah keseimbangan per-axis pada *quadrocopter* dengan pengendalian menggunakan kontroler PID (Proporsional Integral Deferensial).

1.5 Sistematika Penulisan

Agar penyusunan laporan skripsi ini dapat mencapai sasaran dan tidak menyimpang dari judul yang telah ditentukan, maka diperlukan sistematika

pembahasan yang jelas. Pembahasan dalam skripsi ini secara garis besar adalah sebagai berikut:

BAB I **Pendahuluan**

Menjelaskan tentang latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika penulisan.

BAB II **Tinjauan Pustaka**

Menjelaskan teori dasar yang berisi penjelasan tentang teori *quadrocopter*, BLDC, sensor *gyro*, kontroler, kontroler PID, ESC (*Electronic Speed Controller*), dan rangkaian mikrokontroler.

BAB III **Metodologi**

Menjelaskan tentang metodologi penelitian yang terdiri dari studi literatur, perancangan alat, pembuatan alat, pengujian alat, serta pengambilan kesimpulan dan saran.

BAB IV **Perancangan dan Pembuatan Alat**

Menjelaskan tentang perancangan dan pembuatan alat yang meliputi prinsip kerja alat, perancangan perangkat keras dan perangkat lunak.

BAB V **Pengujian dan Analisis**

Menjelaskan tentang pengujian alat dan analisa yang meliputi pengujian bagian blok sistem dan pengujian sistem secara keseluruhan.

BAB VI **Penutup**

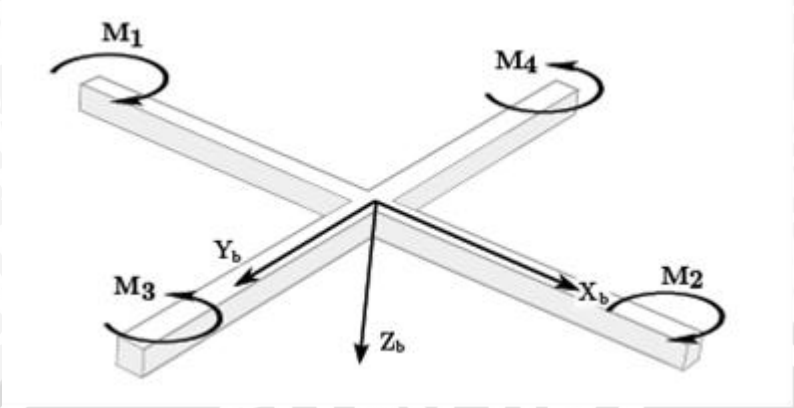
Menjelaskan tentang pengambilan kesimpulan sesuai dengan hasil perealisasian dan pengujian alat sesuai dengan tujuan dan rumusan masalah, serta pemberian saran untuk pengembangan.

BAB II
TINJAUAN PUSTAKA

2.1 *Quadrocopter*

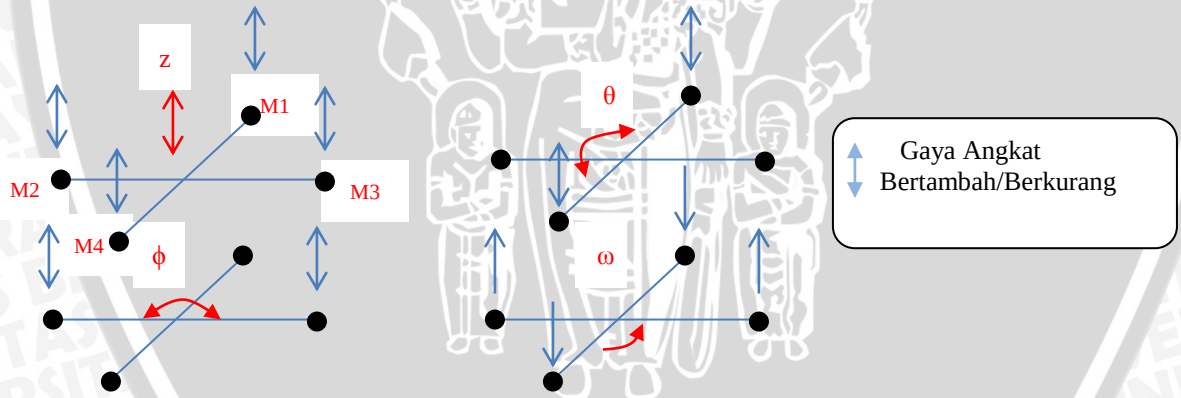
Quadrocopter atau biasa disebut juga dengan quadrotor atau helikopter quadrotor yang diangkat dan menggunakan empat buah *propeller* (baling-baling). *Quadrocopter* bisa diklasifikasikan sebagai helikopter, meskipun berbeda dengan helikopter pada umumnya. *Quadrocopter* menggunakan menggunakan lengan dan rotor yang tetap, yang mana posisi lengan dan rotor tidak akan berubah ketika baling-balingnya berputar. Kontrol dari pergerakan *quadrocopter* bisa dilakukan dengan mengubah-ubah kecepatan relatif dari tiap-tiap rotor untuk menghasilkan gaya angkat dan torsi.

Setiap rotor menghasilkan gaya angkat dan torsi pada pusat rotasinya, dan juga gaya dorong yang berlawanan arah dengan arah gerakan *quadrocopter*. Jika semua rotor berputar dengan kecepatan anguler yang sama, dengan rotor satu dan tiga berputar searah dengan jarum jam serta rotor dua dan rotor empat berputar berlawanan arah jarum jam, maka akan menghasilkan torsi aerodinamis dan juga percepatan anguler terhadap sumbu *yaw* benilai nol. Hal ini mengakibatkan rotor untuk menstabilkan pergerakan sumbu *yaw* seperti pada helikopter pada umumnya tidak dibutuhkan (Luukonen, 2011). Gambar 2.1 menunjukkan skema reaksi torsi dari masing-masing rotor pada *quadrocopter*.



Gambar 2.1 Skema *Quadrocopter* dan *Body Frames*
Sumber : Luukonen, 2011:2

Dengan mengatur gaya angkat yang diberikan tiap motor, ada empat buah kombinasi gerakan yang dapat dilakukan oleh *quadrocopter*. Keempat gerakan tersebut adalah 3 gerakan berputar pada sudut *Tait-Bryan*, ditambah dengan gerakan linear searah sumbu z – koordinat *quadrocopter*. Gambar 2.2 memberikan ilustrasi mengenai skema pergerakan *quadrocopter* (Bresciani, 2008).



Gambar 2.2 Skema Pergerakan *Quadrocopter*
Sumber:Bresciani, 2012

Sesuai dengan skema pergerakan *quadrocopter* di atas, dapat dipastikan bahwa torsi aerodinamis yang dihasilkan masing-masing skema adalah (Raza, 2010:253):

T_z = ((RPM +/- ΔRPM) (M1) + RPM +/- ΔRPM) (M2) + RPM +/- ΔRPM) (M3) + RPM +/- ΔRPM) (M4)).....(2-1)

T_θ = ((RPM +/- ΔRPM) (M1) + (RPM +/- ΔRPM) (M4)) .....(2-2)

T_φ = ((RPM +/- ΔRPM) (M2) + (RPM +/- ΔRPM) (M3)).....(2-3)

T_ω = ((RPM - ΔRPM) (M1) + RPM + ΔRPM) (M2) + RPM + ΔRPM) (M3) + RPM - ΔRPM) (M4)).....(2-4)

dengan : RPM = kecepatan putaran motor

ΔRPM = perubahan kecepatan putaran motor

Jika $P_F^T = [p_x \quad p_y \quad -p_z]$ dan $\Omega_F^T = [\phi \quad \theta \quad \psi]$ merupakan simbol posisi dan orientasi *quadrocopter* pada *frame* F, maka hubungan antara keduanya adalah (Raza, 2010:251) :

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ -\dot{p}_z \end{bmatrix}_{F_i} = \begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \end{bmatrix}_{F_v} = \left[R_{F_v}^{F_b}\right]^T \begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \end{bmatrix}_{F_b} \quad \text{.....(2-5)}$$

dimana $\left[R_{F_v}^{F_b}\right]^T$ merupakan matriks rotasi dengan dimensi 3x3,

$$\left[R_{F_v}^{F_b}\right]^T = \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix} \quad \text{.....(2-6)}$$

dengan sθ = sinθ dan cθ = cosθ. Hal yang sama juga berlaku untuk sψ, cψ, sφ, dan cφ. Untuk mendapatkan persamaan dalam bentuk kecepatan anguler, maka persamaan di atas perlu dideferensialkan terhadap notasi sumbunya masing-masing, sehingga menjadi :

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}_{F_i} = \begin{bmatrix} 1 & s\phi tan\theta & c\phi tan\theta \\ 0 & c\phi & -s\theta \\ 0 & s\phi sec\theta & c\phi sec\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}_{F_b} \quad \text{.....(2-7)}$$

dengan : θ = sumbu *pitch*

φ = sumbu *roll*

ω = sumbu *yaw*

2.2 Kontroler

Keberadaan kontroler dalam sebuah sistem kontrol mempunyai kontribusi yang besar terhadap perilaku sistem. Pada prinsipnya hal itu disebabkan oleh tidak dapat diubahnya komponen penyusun sistem tersebut. Artinya, karakteristik *plant* harus diterima sebagaimana adanya, sehingga perubahan perilaku sistem hanya dapat dilakukan melalui penambahan suatu subsistem, yaitu kontroler.

Salah satu fungsi komponen kontroler adalah mengurangi sinyal kesalahan, yaitu perbedaan antara nilai referensi/nilai yang diinginkan dan nilai aktual. Hal ini sesuai dengan tujuan sistem kontrol di mana mendapat nilai sinyal keluaran sama dengan nilai yang diinginkan/referensi. Semakin kecil kesalahan yang terjadi, semakin baik kinerja sistem kontrol yang diterapkan.

Apabila perbedaan antara nilai referensi dengan nilai keluaran relatif besar, maka kontroler yang baik seharusnya mampu mengatasi perbedaan ini untuk segera menghasilkan sinyal keluaran untuk mempengaruhi *plant*. Dengan demikian sistem secara cepat mengubah keluaran *plant* sampai diperoleh selisih dengan nilai referensi sekecil mungkin.

Prinsip kerja kontroler adalah membandingkan nilai aktual keluaran *plant* dengan nilai referensi, kemudian menentukan nilai kesalahan dan akhirnya menghasilkan sinyal kontrol untuk meminimalkan kesalahan (Ogata, 1995).

2.3 Kontroler PID (Proporsional Integral Deferensial)

2.3.1 Kontroler Proporsional

Kontroler proporsional memiliki keluaran yang sebanding/proporsional dengan besarnya sinyal kesalahan (selisih antara besaran yang diinginkan dengan harga aktualnya). Secara lebih sederhana dapat dikatakan, bahwa keluaran kontroler proporsional merupakan perkalian antara konstanta proporsional dengan masukannya. Perubahan pada sinyal masukan akan segera menyebabkan sistem secara langsung mengubah keluarannya sebesar konstanta pengalinya.

Pada Gambar 2.3 menunjukkan diagram blok yang menggambarkan hubungan antara input (besaran referensi yang diinginkan), besaran aktual dengan

besaran keluaran kontroler proporsional, dan besaran kesalahan (*error*). Sinyal kesalahan (*error*) merupakan selisih antara besaran *setting* dengan besaran aktualnya.

Gambar 2.3 Diagram Blok Kontroler Proporsional
Sumber: Ogata,1995: 157

Pada pengendali proporsional hubungan antara keluaran kontroler m(t) dan sinyal kesalahan e(t) adalah (Ogata, 1995: 157):

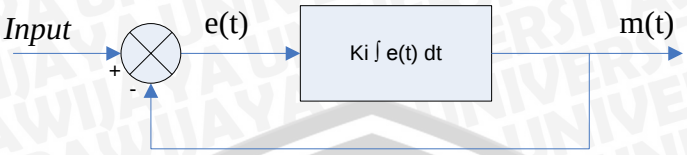
$$m(t) = K_p e(t) \quad \text{.....(2-8)}$$

dengan Kp adalah penguatan proporsional. Keluaran m(t) hanya tergantung pada Kp dan *error*, semakin besar *error* maka semakin besar koreksi yang dilakukan. Penambahan Kp akan menaikkan penguatan sistem sehingga dapat digunakan untuk memperbesar kecepatan respons dan mengurangi kesalahan keadaan mantap.

2.3.2 Kontroler Integral

Kontroler integral berfungsi mengurangi kesalahan keadaan mantap yang dihasilkan pada kontroler proporsional sebelumnya. Kalau sebuah *plant* tidak memiliki unsur integrator (1/s), kontroler proporsional tidak akan mampu menjamin keluaran sistem dengan kesalahan keadaan mantap nol.

Kontroler integral memiliki karakteristik seperti halnya sebuah integral. Keluaran kontroler sangat dipengaruhi oleh perubahan yang sebanding dengan nilai sinyal kesalahan. Keluaran kontroler ini merupakan jumlahan yang terus menerus dari perubahan masukannya. Kalau sinyal kesalahan tidak mengalami perubahan, keluaran akan menjaga keadaan seperti sebelum terjadinya perubahan masukan. Gambar 2.4 menunjukkan diagram blok kontroler integral.



Gambar 2. 4 Diagram Blok Kontroler Integral
Sumber: Ogata, 1995: 158

Nilai keluaran kontroler *m(t)* sebanding dengan integral sinyal kesalahan e(t), sehingga (Ogata, 1995: 157) :

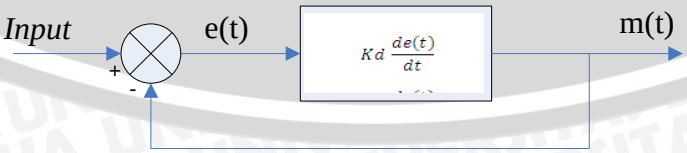
$$\frac{dm(t)}{dt} = Ki.e(t) \quad \text{.....(2-9)}$$

$$m(t) = Ki \int_0^t e(t)dt \quad \text{.....(2-10)}$$

dengan Ki adalah konstanta integral. Jika sinyal kesalahan e(t)=0, maka laju perubahan sinyal kendali integral $\frac{dm(t)}{dt} = 0$ atau sinyal keluaran kendali akan tetap berada pada nilai yang dicapai sebelumnya. Aksi kontrol integral digunakan untuk menghilangkan kesalahan posisi dalam keadaan mantap (*error steady state*) tanpa memperhitungkan kecepatan respons.

2.3.3 Kontroler Deferensial

Kontroler deferensial memiliki sifat seperti halnya suatu operasi derivatif. Perubahan yang mendadak pada masukan kontroler, akan mengakibatkan perubahan yang sangat besar dan cepat. Gambar 2.5 berikut menunjukkan diagram blok pada kontroler deferensial.



Gambar 2. 5 Diagram Blok Kontroler Deferensial
Sumber: Ogata, 1995: 177

Nilai keluaran kontroler m(t) sebanding laju sinyal kesalahan $\frac{de(t)}{dt}$. Hubungan ini dapat ditulis sebagai (Ogata, 1995: 179):

$$m(t) = Kd\ \frac{de(t)}{dt} \quad \text{(2-11)}$$

Kontroler deferensial akan memberikan sinyal kendali keluaran m(t)= 0, untuk sinyal kesalahan e(t) yang konstan sehingga kontroler deferensial tidak mempengaruhi keadaan mantap. Kontroler deferensial digunakan untuk memperbaiki atau mempercepat respons transien sebuah sistem serta dapat meredam osilasi.

Berdasarkan karakteristik kontroler tersebut, kontroler deferensial umumnya dipakai untuk mempercepat respons awal suatu sistem, tetapi tidak memperkecil kesalahan pada keadaan tunaknya. Kerja kontroler deferensial hanyalah efek dari lingkup yang sempit, yaitu pada periode peralihan. Oleh sebab itu kontroler deferensial tidak bisa digunakan tanpa ada kontroler lain.

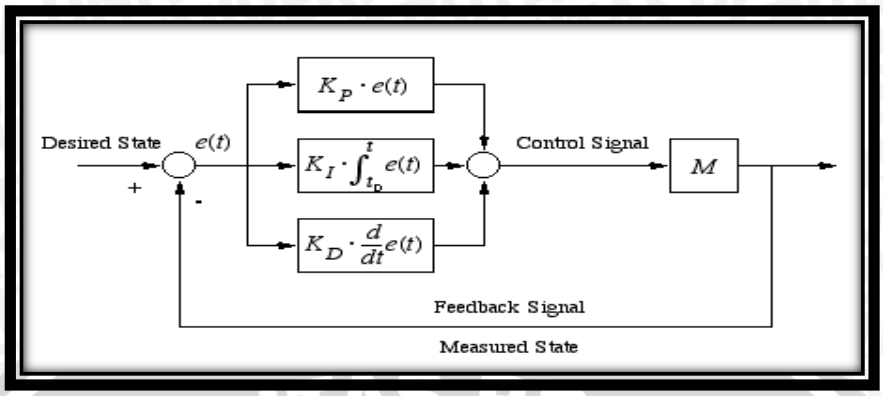
2.3.4 Kontroler Proporsional Integral Deferensial (PID)

Setiap kekurangan dan kelebihan dari masing-masing kontroler P, I dan D dapat saling menutupi dengan menggabungkan ketiganya secara paralel menjadi kontroler proporsional integral deferensial (PID). Elemen-elemen kontroler P, I dan D masing-masing secara keseluruhan bertujuan untuk mempercepat reaksi sebuah sistem, menghilangkan *offset* dan menghasilkan perubahan awal yang besar (Gunterus, 1994, 8-10). Kontroler PID memiliki diagram kendali seperti yang ditujukan dalam Gambar 2.6.

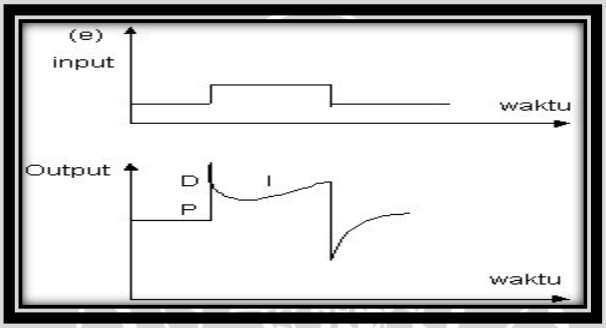
Aksi kontrolnya dinyatakan sebagai (Ogata, 1995: 183):

$$m(t) = K_p e(t) + \frac{Kp}{Ti} \int_0^t e(t)dt + K_p T_d \frac{de(t)}{dt} \quad \text{(2-12)}$$

Jenis kontroler ini digunakan untuk memperbaiki kecepatan respons, mencegah terjadinya kesalahan keadaan mantap serta mempertahankan kestabilan.



Gambar 2. 6 Diagram Blok Kontroler PID
Sumber :Gunterus, 1994:8-11



Gambar 2. 7 Fungsi Waktu antara Sinyal Keluaran dan Sinyal Masukan Kontroler PID
Sumber: Gunterus, 1994:8-11

Keluaran kontroler PID merupakan penjumlahan dari keluaran kontroler proporsional, integral dan deferensial. Gambar 2.7 di atas menunjukkan hubungan tersebut. Karakteristik kontroler PID sangat dipengaruhi oleh kontribusi besar dari ketiga parameter P,I dan D. Penyetelan konstanta Kp, Ti dan Td akan mengakibatkan penonjolan sifat dari masing-masing elemen. Satu atau dua dari ketiga konstanta tersebut dapat disetel lebih menonjol dibanding yang lain.

Konstanta yang menonjol itulah yang akan memberikan kontribusi pada respons sistem secara keseluruhan (Gunterus, 1994, 8-10).

2.3.5 *Hand Tuning* Kontroler PID

Kontroler PID dapat di *tuning* dalam beberapa cara, antara lain Ziegler-Nichols *tuning*, *loop shaping*, metode analitis, optimisasi, *pole placement*, *auto tuning* dan *hand tuning* (Smith, 1979; Astrom & Hagglund, 1995). Pada skripsi ini digunakan cara *hand tuning* untuk menentukan besar K_P, K_i, dan K_d. Hal ini dilakukan karena ada kendala untuk melakukan cara lain yang disebutkan diatas. Kendala tersebut adalah tidak dapat melihat respons motor secara langsung karena tidak digunakannya sensor untuk mengukur kecepatan motor saat sistem berjalan. Selain itu tidak adanya model matematis dari motor membuat cara analitis sulit untuk dilakukan.

Menurut Smith (1979), untuk melakukan *hand tuning* prosedur yang dilakukan adalah sebagai berikut :

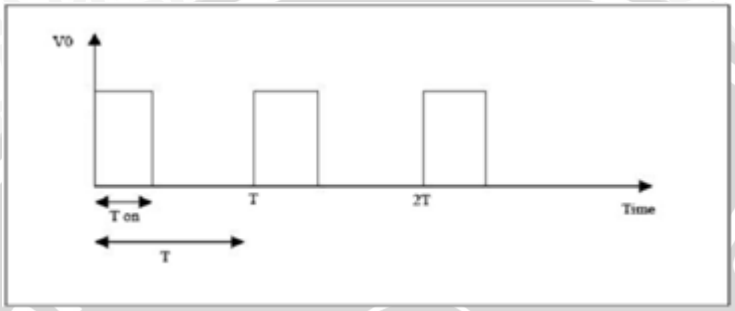
1. Melepaskan kontroler integral dan deferensial dengan memberikan nilai $K_i = 0$ dan $K_d = 0$.
2. Mengatur nilai K_p hingga didapatkan respons yang diinginkan, dengan mengabaikan *offset* dari *setpoint*.
3. Dengan terus menaikkan nilai K_p, nilai dari K_d dinaikkan untuk mengurangi *overshoot* yang terjadi.
4. Naikkan nilai K_i untuk mengurangi *offset*.

Keuntungan dari *hand tuning* adalah prosedur diatas dapat dilakukan dengan segera, *online* dan dapat melihat dengan cepat respons sistem setelah perubahan K_p, K_i dan K_d. Kerugian dari cara ini adalah kesulitan untuk melihat apakah *setting* akhir dari kontroler merupakan nilai optimal atau tidak (Jantzen, 2001).

2.4 PWM (*Pulse Width Modulation*)

PWM (*Pulse Width Modulation*) digunakan untuk mengatur kecepatan dari motor DC. Dimana kecepatan motor DC tergantung pada besarnya *duty cycle* yang diberikan pada motor DC tersebut.

Pada sinyal PWM, frekuensi sinyal konstan sedangkan *duty cycle* bervariasi dari 0%-100%. Dengan mengatur *duty cycle* akan diperoleh keluaran yang diinginkan. Sinyal PWM (*Pulse Width Modulation*) secara umum dapat dilihat dalam Gambar 2.8 berikut:



Gambar 2.8 Gambar Sinyal PWM Secara Umum
Sumber : electronics-scheme.com

$$Dutycycle = \frac{Ton}{T}\ x\ 100\%\ \dots\ (\%) \quad \text{.....(2-13)}$$

Dengan :

Ton = Periode logika tinggi

T = Periode keseluruhan

$$Vdc = Dutycycle\ x\ Vcc\ \dots\ (V) \quad \text{.....(2-14)}$$

Sedangkan frekuensi sinyal dapat ditentukan dengan rumus berikut :

$$f0n = \frac{fclk\ I/0}{N.256}\ \dots\ (Hz) \quad \text{.....(2-15)}$$

2.5 BLDC (*Brushless Motor Direct Current)*

Brushless motor atau yang juga dikenal sebagai motor komutasi elektrik adalah motor elektrik yang dicatu dengan arus/tegangan DC (*direct-current*) dan mempunyai sistem komutasi elektrik, dibandingkan dengan komutator mekanik dengan sikat. Hubungan antara arus dan torsi serta frekuensi dan kecepatan adalah linier (Brown, 2002:6).

$$M = K_T\ x\ I_{line} \quad \text{.....(2-16)}$$

dengan : M = torsi (N.m)

K_T = konstanta torsi BLDC (0.00083)

$RPM = K_v \, x \, V$ .....(2-17)

dengan : RPM = kecepatan putaran per menit

K_v = konstanta BLDC (1200)

V = tegangan masukan

Jika RPM dikonversi dalam satuan kecepatan anguler (ω) maka didapatkan:

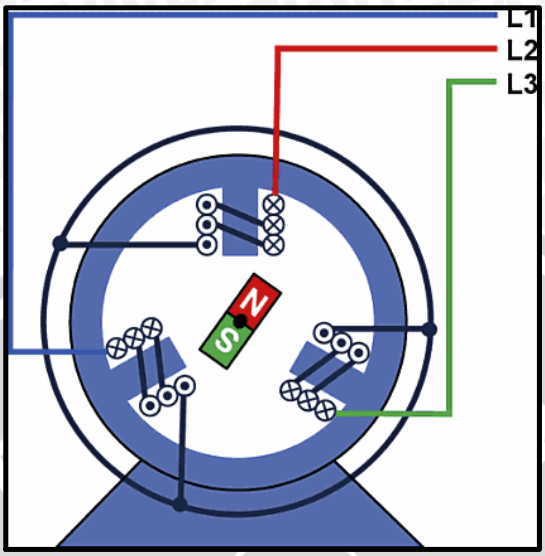
$\omega = \left(\frac{RPM}{60}\right) x \, 2pi$ .....(2-18)

dengan : ω = kecepatan anguler (rad/s)

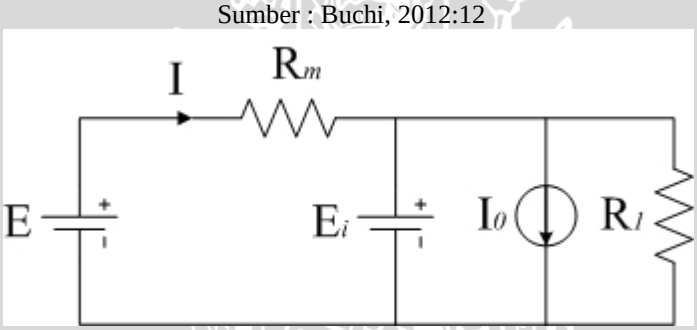
pi = 3.14

RPM = kecepatan putaran per menit

Brushless motor juga bisa dideskripsikan sebagai motor *stepper*. Dengan magnet permanen yang tetap dan mungkin juga dengan beberapa pole pada rotor daripada stator. Versi yang lain mungkin tanpa magnet permanen, hanya pole yang diinduksi pada rotor kemudian ditarik pada lilitan stator. Tapi, kata *stepped motor* biasanya digunakan untuk motor yang didesain secara spesifik pada mode dimana motor tersebut secara berkala berhenti pada posisi anguler. Motor DC *brushless* ditunjukkan dalam Gambar 2.9 di bawah ini:



Gambar 2.9 Skematik *Brushless Motor*
Sumber : Buchi, 2012:12



Gambar 2.10 Rangkaian Ekivalen DC Motor
Sumber : Brown, 2002:6

Sesuai dengan rangkaian ekivalen motor BLDC di atas, dapat ditentukan hubungan (Lance, 2011:3):

$E_i = E - IR_m$ .....(2-19)

dengan : E_i = *back*-EMF (V)

E = tegangan baterai (V)

I = arus baterai (A)

R_m = resistansi *winding* (Ω)

maka kecepatan putaran motor ditentukan melalui:

$K_v = \frac{N}{E_i}$ ..... (2-20)

dengan : K_v = konstanta RPM/volt

N = kecepatan putaran motor (RPM)

E_i = *back*-EMF (V)

nilai R_1 dari rangkaian ekivalen motor BLDC dapat ditentukan melalui:

$R_1 = \frac{E - I(R_m + R_{ESC})}{I - I_0}$ ..... (2-20)

dengan : R_1 = rugi arus Eddy (Ω)

Seperti motor 3 fasa pada umumnya, BLDC motor memliki 2 jenis lilitan pada statornya yaitu Y dan Δ. Hubungan pada nilai torsi antara keduanya ditunjukkan dengan persamaan (Lance, 2011:2):

$M_Y = \sqrt{M_\Delta}$ ..... (2-21)

dengan: M = torsi (N.m)

$\omega_Y = \frac{1}{\sqrt{3}}\omega_\Delta$ ..... (2-22)

dengan : ω = kecepatan anguler (rad/s)

2.6 Mikrokontroler

Mikrokontroler populer yang pertama dibuat oleh Intel pada tahun 1976, yaitu mikrokontroler 8-*bit* Intel 8748. Mikrokontroler tersebut adalah bagian dari keluarga mikrokontroler MCS-48. Sebelumnya, Texas *instruments* telah memasarkan mikrokontroler 4-*bit* pertama yaitu TMS 1000 pada tahun 1974. TMS 1000 yang mulai dibuat sejak 1971 adalah mikrokomputer dalam sebuah *chip*, lengkap dengan RAM dan ROM.

Pengendali mikro (*microcontroller*) adalah sistem mikroprosesor lengkap yang terkandung di dalam sebuah *chip*. Mikrokontroler berbeda dari mikroprosesor serba guna yang digunakan dalam sebuah PC, karena sebuah mikrokontroler umumnya telah berisi komponen pendukung sistem minimal mikroprosesor, yakni memori dan antarmuka I/O.

Berbeda dengan CPU serba-guna, mikrokontroler tidak selalu memerlukan memori eksternal, sehingga mikrokontroler dapat dibuat lebih murah dalam kemasan yang lebih kecil dengan jumlah *pin* yang lebih sedikit.

Sebuah *chip* mikrokontroler umumnya memiliki fitur:

a. *central processing unit* - mulai dari prosesor 4-*bit* yang sederhana hingga prosesor kinerja tinggi 64-*bit*.

b. input/output antarmuka jaringan seperti *port serial* (UART)

c. antarmuka komunikasi serial lain seperti I²C, *Serial Peripheral Interface* and *Controller Area Network* untuk sambungan sistem

d. periferal seperti *timer* dan *watchdog*

e. RAM untuk penyimpanan data

f. ROM, EPROM, EEPROM atau *Flash memory* untuk menyimpan program komputer

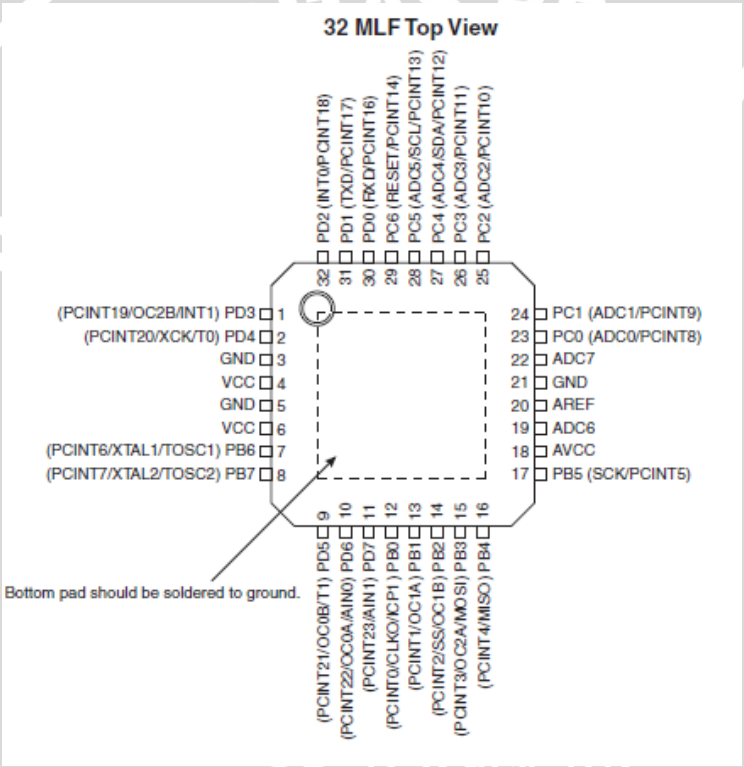
g. pembangkit *clock* - biasanya berupa resonator rangkaian RC

h. pengubah analog-ke-digital

2.6.1 Mikrokontroler ATMega 168-20AU

ATMega 168 AU merupakan seri mikrokontroler CMOS 8-bit buatan Atmel, berbasis arsitektur RISC (*Reduced Instruction Set Computer*). Hampir semua instruksi dieksekusi dalam satu siklus *clock*. ATMega168-20AU mempunyai *throughput* mendekati 1 MIPS per MHz membuat desainer sistem untuk mengoptimasi konsumsi daya *versus* kecepatan proses.

Atmel ATMega 168-20AU memiliki beberapa fitur antara lain 16K *bytes In-system Programmable Flash with Read-While-Write*, 512 *bytes* EEPROM, 1K *bytes* SRAM, 23 jalur I/O untuk tujuan umum, 32 *working registers* untuk tujuan umum, tiga *timer/counter* yang fleksibel dengan *compare mode*, internal dan *external interrupt*, sebuah serial *programmable* USART, sebuah *byte-oriented 2-wire Serial Interface*, sebuah port SPI serial, sebuah *6-channel* 10-bit ADC, sebuah *Watchdog Timer* yang *programmable* dengan internal osilator.



Gambar 2.11 Konfigurasi *Pin* ATMega 168-20AU

Sumber : http://atmel.com

2.6.2 Program CodeVision AVR

CodeVisionAVR merupakan sebuah *cross-compiler* C, *Integrated Development Environtment* (IDE), dan *Automatic Program Generator* yang didesain untuk mikrokontroler buatan Atmel seri AVR. CodeVisionAVR dapat dijalankan pada sistem operasi Windows 95, 98, Me, NT4, 2000, dan XP.

Cross-compiler C mampu menerjemahkan hampir semua perintah dari bahasa ANSI C, sejauh yang diijinkan oleh arsitektur dari AVR, dengan tambahan beberapa fitur untuk mengambil kelebihan khusus dari arsitektur AVR dan kebutuhan pada sistem *embedded*.

File object COFF hasil kompilasi dapat digunakan untuk keperluan *debugging* pada tingkatan C, dengan pengamatan variabel, menggunakan *debugger* Atmel AVR Studio. IDE mempunyai fasilitas internal berupa software AVR Chip In-System Programmer yang memungkinkan pengguna untuk melakukan transfer program kedalam chip mikrokontroler setelah sukses melakukan kompilasi/asembli secara otomatis. *Software* In-System Programmer didesain untuk bekerja dengan Atmel STK500/AVRISP/AVRProg, Kanda Systems STK200+/300, Dontronics DT006, Vogel Elektronik VTEC-ISP, Futurlec JRAVR dan MicroTronics ATCPU/Mega2000 *programmers/development boards*.

Untuk keperluan *debugging* sistem *embedded*, yang menggunakan komunikasi serial, IDE mempunyai fasilitas internal berupa sebuah Terminal.

Selain *library* standar C, CodeVisionAVR juga mempunyai *library* tertentu untuk:

a. Modul LCD alphanumeric
b. Bus I2C dari Philips
c. Sensor Suhu LM75 dari *National Semiconductor*
d. *Real-Time Clock*: PCF8563, PCF8583 dari Philips, DS1302 dan DS1307 dari Maxim/Dallas *Semiconductor*
e. Protokol 1-Wire dari Maxim/Dallas *Semiconductor*
f. Sensor Suhu DS1820, DS18S20, dan DS18B20 dari Maxim/Dallas *Semiconductor*
g. Termometer/Termostat DS1621 dari Maxim/Dallas *Semiconductor*
h. EEPROM DS2430 dan DS2433 dari Maxim/Dallas *Semiconductor*
i. SPI
j. Power Management
k. Delay

l. Konversi ke kode Gray

CodeVisionAVR juga mempunyai *Automatic Program Generator* bernama CodeWizardAVR, yang mengijinkan pengguna untuk menulis, dalam hitungan menit, semua instruksi yang diperlukan untuk membuat fungsi-fungsi berikut:

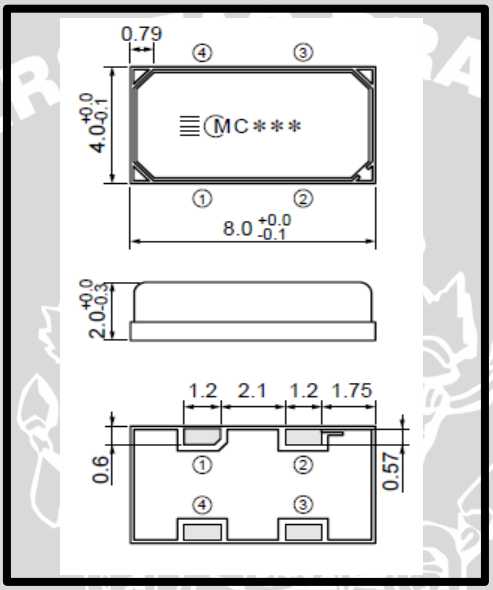
a. *Set-up* akses memori eksternal
b. Identifikasi sumber *reset* untuk *chip*
c. Inisialisasi port *input/output*
d. Inisialisasi interupsi eksternal
e. Inisialisasi *Timer/Counter*
f. Inisialisasi *Watchdog-Timer*
g. Inisialisasi UART (USART) dan komunikasi *serial* berbasis *buffer* yang digerakkan oleh interupsi
h. Inisialisasi Pembanding Analog
i. Inisialisasi ADC
j. Inisialisasi Antarmuka SPI
k. Inisialisasi Antarmuka Two-Wire
l. Inisialisasi Antarmuka CAN
m. Inisialisasi Bus I2C, Sensor Suhu LM75, *Thermometer/Thermostat* DS1621 dan *Real-Time Clock* PCF8563, PCF8583, DS1302, dan DS1307
n. Inisialisasi *Bus 1-Wire* dan Sensor Suhu DS1820, DS18S20
o. Inisialisasi modul LCD

2.7 Sensor VSG (*Vibrating Structure Gyroscope*)

Vibrating structure gyroscope adalah tipe giroskop yang berfungsi seperti *halteres* pada seekor serangga. Prinsip fisik yang mendasarinya adalah bahwa sebuah objek yang bergetar cenderung untuk bergetar pada bidang yang sama. Pada literatur keteknikan, tipe alat seperti ini juga dikenal dengan *Coriolis Vibratory Gyro* (Scarborough, 1958).

Sensor ini terdiri dari 4 buah pin dengan pin 1 adalah *Reference Voltage* (V_{ref}), pin 2 *Ground* (Gnd), pin 3 *Supply Voltage* (V_{cc}) dan pin 4 *Sensor Output* (*Output*). Kecepatan anguler yang mampu dideteksi oleh sensor ini adalah ±300 deg/s, dengan *supply voltage* 2.7-5.25 V. *Output* pada posisi angular 0 adalah 1.35 Vdc dengan skala perubahan 0.67 mV/deg/s.

$\theta = 1.35 + 0.67mV/deg/s$ ..... (2-23)



Gambar 2.12 Sensor *Gyro* Murata ENC-03R
Sumber : http://murata.com

BAB III
METODOLOGI

Kajian dalam skripsi ini adalah penelitian yang bersifat aplikatif, yaitu perencanaan dan pembuatan *Quadrocopter* dengan menitik-beratkan pada pengaturan keseimbangan secara per sumbu (X, Y, Z) yang dikontrol oleh mikrokontroler. Langkah-langkah yang perlu dilakukan untuk mereliasasikan alat yang akan dibuat adalah sebagai berikut :

1. Spesifikasi alat
2. Perancanangan dan realisasi pembuatan alat
3. Pengujian alat
4. Pengambilan kesimpulan

3.1 Spesifikasi Alat

Adapun spesifikasi alat yang akan direalisasikan adalah sebagai berikut:

1. *Quadrocopter* yang digunakan jenis *Quadrcopter* +.
2. *Quadrocopter* memiliki dimensi lengan 26 cm dari ujung lengan menuju *center plate,* jarak antar lengan ialah 90˚ dan menggunakan *acrylic* 5 mm sebagai *center plate*.
3. Mikrokontroler yang digunakan ialah ATMEGA 168-20AU keluaran Atmel.
4. Kontroler yang digunakan ialah kontroler PID (Proporsional, Deferensial dan Integral).
5. Pemrograman menggunakan bahasa pemrograman tingkat ketiga, yaitu C++.
6. Rangkaian mikrokontroler dan *gyro* terletak pada *center plate* sedangkan rangkaian *electronic speed controller* (ESC) terletak pada masing-masing lengan.

7. BLDC (*Brushless Motor DC*) 1200K_v terletak pada setiap ujung lengan, dan menggunakan propeller 10 x 4.5 inchi.

3.2 Perancangan dan Realisasi Pembuatan Alat

3.2.1 Perancangan Perangkat Keras dan Realisasi Pembuatan Alat

a. Pembuatan diagram blok secara lengkap
b. Penentuan dan perhitungan komponen yang akan digunakan
c. Merakit perangkat keras (*hardware*) untuk masing-masing blok.

3.2.2 Perancangan dan Perhitungan Komponen yang akan Digunakan

Setelah mengetahui seperti apa perangkat keras yang dirancang, maka dibutuhkan perangkat lunak untuk mengendalikan dan mengatur kerja dari alat ini. Desain dan parameter yang telah dirancang, kemudian diterapkan pada mikrokontroler dengan menggunakan bahasa pemrograman C++.

3.2.3 Perancangan Perangkat Lunak

Perancangan perangkat lunak dilakukan setelah mengetahui nilai parameter P, I dan D. Perancangan dimulai dari pembuatan *flowchart,* kemudian dilanjutkan dengan penulisan *listing code*.

3.3 Pengujian Alat

Untuk memastikan bahwa sistem ini berjalan sesuai dengan yang direncanakan, maka perlu dilakukan pengujian alat yang meliputi perangkat keras dan perangkat lunak.

1. Pengujian Sensor

   Pengujian sensor dilakukan dengan cara mensimulasikan rangkaian sensor dan hasil pemodelan rangkaian sensor. Pengujian ini bertujuan untuk memastikan sensor dan hasil pemodelan sensor dapat bekerja sesuai dengan perancangan dan memberikan analisis terhadap hasil pengujian. Terdapat

rangkaian sensor utama yang akan diuji, yaitu sensor *gyro* sebagai pengukur keseimbangan *quadrocopter*.

2. Pengujian Karakteristik Motor BLDC

   Pengujian dilakukan dengan menerapkan berbagai macam tegangan masukan pada motor BLDC sehingga motor mengeluarkan putaran yang linier sesuai dengan besar tegangan masukan. Pengujian ini bertujuan untuk memastikan bahwa motor BLDC bekerja sesuai dengan nilai K_v *rating* yang diinginkan.

3. Pengujian Sinyal Kontrol BLDC

   Pengujian dilakukan dengan melihat tegangan keluaran dari kontroler dari sistem. Pengujian ini bertujuan untuk melihat apakah keluaran kontroler sudah sesuai dengan yang direncanakan atau tidak.

4. Pengujian *Thrust*

   Pengujian dilakukan dengan mensimulasikan berbagai macam kecepatan putaran motor yang telah dipasang *propeller* untuk melihat gaya angkat yang dihasilkan. Pengujian ini bertujuan untuk membandingkan *thrust* secara perhitungan dan pengujian.

5. Pengujian Sistem Secara Keseluruhan

   Pengujian ini dilakukan dengan cara menggabungkan semua bagian alat yang dibuat dan melihat kinerja alat. Pengujian ini bertujuan untuk mengetahui kinerja alat yang dibuat dan memberikan analisis terhadap kinerja alat.

3.4 Pengambilan Kesimpulan

Kesimpulan diambil berdasarkan data yang diperoleh dari pengujian sistem secara keseluruhan. Jika hasil yang didapatkan telah sesuai dengan yang direncanakan sebelumnya, maka sistem kendali tersebut telah berhasil memenuhi harapan dan tentunya memerlukan pengembangan lebih lanjut untuk penyempurnaan.

BAB IV
PERANCANGAN DAN PEMBUATAN ALAT

Perancangan dan pembuatan alat ini terdiri dari dua bagian, yaitu perancangan dan pembuatan perangkat keras serta perancangan dan pembuatan perangkat lunak. Perancangan dan pembuatan alat dilakukan secara bertahap untuk memudahkan analisis sistem. Beberapa aspek yang perlu dijelaskan dalam bab ini meliputi penentuan spesifikasi alat, perencanaan masing-masing blok rangkaian serta perencanaan sistem secara keseluruhan.

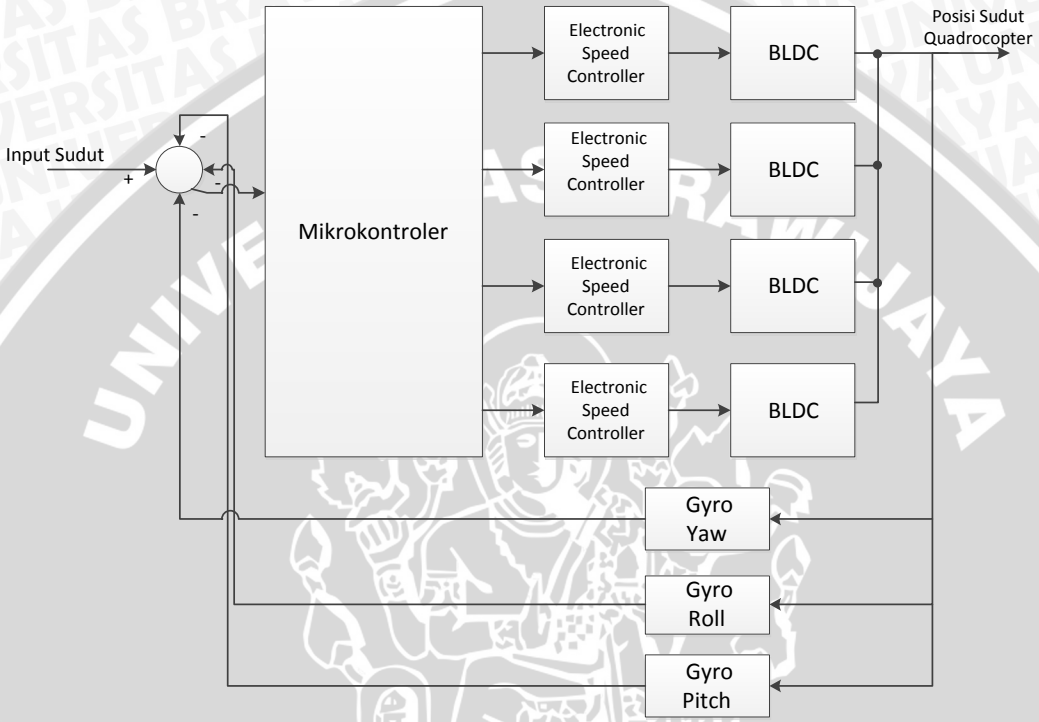
4.1 Spesifikasi Sistem

Spesifikasi sistem yang dirancang adalah sebagai berikut :

1. *Quadrocopter* menggunakan rangka dengan spesifikasi sebagai berikut:
   - Panjang 4 (empat) buah lengan masing-masing 40 cm
   - Lengan *quadrocopter* menggunakan bahan aluminium berbetuk persegi dengan ketebalan 1 mm dan panjang sisi 1.2 cm.
   - Sudut antar lengan berjarak 90˚
   - *Center plate* menggunakan bahan *acrylic* dengan tebal 5 mm sebanyak 2 (dua) buah untuk mengunci bagian atas dan bawah
2. Pergerakan *quadrocopter* menggunakan 4 (empat) buah BLDC (*Brushless Motor DC*) pada masing-masing lengannya.
3. kV *rating* pada BLDC adalah $1200K_v$ dengan maksimum tegangan masukan adalah 11.1 V.
4. *Propeller* pada ujung *shaft* motor menggunakan EPP1045 (10x4.5 inchi).
5. 4 (empat) *Electronic Speed Controller* sebagai pengkondisi sinyal masukan pada BLDC sebesar 30 A.
6. Menggunakan 3 (tiga) buah sensor *gyro*, satu untuk setiap sumbu (*axis*) *yaw*, *pitch* dan *roll*.

7. Mikrokontroler yang digunakan adalah 1 (satu) buah ATMEGA 168-20AU.

4.2 Diagram Blok Sistem



Gambar 4.1 Diagram Blok Sistem *Quadrocopter*
Sumber : Perancangan

Keterangan dari diagram blok di atas adalah:

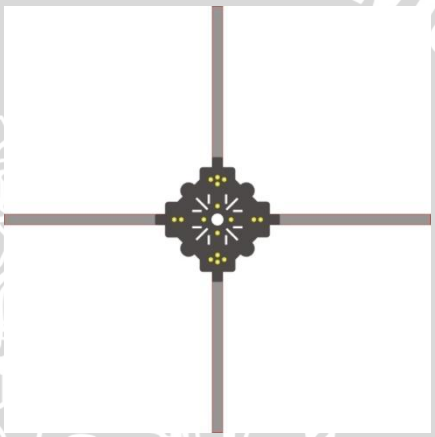
- *Input* berupa sudut di berikan melalui *transmitter* dan kemudian diterima melalui *port receiver* pada mikrokontroler (*throttle, aileron, rudder, elevator*).
- Mikrokontroler kemudian mengolah *input* dan menghasilkan sinyal kontrol yang kemudian akan dikeluarkan menuju ESC.
- ESC kemudian menguatkan sinyal kontrol dan diteruskan menuju BLDC.
- Keluaran sudut (*present value*) kemudian di *feedback* pada tiga buah sensor *gyro*.
- Hasil pembacaan sensor kemudian dikurangkan dengan *input* sehingga mikrokontroler mampu mengkompensasi *error* yang terjadi.

4.3 Perancangan Perangkat Keras

Perangkat keras terdiri dari rangka *quadrocopter*, *board* mikrokontroler, pengkondisi sinyal sensor *gyro*, *electronic speed controller*, BLDC dan *propeller*.

4.3.1 Perancangan Rangka *Quadrocopter*

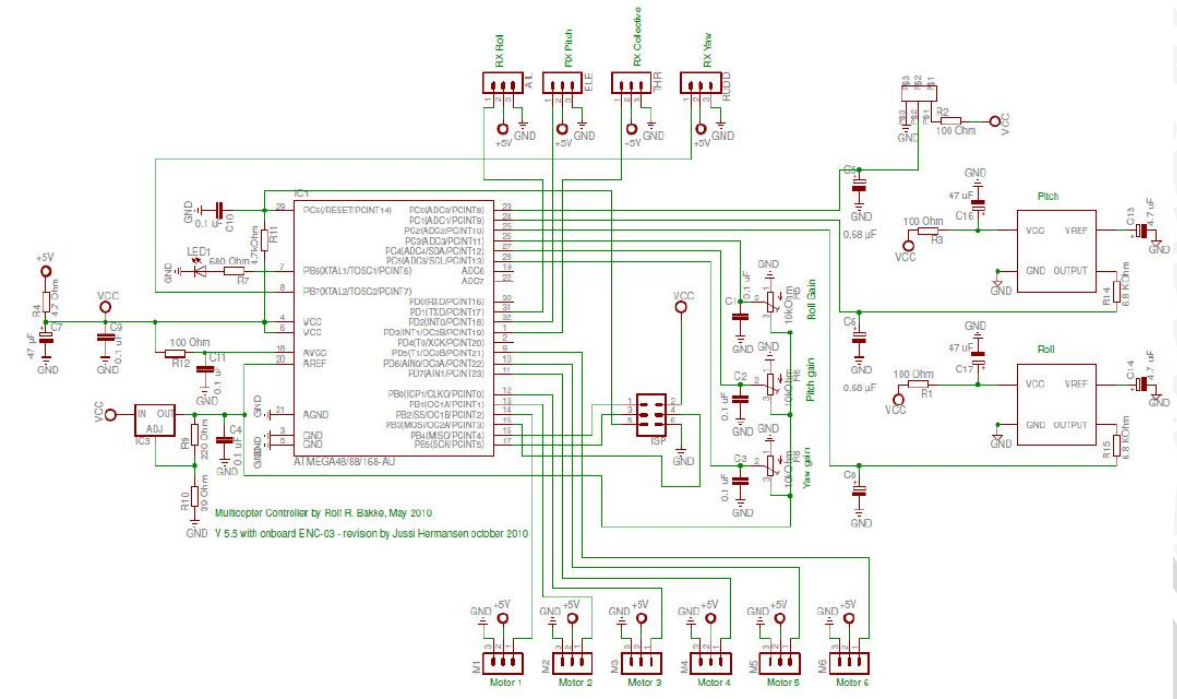
Perancangan rangka dilakukan sebagai dasar dari komponen-komponen yang akan diletakkan. Secara umum rancangan rangka *quadrocopter* ditunjukkan dalam gambar di bawah ini :



Gambar 4.2 *Frame* (rangka) *Quadrocopter*
Sumber : Perancangan

4.3.2 Rangkaian *Board* Mikrokontroler

Board mikrokontroler merupakan rangkaian utama dari sistem *quadrocopter*. Dimana pada rangkaian ini juga berhubungan langsung dengan sensor *gyro*. Pada pengendalian *quadrocopter* digunakan ATMEGA 168-20AU. Rangkaian *board* mikrokontroler ditunjukkan dengan Gambar 4.3:



Gambar 4.3 Rangkaian *Board* Mikrokontroler
Sumber : Perancangan

Mikrokontroler 168-20AU memiliki 32 jalur yang dapat diprogram menjadi keluaran atau masukan. *Pin* masukan dan keluaran mikrokontroler pada perancangan akan difungsikan sesuai tabel 4.1:

Tabel 4.1 Fungsi *Pin* Mikrokontroler

No	*Pin*	Fungsi
1	PC1	Jalur masukan *gyro pitch*
2	PC2	Jalur masukan *gyro roll*
3	PC3	Jalur masukan potensiometer *gain roll*
4	PC4	Jalur masukan potensiometer *gain pitch*
5	PC5	Jalur masukan potensiometer *gain yaw*
6	PD1	Jalur masukan Rx *roll*
7	PD2	Jalur masukan Rx *pitch*
8	PD3	Jalur masukan Rx *collective*
9	PB7	Jalur masukan Rx *yaw*
10	PD7	Jalur keluaran motor 4
11	PB0	Jalur keluaran motor 3
12	PB1	Jalur keluaran motor 2
13	PB2	Jalur keluaran motor 1

14	PC0	Jalur masukan *gyro yaw*

Sumber : Perancangan

4.3.3 *Electronic Speed Controller*

Electronic speed controller berfungsi sebagai penguat tegangan dari keluaran *pin* motor dan juga berfungsi sebagai pemecah fasa. Karena BLDC yang digunakan merupakan motor DC dengan 3 fasa dengan hubungan *star*.

4.3.4 Pemilihan BLDC dan *Propeller*

Kombinasi antara BLDC dan *propeller* harus tepat agar mampu menghasilkan daya angkat (*thrust*) yang sesuai dengan berat keseluruhan *quadrocopter* (termasuk *battery*), BLDC dan komponen-komponen lainnya. Perhitungan mengenai *thrust* yang dihasilkan sebuah *propeller* didasarkan pada persamaan 4.1 berikut ini:

$$T = cT \frac{4\rho r^4}{\pi^2} \omega^2 \quad \text{.....(4.1)}$$

dimana : T = *thrust propeller* (N)

cT = koefisien *propeller* (0.1154 untuk EPP 1045)

ρ = kerapatan udara (1.225 kg/m^3)

ω = kecepatan anguler *propeller* (rad/s)

r = jari-jari *propeller* (m)

Sesuai dengan spesifikasi sistem yang dijabarkan sebelumnya, bahwa pada skripsi ini menggunakan EPP 1045 dengan BLDC 1200kV. EPP 1045 memiliki diameter 25.4 cm. Sementara BLDC 1200K$_v$ mampu menghasilkan putaran hingga 12000 rpm dengan tegangan masukan 10 V. Tabel 4.1 menunjukkan *thrust* yang dihasilkan setiap kenaikan kecepatan BLDC.

Tabel 4.2 Perhitungan *Thrust*

RPM	RPS (rad/s)	*Thrust* (N)	*Thrust* x 4 (N)
600	62.83	0.0588	0.235
1200	125.66	0.2353	0.9414
1800	188.49	0.5295	2.1182
2400	251.32	0.9414	3.7657

3000	314.15	1.4710	5.8840
3600	376.99	2.1182	8.47303
4200	439.82	2.8831	11.5327
4800	502.65	3.7657	15.0631
5400	565.48	4.7660	19.0643 ▶ RPM Minimum
6000	628.31	5.8840	23.5362
6600	691.15	7.1197	28.4788
7200	753.98	8.4730	33.8921
7800	816.81	9.9440	39.7762
8400	879.64	11.5327	46.1309
9000	942.47	13.2391	52.9564
9600	1005.30	15.0631	60.2527
10200	1068.14	17.0049	68.0196
10800	1130.97	19.0643	76.2573
11400	1193.80	21.2414	84.9657
12000	1256.63	23.5362	94.1448

Sumber : Perancangan

Jika berat maksimal *quadrocopter* mencapai 1.8 Kg, maka perhitungan gaya berat *quadrocopter* adalah :

$$F = ma \quad \text{.....(4.2)}$$

dengan : F = gaya berat (N)

m = massa benda (Kg)

a = gravitasi bumi (9,81 m/s^2)

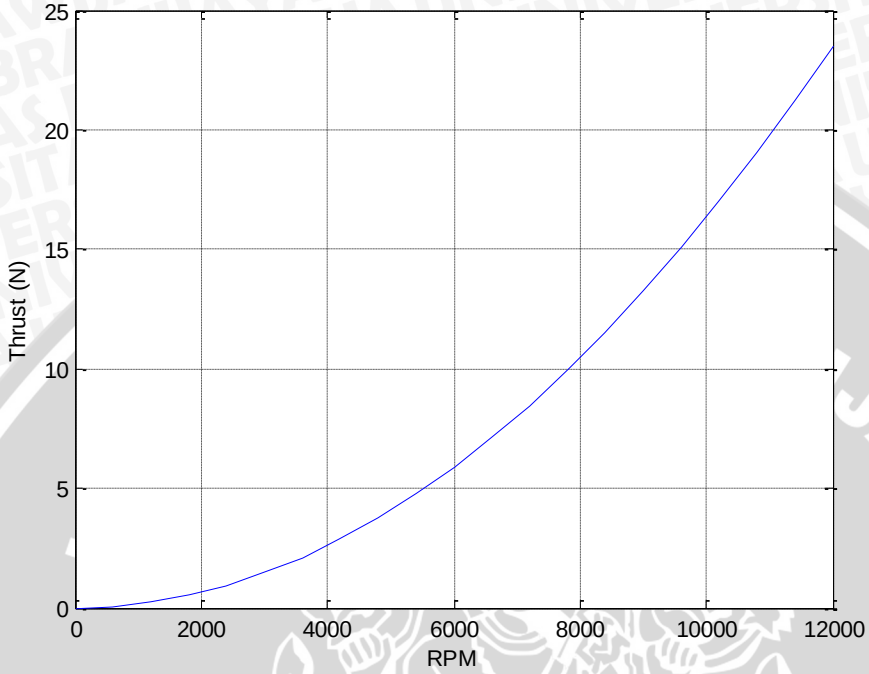
maka :

$F = 1.8 \text{ Kg} \times 9.81 \text{ m/s}^2$

$= 17.568 \text{ N}$

hal ini berarti *thrust* minimal yang harus dihasilkan oleh keempat *propeller* adalah 17.658 N. Sesuai dengan perhitungan pada Tabel 4.1, pada kecepatan 5400 RPM menghasilkan *thrust* 19.04 N. Maka dapat dipastikan bahwa pada kecepatan tersebut,

thrust yang dihasilkan sudah mampu membuat *quadrocopter* terbang dalam kondisi



Gambar 4.4 Hubungan *Thrust* dan RPM EPP1045
Sumber : Perancangan

hoover. Gambar 4.4 menunjukkan hubungan linier antara RPM dan *thrust* pada EPP 1045.

Gambar 4.5 *Propeller* EPP1045 CW dan CCW
Sumber : Perancangan

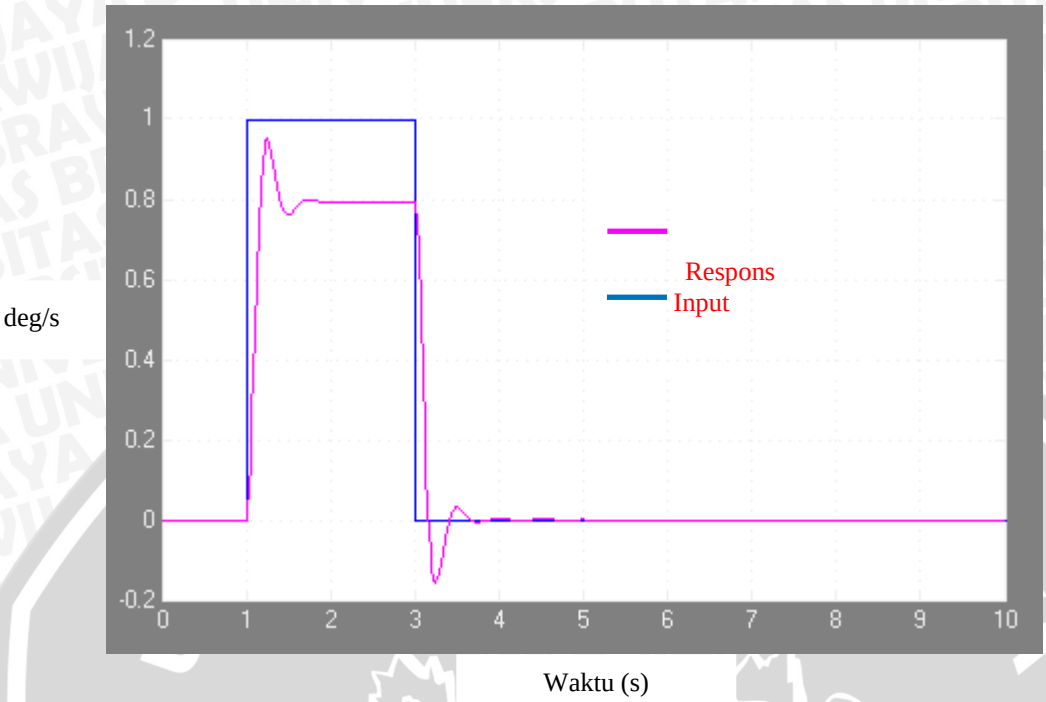
Pada skripsi ini juga menggunakan BLDC dengan K_v *rating* 1200 dengan masukan tegangan maksimum 11.1 volt. Hal ini berarti BLDC mampu menghasilkan putaran hingga 13320 RPM atau setara dengan 1394.86 rad/s dan cukup untuk memenuhi nilai putaran minimal untuk melakukan gaya angkat.

Gambar 4.6 BLDC 1200kV
Sumber : Perancangan

4.4 Penetuan Nilai Penguatan Kontroler

Perancangan kontroler dilakukan dengan menggunakan program dalam bentuk m-*file* pada *software* MATLAB R2011a. Simulasi dilakukan dengan memberikan sinyal pulsa dengan rentang waktu 2 (dua) detik dan kecepatan anguler = 1 (satu) deg/s. Pengubahan nilai penguatan kontroler dilakukan dengan metode *hand tuning* tetapi dengan penetuan awal nilai kontroler menggunakan metode kedua Ziegler-Nichols.

Langkah pertama yang dilakukan adalah dengan melihat respons sistem tanpa kontroler seperti gambar di bawah ini:



Gambar 4.7 Respons Sistem Tanpa Kontroler

Sumber : Perancangan

untuk dapat menyederhanakan sistem menjadi sistem orde 1, sesuai dengan persamaan:

$$\frac{1}{1+Ts} \quad \text{.....(4-3)}$$

dengan : T (tau) = (63.2% Tr + 33% Tf) x 0.5

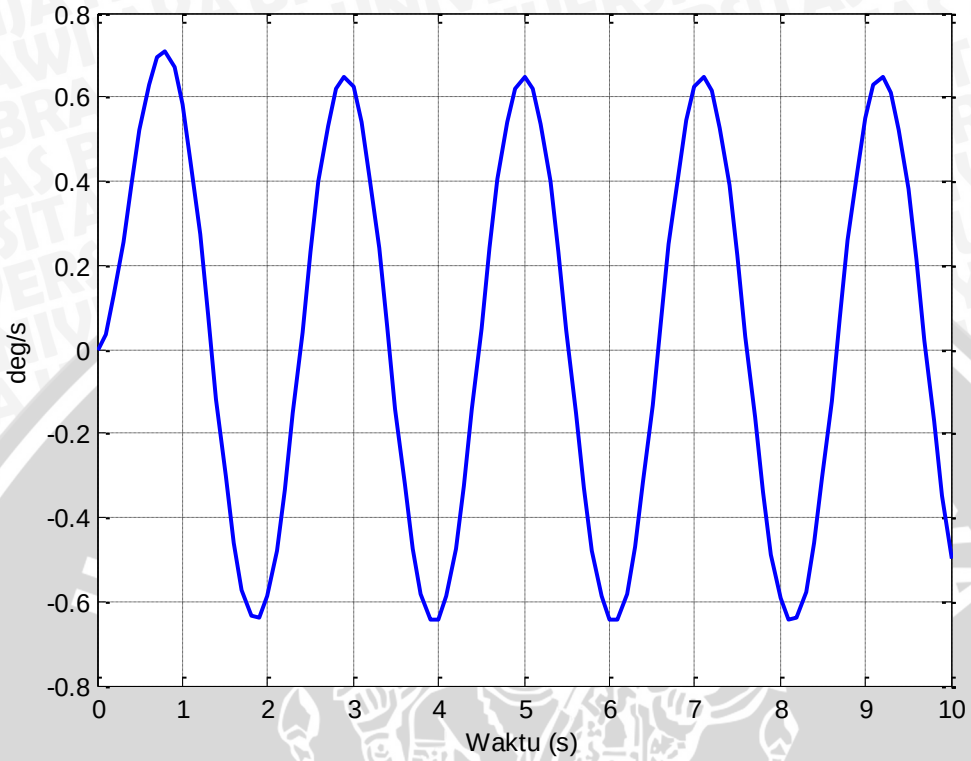
Tr = waktu naik

Tf = waktu turun

Dari respons sistem tanpa kontroler di atas, didapatkan nilai Tr = 0.5056 s dan nilai Tf = 0.2640 s. Maka nilai fungsi alih sistem orde satunya adalah :

$$\frac{1}{1 + 0.38483\,s}$$

Dengan mengaplikasikan metode kedua Ziegler-Nichols untuk fungsi alih sistem orde 1 tersebut, pada sebuah sistem *loop* tertutup dengan penguatan K_{cr} sebesar 3, maka didapatkan respons:



Gambar 4.8 Osilasi Berkesinambungan dengan Periode Pcr

Sumber: Perancangan

Berdasarkan Gambar 4.8, didapatkan nilai P_{cr} sebesar 2.1. Untuk dapat menetukan nilai parameter kontroler PID, harus berdasarkan tabel aturan dasar Ziegler-Nichols berdasarkan *critical gain* K_{cr} dan *critical period* P_{cr} di bawah ini:

Tabel 4.3 Aturan Dasar Ziegler-Nichols Berdasarkan *Critical Gain Kcr* dan *Critical Period Pcr*

Tipe Kontroler	K_p	T_i	Td
P	0.5 Kcr	∞	0
PI	0.45 Kcr	$\frac{1}{1,2}$ Pcr	0
PID	0.60 Kcr	0.5 Pcr	0.125 Pcr

Sumber : Ogata, 1997

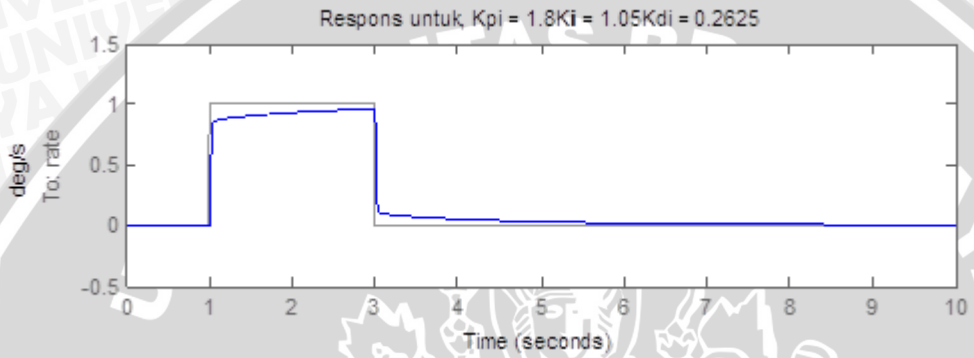
maka nilai parameter PID adalah :

P = 0.6 x 3 = 1.8

I = 0.5 x 2.1 = 1.05

D = 0.125 x 2.1 = 0.2625

dengan mengaplikasikan nilai-nilai parameter PID di atas, maka didapatkan respons sistem sperti di bawah ini:



Gambar 4.9 Respon Sistem untuk Nilai PID Berdasarkan Ziegler-Nichols

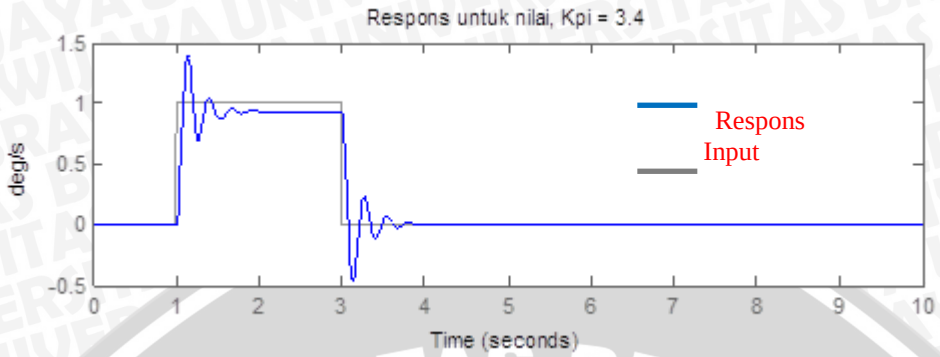
Sumber : Perancangan

Dari respons sistem dengan parameter PID menurut Ziegler-Nichols metode kedua, dapat dilihat bahwa sistem masih memiliki *offset*, sehingga nilai parameter PID perlu diubah-ubah secara manual (*hand tuning*).

Tabel 4.4 Penguatan Kp yang Berbeda

Kp	***Error Steady State* (%)**
1.8	13.1
2.0	11.3
3.4	7.6

Sumber : Perancangan



Gambar 4.10 Respons untuk nilai Kp = 3.4

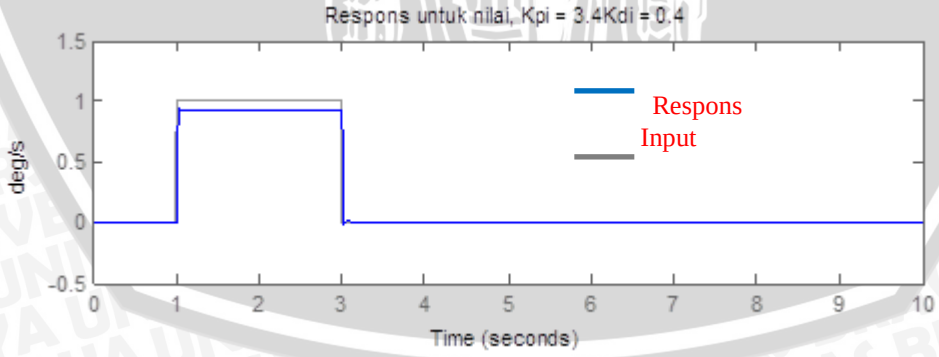
Sumber : Perancangan

Dari gambar respons untuk Kp = 3.4 di atas masih terdapat *overshoot*. Untuk menghilangkan *overshoot*, maka nilai peguatan Kd perlu dinaikkan. Performansi sistem yang dilihat pada penguatan Kd adalah nilai *overshoot*.

Tabel 4.5 Penguatan Kd yang Berbeda

Kp	Kd	*Overshoot* (%)
3.4	0.3	0.75
3.4	0.4	0

Sumber : Perancangan



Gambar 4.11 Respons Untuk Kp = 3.4 dan Kd = 0.4

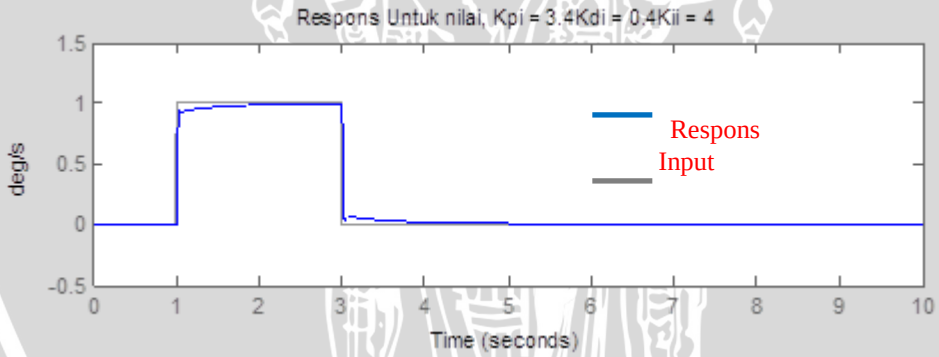
Sumber : Perancangan

Dari Gambar 4.8 di atas dapat dilihat bahwa respons sistem masih memiliki *offset*. Untuk menghilangkan *offset* yang terjadi, maka perlu menaikkan nilai Ki. Karena pada kontroler dengan aksi kontrol integral, nilai masukan kontroler u(t) diubah pada laju proporsional dari sinyal pembangkit kesalahan e(t).

Tabel 4.7 Penguatan Ki yang Berbeda

Ki	Kp	Kd	*Error Steady State (%)*
1.5	3.4	0.4	3.2
2.0	3.4	0.4	2.5
2.5	3.4	0.4	1.7
3.0	3.4	0.4	1.3
3.5	3.4	0.4	0.5
4	3.4	0.4	0.2

Sumber : Perancangan

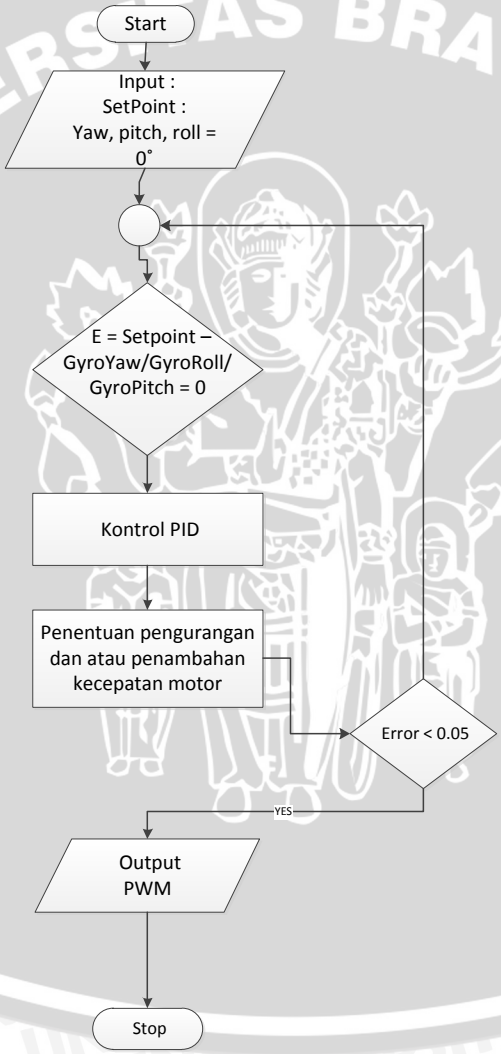


Gambar 4.12 Respons Untuk Kp = 3.4, Kd = 0.4 dan Ki = 4

Dari penentuan nilai penguatan Kp, Ki dan Kd di atas dapat dipastikan nilai penguatan yang digunakan untuk sistem *quadrocopter* adalah Kp = 3.4, Ki = 4 dan Kd = 0.4.

4.5 Perancangan Perangkat Lunak

Perancangan perangkat lunak pada skripsi ini menggunakan bahasa pemrograman C++ dengan menggunakan *software* CodeVision AVR. *Tuning* kontroler PID adalah dengan menggunakan metode *hand tuning*. *Flowchart* perancangan perangkat lunak ditunjukkan pada Gambar 4.11 berikut:



Gambar 4.13 *Flowchart* Perangkat Lunak
Sumber : Perancangan

BAB V
PENGUJIAN DAN ANALISIS

Tujuan pengujian sistem ini adalah untuk menentukan apakah alat yang telah dibuat berfungsi dengan baik dan sesuai dengan perancangan. Pengujian pada sistem ini meliputi pengujian setiap blok maupun pengujian secara keseluruhan. Pengujian setiap blok ini dilakukan untuk menemukan letak kesalahan dan mempermudah analisis pada sistem apabila alat tidak bekerja sesuai dengan perancangan. Pengujian dibagi menjadi beberapa bagian, yaitu:

1. Pengujian sensor *gyro*
2. Pengujian karakteristik motor BLDC
3. Pengujian sinyal kontrol BLDC
4. Pengujian *thrust*
5. Pengujian sistem secara keseluruhan

5.1 Pengujian Sensor *Gyro*

Pengujian ini bertujuan untuk mengetahui keberhasilan rangkaian sensor *gyro* dalam mendeteksi perubahan sudut pada sumbunya. Untuk pengujian respon rangkaian sensor *gyro* digunakan fasilitas *Hyperterminal* pada Windows XP untuk melihat data keluaran sensor.

5.1.1 Peralatan Pengujian

1. Sensor *gyro*
2. *Converter* USB *to* RS232 PL2303
3. Laptop dengan OS (*Operating System*) Windows XP
4. Beberapa buah kabel
5. MATLAB R2011 Rev.B

5.1.2 Prosedur Pengujian

Pengujian dilakukan dengan menghubungkan keempat buah *pin* keluaran untuk motor dengan *pin* Rx pada *board* mikrokontroler. Kemudian *pin* untuk sinyal pada M6 dihubungkan dengan *pin* Rx pada RS232 dengan *pin ground*-nya juga dihubungkan pada *pin ground* M6. USB kemudian dihubungkan dengan salah satu port USB pada laptop. Untuk melihat perubahan yang ditunjukkan oleh sensor *gyro*, *board* mikrokontroler harus diubah posisi sudutnya. Data yang di dapatkan lalu kemudian diplot dengan MATLAB untuk melihat linearitas sensor.

5.1.3 Hasil Pengujian

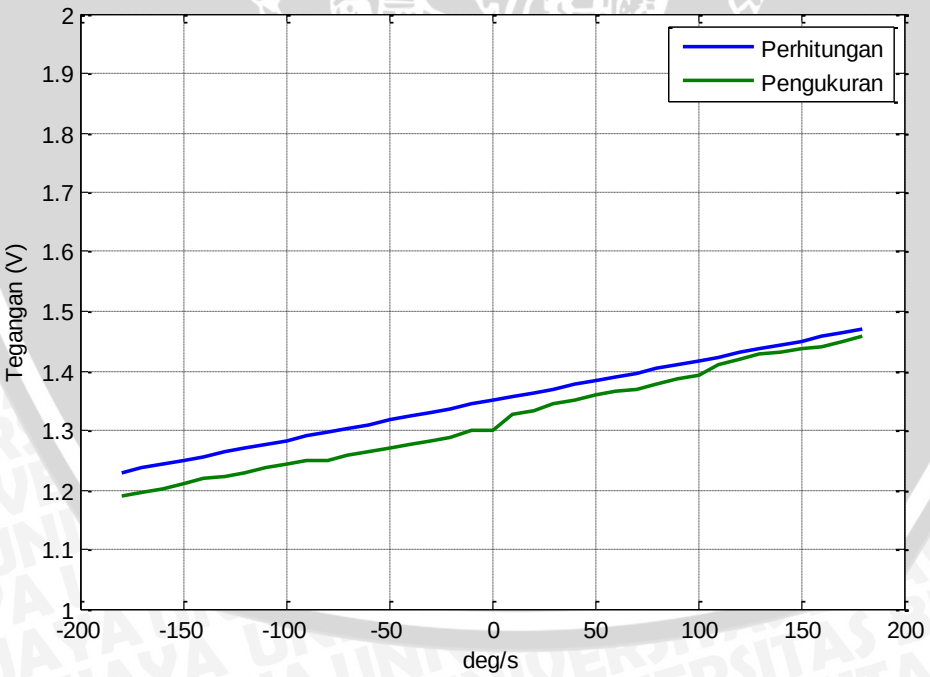
Setelah melakukan prosedur pengujian dengan tegangan perhitungan berdasarkan persamaan (2-13), maka akan didapatkan hasil yang ditunjukkan pada tabel dan gambar di bawah ini:

Tabel 5.1 Hasil Perhitungan dan Pengukuran Sensor *Gyro*

Kecepatan sudut (deg/s)	Tegangan Perhitungan (V)	Tegangan Terukur (V)	Error (%)
0	1.3500	1.3002	0.036
10	1.3567	1.3266	0.022
20	1.3634	1.3333	0.022
30	1.3701	1.3459	0.017
40	1.3768	1.3509	0.018
50	1.3835	1.3596	0.017
60	1.3902	1.367	0.016
70	1.3969	1.3701	0.019
80	1.4036	1.378	0.018
90	1.4103	1.3851	0.017
100	1.417	1.3925	0.017
110	1.4237	1.41	0.009
120	1.4304	1.419	0.007
130	1.4371	1.428	0.006
140	1.4438	1.431	0.008
150	1.4505	1.437	0.009
160	1.4572	1.44	0.011
170	1.4639	1.4501	0.009
180	1.4706	1.457	0.009

0	1.350	1.3002	0.036
-10	1.3433	1.299	0.032
-20	1.3366	1.2894	0.034
-30	1.3299	1.283	0.034
-40	1.3232	1.277	0.034
-50	1.3165	1.2711	0.033
-60	1.3098	1.2642	0.033
-70	1.3031	1.258	0.033
-80	1.2964	1.25	0.034
-90	1.2897	1.2498	0.029
-100	1.2830	1.2421	0.030
-110	1.2763	1.2359	0.029
-120	1.2696	1.2293	0.029
-130	1.2629	1.222	0.030
-140	1.2562	1.2194	0.027
-150	1.2495	1.2093	0.029
-160	1.2428	1.2019	0.030
-170	1.2361	1.1960	0.029
-180	1.2294	1.1889	0.030

Sumber : Pengujian



Gambar 5.1 Respon Sensor *Gyro* CW dan CCW

Sumber : Pengujian

Dari hasil pengujian yang dilakukan pada *gyro* dapat dilihat bahwa *gyro* yang digunakan memiliki kelinieran yang baik, sehingga ideal untuk digunakan sebagai pendeteksi kecepatan sudut dari *quadrocopter*.

Untuk meyakinkan hal ini, maka perlu dilakukan perhitungan tingkat kepercayaan terhadap data yang ada. Karena besarnya sampel adalah 19 atau kurang dari 30, maka digunakanlah uji-T untuk pengujian statistik.

Tabel 5.2 Statistik Uji-T untuk Pengujian Sensor *Gyro*

TEGANGAN (V)						
PENGUKURAN (x)	PERHITUNGAN (μ_0)	x rata-rata	μ_0 rata-rata	s	n	t
1.3002	1.35	1.38849	1.41030	0.045903	119	-2.0710
1.3266	1.3567					
1.3333	1.3634					
1.3459	1.3701					
1.3509	1.3768					
1.3596	1.3835					
1.367	1.3902					
1.3701	1.3969					
1.378	1.4036					
1.3851	1.4103					
1.3925	1.417					
1.41	1.424					
1.419	1.430					
1.428	1.437					
1.431	1.444					
1.437	1.451					
1.44	1.457					
1.4501	1.464					
1.457	1.471					

Sumber : Pengujian

$$t = \frac{\overline{x} - \mu_o}{s/\sqrt{n}} \quad \text{.....(5-1)}$$

dengan:

μ_o = rata-rata data hasil perhitungan

$\overline{x}$ = rata-rata sampel hasil pengukuran

S = standar deviasi sampel

N = jumlah sampel

V = derajat bebas = n-1

α = tingkat signifikansi

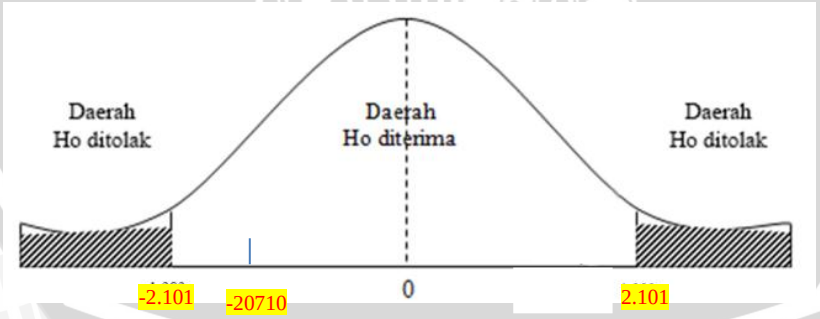
Statistik uji:

$$t = \frac{\overline{x} - \mu_o}{s/\sqrt{n}}$$

$$\frac{1.38849 - 1.41030}{\frac{0.045903}{\sqrt{19}}}$$

= -2.0710

Tingkat signifikansi (α) 0,05 maka α/2 adalah 0,025 dan derajat bebas *v* = 18, maka dari tabel distribusi t diperoleh -$t_{\alpha/2.v}$ dan $t_{\alpha/2.v}$ adalah -2.101 dan 2.101. Jika dibandingkan dengan t hitung, maka t hitung (t= -2.0710) berada di antara angka-angka tersebut, sehingga Ho diterima. Oleh karena dapat diambil keputusan tingkat kepercayaan 95%.



Gambar 5.2 Perbandingan Nilai t Tabel dan t Hitung

5.2 Pengujian Karakteristik Motor BLDC

Pengujian ini bertujuan untuk mengetahui tingkat kelinieran kecepatan motor BLDC 1200kV dengan tegangan masukan yang diberikan. Tegangan diberikan melalui ESC dengan masukan ESC merupakan sinyal PWM dari mikrokontroler.

5.2.1 Peralatan Pengujian

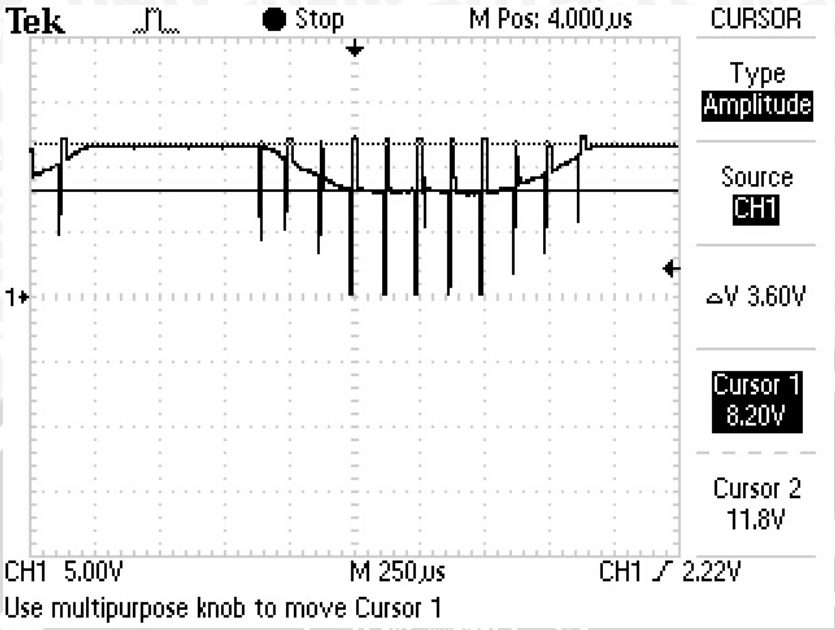
1. *Borad* Mikrokontroler
2. Osiloskop
3. ESC
4. BLDC Motor
5. *Tachometer*

5.2.2 Prosedur Pengujian

Pengujian dilakukan dengan memberikan tegangan melalui ESC dengan nilai PWM divariasi mulai dari 10% sampai dengan 100%. Salah satu *line* masukan dari BLDC dihubungkan dengan osiloskop dengan bagian luar motor diukur kecepatannya dengan menggunakan *tachometer*.

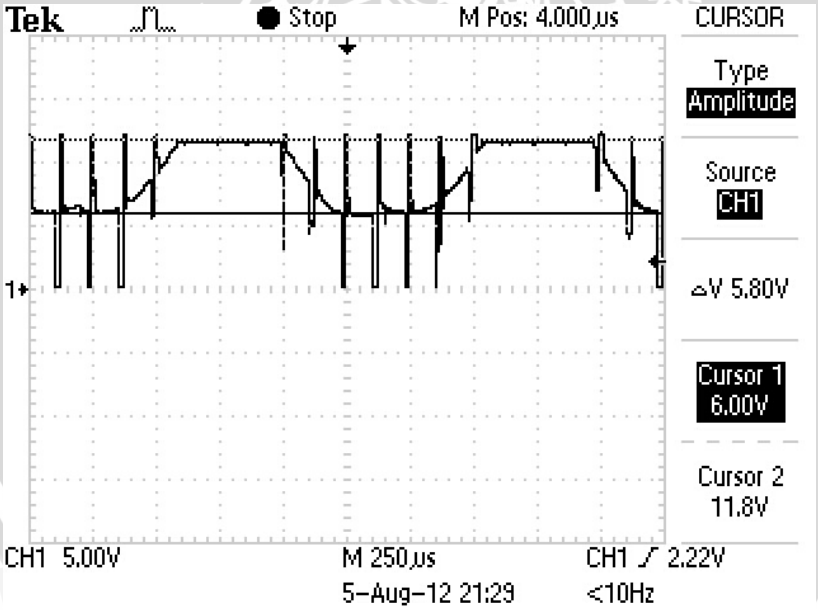
5.2.3 Hasil Pengujian

Dari prosedur pengujian yang dilakukan maka didapatkan hasil seperti dibawah ini:



Gambar 5.3 Sinyal Masukan dengan Tegangan 3.60V pada Motor BLDC

Sumber :Pengujian



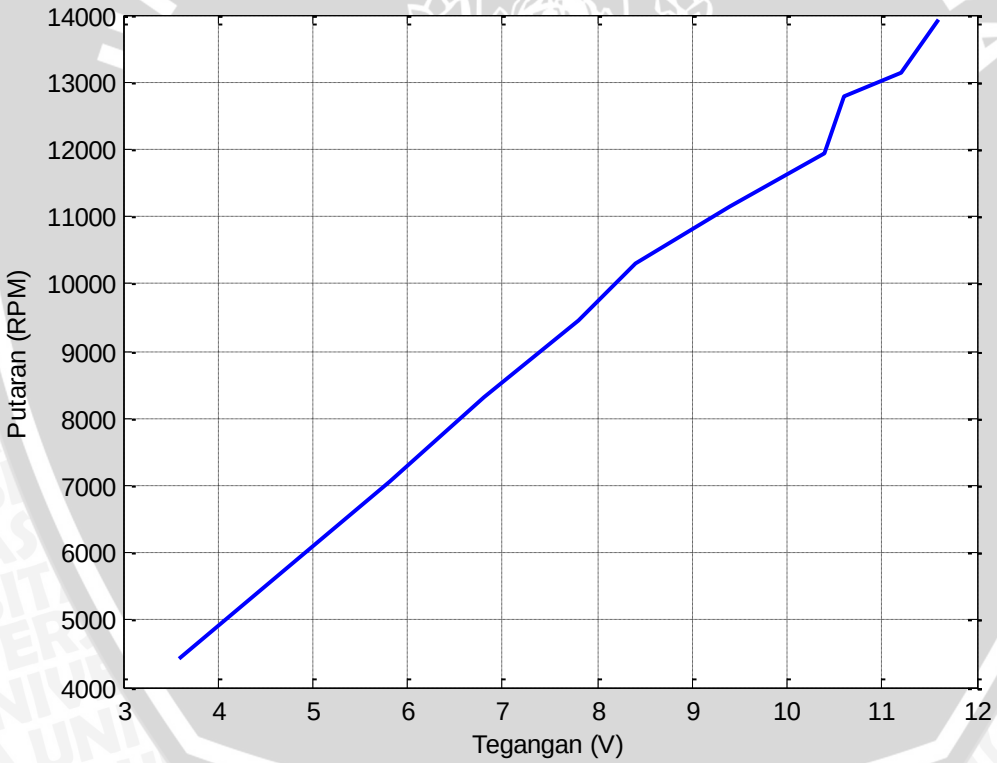
Gambar 5.4 Sinyal Masukan dengan Tegangan 5.80V pada Motor BLDC

Sumber : Pengujian

Tabel 5.3 Hubungan Antara PWM, Tegangan ESC dan Kecepatan Motor

Tegangan	Kecepatan Motor (RPM)
3.6	4424
5.8	7060
6.8	8300
7.8	9460
8.4	10293
9.4	11150
10.4	11950
10.6	12800
11.2	13150
11.6	13932

Sumber : Pengujian



Gambar 5.5 Grafik Hubungan Tegangan dan RPM BLDC 1200K_v

Sumber : Pengujian

5.3 Pengujian Sinyal Kontrol BLDC

Pengujian ini bertujuan untuk mengetahui ketepatan lebar sinyal *high* yang dihasilkan oleh mikrokontroler. Lebar sinyal *high* mempengaruhi kecepatan putaran motor BLDC.

5.3.1 Peralatan Pengujian

1. *Board* mikrokontroler
2. Osiloskop
3. BLDC
4. ELAB
5. Battery Li-Po 11.1 V
6. *Transmitter*

5.3.2 Prosedur Pengujian

Pengujian dilakukan dengan menghubungkan pin keluaran motor BLDC pada *board* mikrokontroler dengan osiloskop dan ELAB. Perubahan lebar pulsa pada pin keluaran motor dapat dihasilkan dengan menaikkan atau menurunkan *stick* pada *transmitter*.

Untuk pengujian dengan menggunakan ELAB, empat *pin* keluaran motor dihubungkan sekaligus pada ELAB, dengan konfigurasi *channel* sebagai berikut:

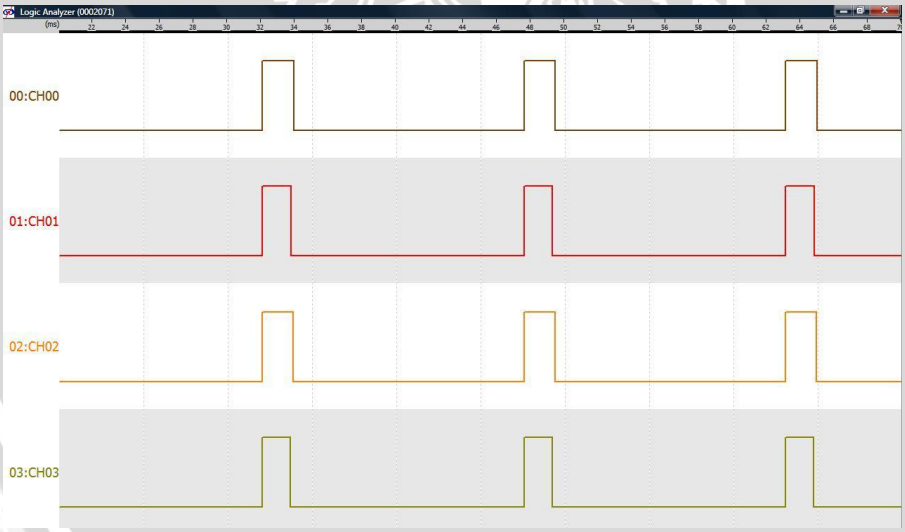
- *Channel 00* / CH00 = Motor 1
- *Channel 01* / CH01 = Motor 2
- *Channel 02* / CH02 = Motor 3
- *Channel 03* / CH03 = Motor 4

5.3.3 Hasil Pengujian

Setelah melakukan prosedur pengujian, maka akan didapatkan hasil yang ditunjukkan pada gambar-gambar dibawah ini.

Gambar 5.6 Sinyal PWM Waktu Kondisi Diam

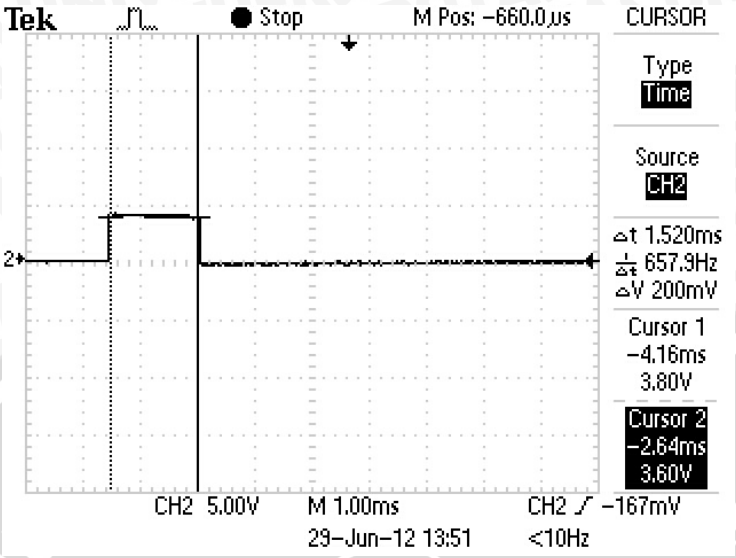
Sumber : Pengujian



Gambar 5.7 Sinyal PWM Waktu Kondisi Berputar

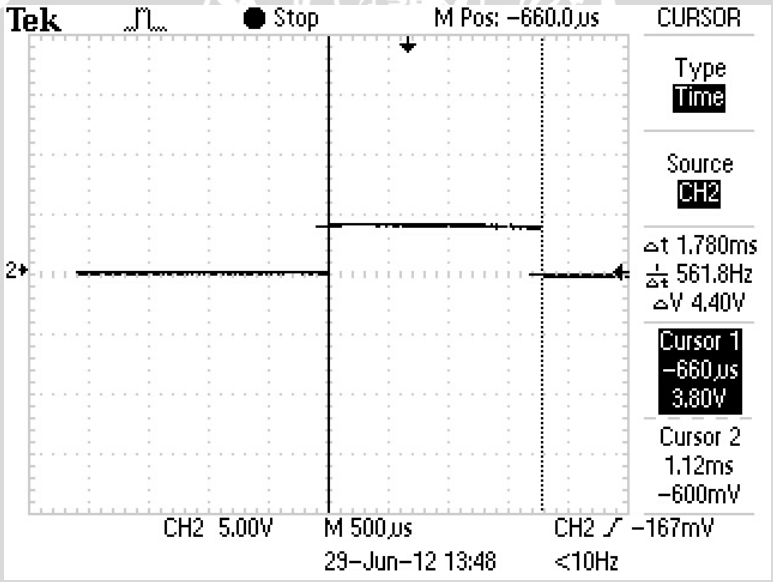
Sumber : Pengujian

Pada kedua gambar di atas, dapat dilihat bahwa mikrokontroler menghasilkan sinyal PWM secara bersamaan. Artinya bahwa keempat motor BLDC bekerja secara bersamaan.



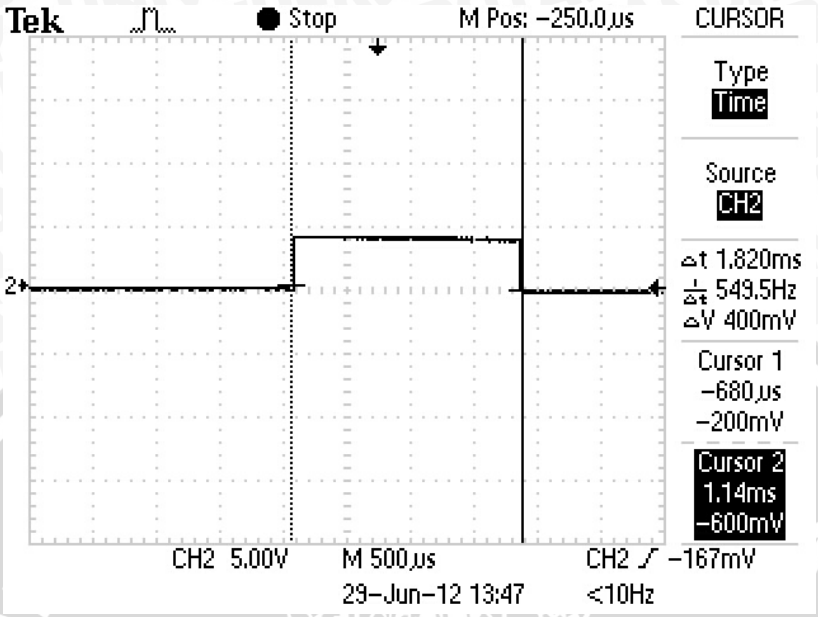
Gambar 5.8 Sinyal Kontrol dengan Lebar Pulsa *High* 1520ms

Sumber : Pengujian



Gambar 5.9 Sinyal Kontrol dengan Lebar Pulsa *High* 1780ms

Sumber : Pengujian



Gambar 5.10 Sinyal Kontrol dengan Lebar Pulsa *High* 1820ms

Sumber : Pengujian

Berdasarkan hasil pengujian sinyal kontrol, dapat disimpulkan bahwa rangkaian *board* mikrokontroler ATMEGA 168-20AU dapat mengeluarkan sinyal kontrol dengan baik dan sesuai dengan sistem yang direncanakan.

5.4 Pengujian *Thrust*

Pengujian ini bertujuan untuk melihat *thrust* yang dihasilkan oleh kombinasi *propeller* dan BLDC. Pengujian ini juga bertujuan untuk melihat perbandingan antara *thrust* dari perhitungan secara teori dan dari pengambilan data pada saat pengujian.

5.4.1 Peralatan Pengujian

1. *Board* mikrokontroler
2. Timbangan *digital*
3. Motor *test bench*
4. Li-Po *Battery* 11.1 V
5. BLDC 1200kV

6. *Propeller* EPP 1045
7. *Transmitter*
8. *Tachometer*

5.4.2 Prosedur Pengujian

Pengujian dilakukan dengan memasang motor BLDC yang sudah terpasang dengan *propeller* EPP1045 pada motor *test bench*. *Test Bench* kemudian diletakkan pada bagian atas timbangan *digital*. Untuk melihat perubahan *thrust* yang terjadi, *stick* pada *transmitter* dinaikkan atau diturunkan.

5.4.3 Hasil Pengujian

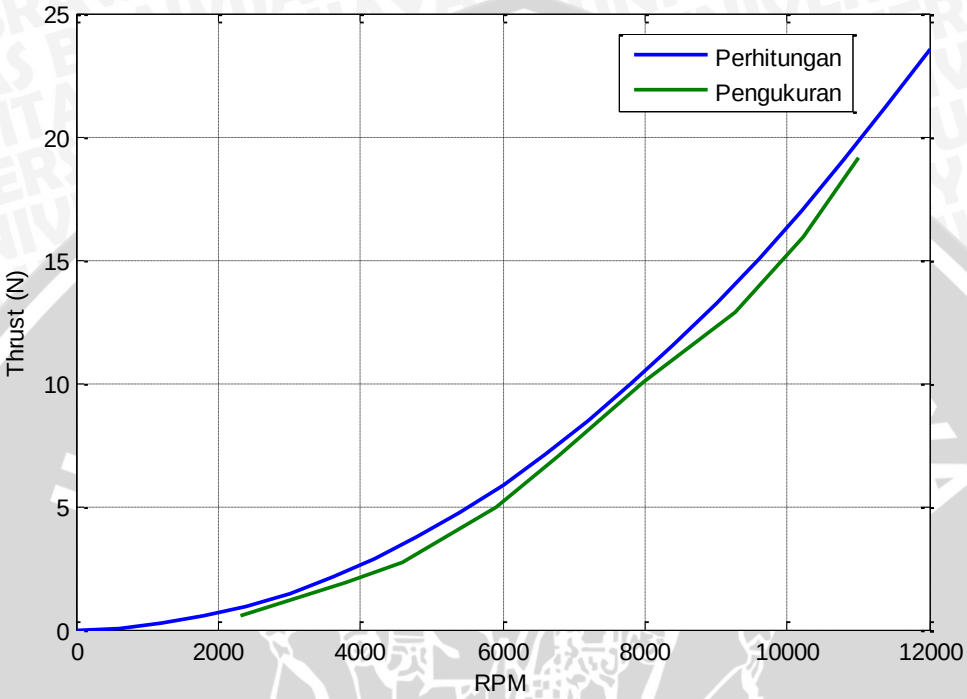
Setelah melakukan pengujian, maka didapatkan hasil seperti dibawah ini:

Tabel 5.4 Hasil Pengujian *Thrust*

No.	RPM	*Thrust Pengukuran* (N)	*Thrust Perhitungan (N)*
1.	2314	2.132	0.665
2.	3781	3.433	1.959
3.	4590	4.694	2.967
4.	5900	7.113	5.157
5.	6788	9.992	7.124
6.	7991	12.33	10.001
7.	9267	14.556	13.42
8.	10239	18.783	16.958
9.	11006	22.75	20.145
10.	12045	23.235	20.860

Sumber : Pengujian

Dengan membandingkan hasil pengujian di atas dengan hasil perhitungan teori sesuai dengan persamaan (4-1), maka didapatkan hasil seperti di bawah ini:



Gambar 5.11 Grafik Perbandingan *Thrust* Perhitungan dan Hasil Pengujian

Sumber : Pengujian

Untuk meyakinkan hal ini, maka perlu dilakukan perhitungan tingkat kepercayaan terhadap data yang ada. Karena besarnya sampel adalah 10 atau kurang dari 30, maka digunakanlah uji-T untuk pengujian statistik.

Tabel 5.5 Statistik Uji-T untuk Pengujian *Propeller*

THRUST (N)						
PENGUKURAN (x)	PERHITUNGAN (μ_0)	x rata-rata	μ_0 rata-rata	s	n	t
2.132	0.665	11.90180	9.92560	7.556487	110	0.8270104
3.433	1.959					
4.694	2.967					
7.113	5.157					
9.992	7.124					

12.33	10.001					
14.556	13.42					
18.783	16.958					
22.75	20.145					
23.235	20.86					

Sumber : Pengujian

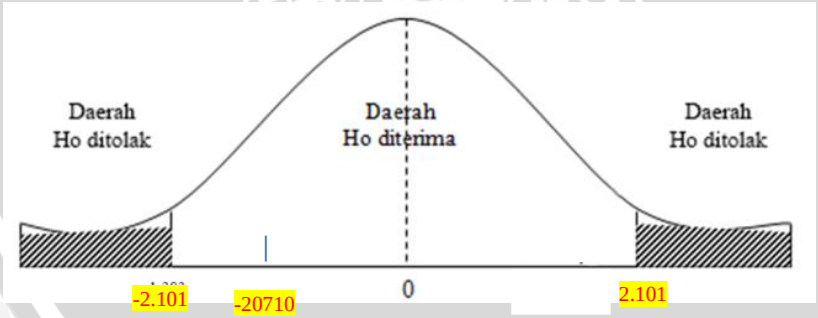
Statistik uji:

$$t = \frac{\overline{x} - \mu_o}{s/\sqrt{n}}$$

$$= \frac{11.90180 - 9.92560}{\frac{7.556487}{\sqrt{10}}}$$

= 0.8270104

Tingkat signifikansi (α) 0,40 maka $\alpha/2$ adalah 0,20 dan derajat bebas *v* = 9, maka dari tabel distribusi t diperoleh -$t_{\alpha/2.v}$ dan $t_{\alpha/2.v}$ adalah -0.883 dan 0.883. Jika dibandingkan dengan t hitung, maka t hitung (t= 0.8270104) berada di antara angka-angka tersebut, sehingga Ho diterima. Oleh karena dapat diambil keputusan tingkat kepercayaan 60%.



Gambar 5.12 Perbandingan Nilai t Tabel dan t Hitung

5.5 Pengujian Keseluruhan

Pengujian ini bertujuan untuk melihat respon kesetimbangan sistem *quadrocopter* secara keseluruhan yang mencakup gerak rotasional *yaw*, *pitch*, *roll*.

5.5.1 Peralatan Pengujian

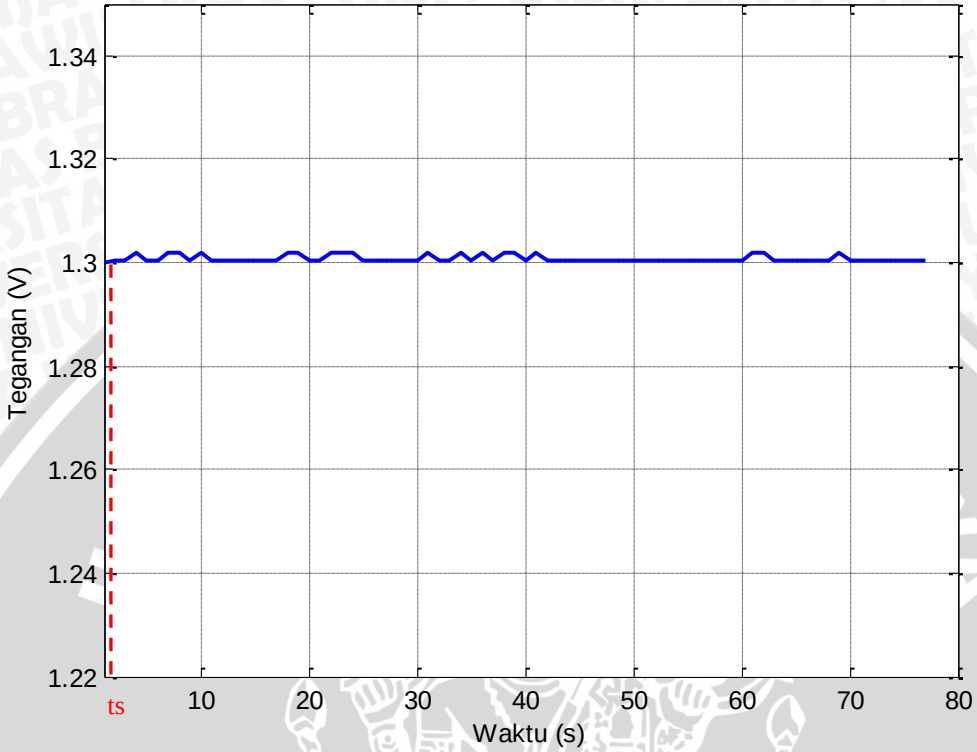
1. *Converter* USB *to* RS232 PL2303
2. Komputer
3. Minimum Sistem ATMega8
4. Beberapa buah kabel

5.5.2 Prosedur Pengujian

Quadrocopter digantungkan dengan senar sehingga dapat melihat respon pengujian secara per-*axis*. Untuk dapat melihat respon per-*axis*, pin *output* dari *gyro* dihubungkan dengan *pin* adc dari mikrokontroler ATMega8. Dari ATMega 8, *pin* Tx dan Rx dihubungkan dengan *pin* Tx dan Rx pada RS232. Kemudian dihubungkan pada komputer secara serial.

5.5.3 Hasil Pengujian

Setelah melakukan prosedur pengujian, maka didapatkan hasil seperti gambar-gambar di bawah ini.



Gambar 5.13 Grafik Respons Kesetimbangan *Yaw Axis*

Sumber:Pengujian

Dari gambar 5.8 di atas dapat diketahui performansi sistem sebagai berikut:

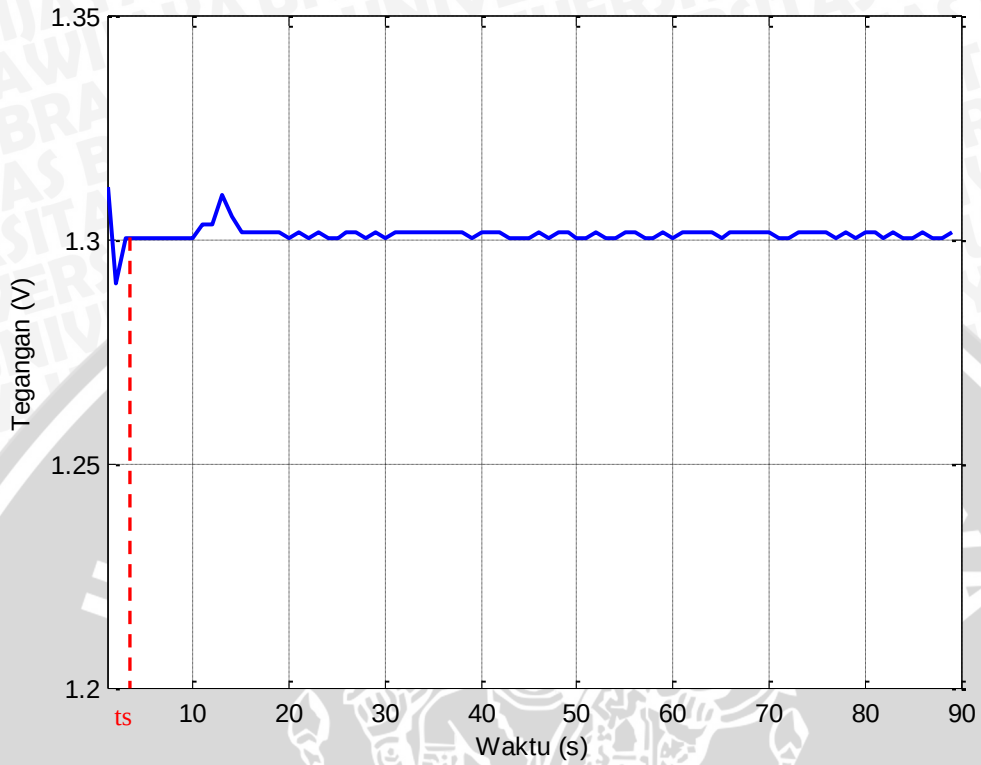
a. *Time Settling* (ts)

Dari gambar 5.8 dapat dilihat pada saat *start* awal *quadrocopter* sudah pada kondisi setimbang pada sumbu *yaw* sehingga ts = 1s

b. *Error Steady State* (ESS)

nilai osilasi maksimum pada gambar 5.8 adalah 1.302 dengan titik *setpoint* adalah 1.3, maka:

$$ESS = \frac{1.302 - 1.3}{1.3} \; x \; 100\%$$
$$= 0.15\ \%$$



Gambar 5.14 Grafik Respons Kesetimbangan *Roll Axis*

Sumber : Pengujian

Dari gambar 5.9 di atas dapat diketahui performansi sistem sebagai berikut:

a. *Time Settling* (ts)

Dari gambar 5.9 dapat dilihat bahwa nilai *time settling* adalah 3s.

b. *Error Steady State* (ESS)

Nilai osilasi maksimum pada gambar 5.9 adalah 1.302 dengan *setpoint* adalah 1.3, maka:

$$ESS = \frac{1.302 - 1.3}{1.3}\ x\ 100\%$$

$= 0.15\ \%$

c. *Maximum Overshoot* (Mp)

Nilai tertinggi dari gambar 5.9 adalah 1.312, maka:

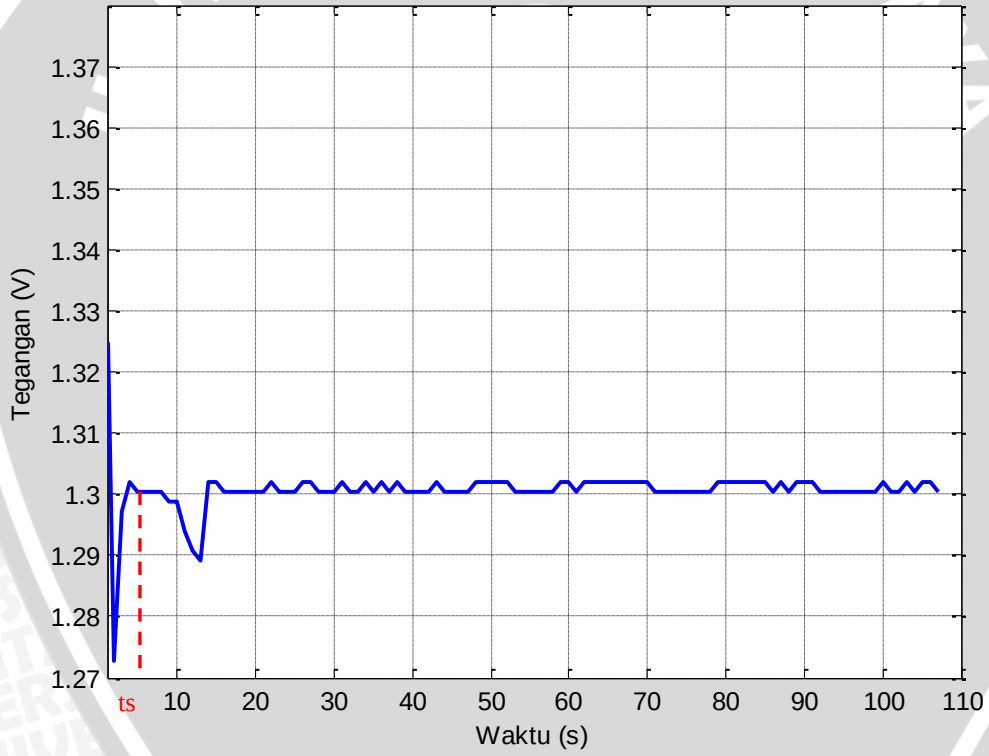
$$Mp = \frac{1.312 - 1.3}{1.3} \; x \; 100\%$$

$$= 0.9231 \; \%$$

sedangkan nilai terendahnya adalah 1.291, maka:

$$Mp = \frac{1.291 - 1.3}{1.3} \; x \; 100\%$$

$$= -0.6923 \; \%$$



Gambar 5.15 Grafik Respons Kesetimbangan *Pitch Axis*

Sumber : Pengujian

Dari gambar 5.10 di atas dapat diketahui performansi sistem sebagai berikut:

a. Time Settling (ts)

Time settling pada respons kesetimbangan *pitch axis* adalah 5s.

b. *Error Steady State* (ESS)

Nilai osilasi maksimum pada grafik 5.10 adalah 1.302 dengan *setpoint* adalah 1.3, maka :

$$ESS = \frac{1.302 - 1.3}{1.3} \; x \; 100\%$$

$= 0.15\ \%$

c. *Maximum Overshoot* (Mp)

Nilai tertinggi dari gambar 5.10 adalah 1.325, maka:

$$Mp = \frac{1.325 - 1.3}{1.3} \; x \; 100\%$$

$$= 1.9231\ \%$$

sedangkan nilai terendah dari gambar 5.10 adalah 1.289, maka:

$$Mp = \frac{1.289 - 1.3}{1.3} \; x \; 100\%$$

$$= -\ 0.8462\ \%$$

BAB VI
PENUTUP

6.1 Kesimpulan

Dari perancangan, pengujian dan pengamatan yang telah dilakukan pada *quadrocopter,* maka dapat diambil kesimpulan sebagai berikut :

1. Dengan menggunakan algoritma PID, sistem *quadrocopter* mampu menaikkan/menurunkan kecepatan motor BLDC 1 dan 3 untuk pergerakan *pitch*, menaikkan/menurunkan kecepatan motor BLDC 2 dan 4 untuk pergerakan *roll*, menaikkan kecepatan motor BLDC 2 dan 4 serta menurunkan kecepatan motor BLDC 1 dan 3 untuk pergerakan *yaw*. Parameter PID ditentukan dengan menggunakan metode *hand tuning* dan didapatkan nilai Kp = 3.4, Ki = 4 dan Kd = 0.4. Dengan menggunakan parameter tersebut sistem mampu mempertahankan posisi setimbang masing-masing sumbu pada sudut rotasional 0˚.
2. *Quadrocopter* yang dirancang memiliki respon sistem sesuai dengan yang direncanakan dengan panjang lengan masing-masing 40cm dan sudut antar lengan 90˚. *Quadrocopter* menggunakan sensor *gyro* sebagai *feedback* sistem dan BLDC motor 1200 kV sebagai aktuator. Gaya dorong dihasilkan melalui perpaduan kecepatan putaran motor dan *propeller* EPP1045 dengan *thrust* maksimal mencapai 20.860 N.

6.2 Saran

Dalam perancangan dan pembuatan alat ini masih terdapat beberapa kelemahan. Untuk memperbaiki kinerja *quadrocopter* dan pengembangan lebih lanjut disarankan :

1. Penyempuranaan sistem *feedback* dengan menggunakan IMU (*Inertial Measurement Unit*) yang merupakan gabungan antara *gyroscope*,

accelerometer dan *magnetometer*. Hal ini penting agar posisi sudut rotasional bisa presisi.

2. Penambahan data *logger* agar mampu merekam data pada saat benar-benar terbang, sehingga dapat menganalisa sistem *quadrocopter*.
3. *Frame* diganti dengan serat karbon agar bisa lebih ringan dan kuat.
4. Algoritma PID perlu digabungkan dengan kalman filter dengan untuk sistem yang lebih *robust*.

DAFTAR PUSTAKA

Astrom, K.J, & Hagglund, Tore. 1995. *PIDControllers: Theory, Design and Tuning*. Research Triangle Park. Instrument Society of America.

Atmel Corporation. 2011. *ATMEGA 168 Series.*

Bresciani, Tomasso. 2008. *Modelling, Identification and Controlling of Quadrotor Helicopter*. Lund. Department of Automatic Control (Lund University).

Brown, Ward. 2002. *Brushless DC Motor Control Made Easy.* Microchip Technology. Inc.

Buchi, Roland. 2012. *Brushless Motors and Controllers.* Norderstedt. Books on Demand.

Domingues, Jorge. 2009. *Quadrotor Prototype*. Lisbon. Instituto Superior Tecnico, Universidade Tecnica de Lisboa.

Gunterus, Frans. 1977. *Falsafah Dasar : Sistem Pengendalian Proses*. Jakarta. Elex Media Komputindo.

Luukonen, Teppo. 2011. *Modelling and Control of Quadcopter*. Espoo.

Murata Manufacturing.,Co.Ltd. *Piezoelectric Vibrating Gyroscope ENC Series*.

Ogata, Katsuhiko. 1997. *Teknik Kontrol Automatik*. Jakarta. Penerbit Erlangga.

Scarborough, J.B. 1958. *The Gyroscope Theory and Applications*. New York. Interscience Publishers, Inc.

Vincent, David. 2010. *Development of an Aerial Test Bed.* Massey University.

LAMPIRAN 1

Gambar Alat

Tampak Samping

Tampak Atas

LAMPIRAN 2

Datasheet Komponen

LAMPIRAN 3

Program Keseluruhan

```
#define QUAD_COPTER
//#define QUAD_X_COPTER
#define ESC_RATE 450   // in Hz
#define STICK_THROW 300
#define GYRO_GAIN_SHIFT 5
#define STICK_GAIN_SHIFT 8
#define MAX_COLLECTIVE 1000                    // 95
#define PWM_LOW_PULSE_US ((1000000 / ESC_RATE) - 2000)
#define ADC_MAX 1023
#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include "typedefs.h"
#include "io_cfg.h"
#define EEPROM_DATA_START_POS 0                    enum GyroDirection { GYRO_NORMAL
= 0, GYRO_REVERSED };
enum GyroArrayIndex { ROLL = 0, PITCH, YAW };
// eeProm data structure
static struct config {
       uint8_t setup;
       uint8_t RollGyroDirection;
       uint8_t PitchGyroDirection;
       uint8_t YawGyroDirection;
} Config;
bool Armed;

static uint16_t GainInADC[3];                  // hasil ADC
static uint16_t RxChannel1;
static uint16_t RxChannel2;
static uint16_t RxChannel3;
static uint16_t RxChannel4;
register uint16_t i_tmp asm("r2");          // ISR vars
register uint16_t RxChannel1Start asm("r4");
register uint16_t RxChannel2Start asm("r6");
register uint16_t RxChannel3Start asm("r8");
register uint16_t RxChannel4Start asm("r10");
register uint8_t i_sreg asm("r12");


static int16_t gyroADC[3];
static int16_t gyroZero[3];
static int16_t integral[3];                    // PID integral
static int16_t last_error[3];                  // proporsional

static uint16_t ModeDelayCounter;

static int16_t MotorOut1;
static int16_t MotorOut2;
static int16_t MotorOut3;
static int16_t MotorOut4;
static int16_t MotorOut5;
static int16_t MotorOut6;
static void setup();
static void init_adc();
static void ReadGyros();
static void CalibrateGyros();

static void ReadGainPots();
static void RxGetChannels();
static void read_adc(uint8_t channel);
static void output_motor_ppm();
static void Initial_EEPROM_Config_Load();
```

```
static void Save_Config_to_EEPROM();
static void Set_EEPROM_Default_Config();
static void eeprom_write_byte_changed( uint8_t *  addr, uint8_t value);
static  void  eeprom_write_block_changes(  const  uint8_t  *  src,  void  *  dest,  size_t
size);
#if 1
ISR(PCINT2_vect, ISR_NAKED)
{
	if (RX_ROLL) {                          // rising
		asm volatile("lds %A0, %1" : "=r" (RxChannel1Start) : "i" (&TCNT1L));
		asm volatile("lds %B0, %1" : "=r" (RxChannel1Start) : "i" (&TCNT1H));
		asm volatile("reti");
	} else {                                // falling
		asm volatile(
				"lds %A0, %3\n"
				"lds %B0, %4\n"
				"in %1, __SREG__\n"
				"sub %A0, %A2\n"
				"sbc %B0, %B2\n"
				"out __SREG__, %1\n"
					: "+r" (i_tmp), "+r" (i_sreg), "+r" (RxChannel1Start)
					: "i" (&TCNT1L), "i" (&TCNT1H));
		RxChannel1 = i_tmp;
	}
	asm volatile ("reti");
}

ISR(INT0_vect, ISR_NAKED)
{
	if (RX_PITCH) {                                 // rising
		asm volatile("lds %A0, %1" : "=r" (RxChannel2Start) : "i" (&TCNT1L));
		asm volatile("lds %B0, %1" : "=r" (RxChannel2Start) : "i" (&TCNT1H));
		asm volatile("reti");
	} else {                                // falling
		asm volatile(
				"lds %A0, %3\n"
				"lds %B0, %4\n"
				"in %1, __SREG__\n"
				"sub %A0, %A2\n"
				"sbc %B0, %B2\n"
				"out __SREG__, %1\n"
					: "+r" (i_tmp), "+r" (i_sreg), "+r" (RxChannel2Start)
					: "i" (&TCNT1L), "i" (&TCNT1H));
		RxChannel2 = i_tmp;
	}
	asm volatile ("reti");
}

ISR(INT1_vect, ISR_NAKED)
{
	if (RX_COLL) {                          // rising
		asm volatile("lds %A0, %1" : "=r" (RxChannel3Start) : "i" (&TCNT1L));
		asm volatile("lds %B0, %1" : "=r" (RxChannel3Start) : "i" (&TCNT1H));
		asm volatile("reti");
	} else {                                // falling
		asm volatile(
				"lds %A0, %3\n"
				"lds %B0, %4\n"
				"in %1, __SREG__\n"
				"sub %A0, %A2\n"
				"sbc %B0, %B2\n"
				"out __SREG__, %1\n"
					: "+r" (i_tmp), "+r" (i_sreg), "+r" (RxChannel3Start)
					: "i" (&TCNT1L), "i" (&TCNT1H));
		RxChannel3 = i_tmp;
	}
```

```
        asm volatile ("reti");
}
ISR(PCINT0_vect, ISR_NAKED)
{
        if (RX_YAW) {                           // rising
                asm volatile("lds %A0, %1" : "=r" (RxChannel4Start) : "i" (&TCNT1L));
                asm volatile("lds %B0, %1" : "=r" (RxChannel4Start) : "i" (&TCNT1H));
                asm volatile("reti");
        } else {                                // falling
                asm volatile(
                        "lds %A0, %3\n"
                        "lds %B0, %4\n"
                        "in %1, __SREG__\n"
                        "sub %A0, %A2\n"
                        "sbc %B0, %B2\n"
                        "out __SREG__, %1\n"
                                : "+r" (i_tmp), "+r" (i_sreg), "+r" (RxChannel4Start)
                                : "i" (&TCNT1L), "i" (&TCNT1H));
                RxChannel4 = i_tmp;
        }
        asm volatile ("reti");
}

#else
ISR(PCINT2_vect)
{
        if (RX_ROLL) { // rising edge
                RxChannel1Start = TCNT1;
        } else {       // falling edge
                RxChannel1 = TCNT1 - RxChannel1Start;
                i_sreg = 0;
        }
}
ISR(INT0_vect)
{
        if (RX_PITCH) {
                RxChannel2Start = TCNT1;
        } else {
                RxChannel2 = TCNT1 - RxChannel2Start;
                i_sreg = 0;
        }
}

ISR(INT1_vect)
{
        if (RX_COLL) {
                RxChannel3Start = TCNT1;
        } else {
                RxChannel3 = TCNT1 - RxChannel3Start;
                i_sreg = 0;
        }
}
ISR(PCINT0_vect)
{
        if (RX_YAW) {
                RxChannel4Start = TCNT1;
        } else {
                RxChannel4 = TCNT1 - RxChannel4Start;
                i_sreg = 0;
        }
}
#endif

static void setup()
{
        uint8_t i;
```

```
        MCUCR = _BV(PUD);
#if 0
        RX_ROLL_DIR     = INPUT;
        RX_PITCH_DIR    = INPUT;
        RX_COLL_DIR     = INPUT;
        RX_YAW_DIR      = INPUT;

        GYRO_YAW_DIR    = INPUT;
        GYRO_PITCH_DIR  = INPUT;
        GYRO_ROLL_DIR   = INPUT;
        GAIN_YAW_DIR    = INPUT;
        GAIN_PITCH_DIR  = INPUT;
        GAIN_ROLL_DIR   = INPUT;

        M1_DIR          = OUTPUT;
        M2_DIR          = OUTPUT;
        M3_DIR          = OUTPUT;
        M4_DIR          = OUTPUT;
        M5_DIR          = OUTPUT;
        M6_DIR          = OUTPUT;

        LED_DIR         = OUTPUT;

        LED             = 0;
        RX_ROLL         = 0;
        RX_PITCH        = 0;
        RX_COLL         = 0;
        RX_YAW          = 0;
#else
        DDRB = 0b01111111;
        DDRC = 0b11000000;
        DDRD = 0b11110001;
#endif


        if (OSCCAL == 0x9d)
                OSCCAL = 0x9f;


        TCCR0B = _BV(CS00);


        TCCR1B = _BV(CS10);

        TCCR2B = _BV(CS22) | _BV(CS21) | _BV(CS20);

        PCICR = _BV(PCIE0) | _BV(PCIE2);
        PCMSK0 = _BV(PCINT7);
        PCMSK2 = _BV(PCINT17);
        EICRA = _BV(ISC00) | _BV(ISC10);
        EIMSK = _BV(INT0) | _BV(INT1);


#endif

        Initial_EEPROM_Config_Load();

        init_adc();

        Armed = false;

        /*
         * Flash the LED once at power on
         */
        LED = 1;
```

```
_delay_ms(150);
LED = 0;

sei();

_delay_ms(1500);

ReadGainPots();
ReadGainPots();

// clear config
if (GainInADC[PITCH] < (ADC_MAX * 5) / 100 &&
    GainInADC[ROLL]  < (ADC_MAX * 5) / 100 &&
    GainInADC[YAW]   < (ADC_MAX * 5) / 100) {

        Set_EEPROM_Default_Config();
        while (1)
                ;
}

// Stick Centering Test
if (GainInADC[PITCH] < (ADC_MAX * 5) / 100) {
        while (1) {
                RxGetChannels();
                i = abs(RxInRoll) + abs(RxInPitch) + abs(RxInYaw);
                LED = 1;
                while (i) {
                        LED = 0;
                        i--;
                }
        }
}

// Gyro direction reversing
if (GainInADC[ROLL] < (ADC_MAX * 5) / 100) {
        // flash LED 3 times
        for (i = 0;i < 3;i++) {
                LED = 1;
                _delay_ms(25);
                LED = 0;
                _delay_ms(25);
        }

        while (1) {
                RxGetChannels();

                if (RxInRoll < -STICK_THROW) {        // normal(left)
                        Config.RollGyroDirection = GYRO_NORMAL;
                        Save_Config_to_EEPROM();
                        LED = 1;
                } if (RxInRoll > STICK_THROW) {       // reverse(right)
                        Config.RollGyroDirection = GYRO_REVERSED;
                        Save_Config_to_EEPROM();
                        LED = 1;
                } else if (RxInPitch < -STICK_THROW) { // normal(up)
                        Config.PitchGyroDirection = GYRO_NORMAL;
                        Save_Config_to_EEPROM();
                        LED = 1;
                } else if (RxInPitch > STICK_THROW) { // reverse(down)
                        Config.PitchGyroDirection = GYRO_REVERSED;
                        Save_Config_to_EEPROM();
                        LED = 1;
                } else if (RxInYaw < -STICK_THROW) { // normal(left)
                        Config.YawGyroDirection = GYRO_NORMAL;
                        Save_Config_to_EEPROM();
                        LED = 1;
```

```
                        } else if (RxInYaw > STICK_THROW) { // reverse(right)
                                Config.YawGyroDirection = GYRO_REVERSED;
                                Save_Config_to_EEPROM();
                                LED = 1;
                        }

                        _delay_ms(50);
                        LED = 0;

                }
        }

                if (GainInADC[YAW] < (ADC_MAX * 5) / 100) {
                // flash LED 3 times
                for (i = 0;i < 3;i++) {
                        LED = 1;
                        _delay_ms(25);
                        LED = 0;
                        _delay_ms(25);
                }

                Armed = true;
                while (1) {
                        RxGetChannels();

#elif defined(QUAD_COPTER) || defined(QUAD_X_COPTER) || defined(Y4_COPTER)
                        MotorOut1 = RxInCollective;
                        MotorOut2 = RxInCollective;
                        MotorOut3 = RxInCollective;
                        MotorOut4 = RxInCollective;

#else
#endif
                        output_motor_ppm();
                }
        }
}

static inline void loop()
{
//      static uint8_t i;
        static uint16_t Change_Arming = 0;
        static uint8_t Arming_TCNT2 = 0;
        int16_t error, emax = 1023;
        int16_t imax, derivative;

        RxGetChannels();

        if (RxInCollective <= 0) {
                // Check for stick arming (Timer2 at 8MHz/1024 = 7812.5KHz)
                Change_Arming+= (uint8_t)(TCNT2 - Arming_TCNT2);
                Arming_TCNT2 = TCNT2;

                if (Armed) {
                        if (RxInYaw < STICK_THROW || abs(RxInPitch) > STICK_THROW)
                                Change_Arming = 0;            // re-set count
                } else {
                        if (RxInYaw > -STICK_THROW || abs(RxInPitch) > STICK_THROW)
                                Change_Arming = 0;            // re-set count
                }

                // 3Sec / 0.000128 = 23437 = 0x5B8D or
                // 2.5Sec / 0.000128 = 19531 = 0x4C4B
                // 0.5Sec / 0.000128 = 3906 = 0x0F42
                if (Change_Arming > 0x0F42) {
                        Armed = !Armed;
```

```
                        if (Armed)
                                CalibrateGyros();
                        ModeDelayCounter = 0;
                }
        }

        ReadGyros();

        LED = Armed;

        gyroADC[ROLL]-= gyroZero[ROLL];
        gyroADC[PITCH]-= gyroZero[PITCH];
        gyroADC[YAW]-= gyroZero[YAW];

        //--- Start mixing by setting collective to motor outputs

        RxInCollective = (RxInCollective * 10) >> 3; // 0-800 -> 0-1000

#ifndef SINGLE_COPTER
        if (RxInCollective > MAX_COLLECTIVE)
                RxInCollective = MAX_COLLECTIVE;
#endif


#elif defined(QUAD_COPTER) || defined(QUAD_X_COPTER)
        MotorOut1 = RxInCollective;
        MotorOut2 = RxInCollective;
        MotorOut3 = RxInCollective;
        MotorOut4 = RxInCollective;

#endif

        imax = RxInCollective;
        if (imax < 0)
                imax = 0;
        imax>>= 3;      /* 1000 -> 200 */


        RxInRoll    =    ((int32_t)RxInRoll    *    (uint32_t)GainInADC[ROLL])    >>
STICK_GAIN_SHIFT;
        gyroADC[ROLL]   =   ((int32_t)gyroADC[ROLL]   *   (uint32_t)GainInADC[ROLL])   >>
GYRO_GAIN_SHIFT;
        if (Config.RollGyroDirection == GYRO_NORMAL)
                gyroADC[ROLL] = -gyroADC[ROLL];

        if (Armed) {
                if (0) {
                        error = RxInRoll - gyroADC[ROLL];
                        if (error > emax)
                                error = emax;
                        else if (error < -emax)
                                error = -emax;
                        integral[ROLL]+= error;
                        if (integral[ROLL] > imax)
                                integral[ROLL] = imax;
                        else if (integral[ROLL] < -imax)
                                integral[ROLL] = -imax;
                        derivative = error - last_error[ROLL];
                        last_error[ROLL] = error;
                        RxInRoll+= error + (integral[ROLL] >> 2) + (derivative >> 2);
                } else {
                        RxInRoll-= gyroADC[ROLL];
                }
        }
```

```
#elif defined(QUAD_COPTER)
	MotorOut2+= RxInRoll;
	MotorOut3-= RxInRoll;
#elif defined(QUAD_X_COPTER)
	RxInRoll = RxInRoll >> 1;
	MotorOut1+= RxInRoll;
	MotorOut2-= RxInRoll;
	MotorOut3-= RxInRoll;
	MotorOut4+= RxInRoll;

#endif


	RxInPitch    =    ((int32_t)RxInPitch    *    (uint32_t)GainInADC[PITCH])    >>
STICK_GAIN_SHIFT;
	gyroADC[PITCH]  =  ((int32_t)gyroADC[PITCH]  *  (uint32_t)GainInADC[PITCH])  >>
GYRO_GAIN_SHIFT;
	if (Config.PitchGyroDirection == GYRO_NORMAL)
		gyroADC[PITCH] = -gyroADC[PITCH];

	if (Armed) {
		if (0) {
			error = RxInPitch - gyroADC[PITCH];
			if (error > emax)
				error = emax;
			else if (error < -emax)
				error = -emax;
			integral[PITCH]+= error;
			if (integral[PITCH] > imax)
				integral[PITCH] = imax;
			else if (integral[PITCH] < -imax)
				integral[PITCH] = -imax;
			derivative = error - last_error[PITCH];
			last_error[PITCH] = error;
			RxInPitch+= error + (integral[PITCH] >> 2) + (derivative >> 2);
		} else {
			RxInPitch-= gyroADC[PITCH];
		}
	}


#if defined(QUAD_COPTER)
	MotorOut1+= RxInPitch;
	MotorOut4-= RxInPitch;
#elif defined(QUAD_X_COPTER)
	RxInPitch = (RxInPitch >> 1);
	MotorOut1+= RxInPitch;
	MotorOut2+= RxInPitch;
	MotorOut3-= RxInPitch;
	MotorOut4-= RxInPitch;

#endif

	/* Calculate yaw output - Test without props!! */

	RxInYaw = ((int32_t)RxInYaw * (uint32_t)GainInADC[YAW]) >> STICK_GAIN_SHIFT;
	gyroADC[YAW]    =    ((int32_t)gyroADC[YAW]    *    (uint32_t)GainInADC[YAW])    >>
GYRO_GAIN_SHIFT;
	if (Config.YawGyroDirection == GYRO_NORMAL)
		gyroADC[YAW] = -gyroADC[YAW];

	if (Armed) {
		error = RxInYaw - gyroADC[YAW];
		if (error > emax)
			error = emax;
```

```
                else if (error < -emax)
                        error = -emax;
                integral[YAW]+= error;
                if (integral[YAW] > imax)
                        integral[YAW] = imax;
                else if (integral[YAW] < -imax)
                        integral[YAW] = -imax;
                derivative = error - last_error[YAW];
                last_error[YAW] = error;
                RxInYaw+= error + (integral[YAW] >> 4) + (derivative >> 4);
        }


#if defined(QUAD_COPTER)
        MotorOut1-= RxInYaw;
        MotorOut2+= RxInYaw;
        MotorOut3+= RxInYaw;
        MotorOut4-= RxInYaw;
#elif defined(QUAD_X_COPTER)
        MotorOut1-= RxInYaw;
        MotorOut2+= RxInYaw;
        MotorOut3-= RxInYaw;
        MotorOut4+= RxInYaw;

#endif


#endif
#if defined(QUAD_COPTER) || defined(QUAD_X_COPTER)   if (MotorOut4 < imax)
                MotorOut4 = imax;

#elif defined(QUAD_COPTER) || defined(QUAD_X_COPTER)        MotorOut1 = 0;
                MotorOut2 = 0;
                MotorOut3 = 0;
                MotorOut4 = 0;

#endif
        }

        LED = 0;
        output_motor_ppm();
}

int main()
{
        setup();

        while (1)
                loop();
        return 1;
}

static void init_adc()
{
        DIDR0   = 0b00111111;
        ADCSRB  = 0b00000000;
}

static void read_adc(uint8_t channel)
{
        ADMUX   = channel;                                              ADCSRA =
_BV(ADEN) | _BV(ADSC) | _BV(ADPS1) | _BV(ADPS2);     // 0b11000110

        while (ADCSRA & _BV(ADSC))
                ;       }
```

```
static void ReadGainPots()
{
	read_adc(3);                    // read roll gain ADC3
	GainInADC[ROLL] = GAIN_POT_REVERSE ADCW;

	read_adc(4);                    // read pitch gain ADC4
	GainInADC[PITCH] = GAIN_POT_REVERSE ADCW;

	read_adc(5);                    // read yaw gain ADC5
	GainInADC[YAW] = GAIN_POT_REVERSE ADCW;
}

static void ReadGyros()
{
	read_adc(2);                    // read roll gyro ADC2
	gyroADC[ROLL] = ADCW;

	read_adc(1);                    // read pitch gyro ADC1
	gyroADC[PITCH] = ADCW;

#ifdef EXTERNAL_YAW_GYRO
	gyroADC[YAW] = 0;
#else
	read_adc(0);                    // read yaw gyro ADC0
	gyroADC[YAW] = ADCW;
#endif
}

static void CalibrateGyros()
{
	uint8_t i;

	ReadGainPots();
	gyroZero[ROLL] = 0;
	gyroZero[PITCH] = 0;
	gyroZero[YAW] = 0;

	for (i = 0;i < 16;i++) {
		ReadGyros();

		gyroZero[ROLL]+= gyroADC[ROLL];
		gyroZero[PITCH]+= gyroADC[PITCH];
		gyroZero[YAW]+= gyroADC[YAW];
	}

	gyroZero[ROLL] = (gyroZero[ROLL] + 8) >> 4;
	gyroZero[PITCH] = (gyroZero[PITCH] + 8) >> 4;
	gyroZero[YAW] = (gyroZero[YAW] + 8) >> 4;
}


static int16_t fastdiv8(int16_t x) {
	if (x < 0)
		x+= 7;
	return x >> 3;
}


static void RxGetChannels()
{
	uint8_t t = 0xff;
	do {
		asm volatile("mov %0, %1":"=r" (i_sreg),"=r" (t)::"memory");
		RxInRoll = fastdiv8(RxChannel1 - 1520 * 8);
```

```
                RxInPitch = fastdiv8(RxChannel2 - 1520 * 8);
                RxInCollective = fastdiv8(RxChannel3 - 1120 * 8);
                RxInYaw = fastdiv8(RxChannel4 - 1520 * 8);
        } while (i_sreg != t);
#ifdef TWIN_COPTER
        RxInOrgPitch = RxInPitch;
#endif
}

static void output_motor_ppm()
{
        int16_t t;


        t = 1000;
        if (MotorOut1 < 0)
                MotorOut1 = 0;
        else if (MotorOut1 > t)
                MotorOut1 = t;



        t = MotorStartTCNT1 + MotorOut1;
        asm(""::"r" (t)); /* Avoid reordering of add after cli */
        cli();
        OCR1B = t;
        sei();
        t = MotorStartTCNT1 + MotorOut2;
        asm(""::"r" (t)); /* Avoid reordering of add after cli */
        cli();
        OCR1A = t;
        sei();
        TCCR1A = _BV(COM1A1) | _BV(COM1B1);
        OCR0A = MotorStartTCNT1 + MotorOut5;
        OCR0B = MotorStartTCNT1 + MotorOut6;

        do {
                cli();
                t = TCNT1;
                sei();
                t-= MotorStartTCNT1;
                if (t >= MotorOut3)
                        M3 = 0;
                if (t >= MotorOut4)
                        M4 = 0;
                if (t + 0xff >= MotorOut5)
                        TCCR0A&= ~_BV(COM0A0); /* Clear pin on match */
                if (t + 0xff >= MotorOut6)
                        TCCR0A&= ~_BV(COM0B0); /* Clear pin on match */
                t-= ((2000 + PWM_LOW_PULSE_US) << 3) - 0xff;
        } while (t < 0);



        MotorStartTCNT1+= (2000 + PWM_LOW_PULSE_US) << 3;
#if 0
        cli();
        t = TCNT1;
        sei();
        t+= 0x3f;
        t-= MotorStartTCNT1;
        if (t >= 0) {
                /*
                 * We've already passed the on cycle, hmm.
                 * Push it into the future.
                 */
```

```
                cli();
                t = TCNT1;
                sei();
                MotorStartTCNT1 = t + 0xff;
        }
#endif
        t = MotorStartTCNT1;
        cli();
        OCR1B = t;
        sei();
        OCR0A = t;
        OCR0B = t;
        cli();
        OCR1A = t;
        sei();


#if defined(QUAD_COPTER) || defined(QUAD_X_COPTER)
        TCCR1A = _BV(COM1A1) | _BV(COM1A0) | _BV(COM1B1) | _BV(COM1B0);
//      TCCR1C = _BV(FOC1A) | _BV(FOC1B);
        TCCR0A = _BV(COM0A1) | _BV(COM0A0) | _BV(COM0B1) | _BV(COM0B0);
//      TCCR0B = _BV(CS00) | _BV(FOC0A) | _BV(FOC0B);
#endif

        do {
                cli();
                t = TCNT1;
                sei();
                t-= MotorStartTCNT1;
        } while (t < 0);


#elif defined(QUAD_COPTER) || defined(QUAD_X_COPTER)
        M3 = 1;
        M4 = 1;
#endif

}

static void eeprom_write_byte_changed(uint8_t *addr, uint8_t value)
{
        if (eeprom_read_byte(addr) != value)
                eeprom_write_byte(addr, value);
}

static void eeprom_write_block_changes(const uint8_t *src, void *dest, size_t size)
{
        size_t len;

        for (len = 0;len < size;len++) {
                eeprom_write_byte_changed(dest, *src);
                src++;
                dest++;
        }
}

static void Initial_EEPROM_Config_Load()
{
        // load up last settings from EEPROM
        if (eeprom_read_byte((uint8_t *)EEPROM_DATA_START_POS) != 42) {
                Config.setup = 42;
                Set_EEPROM_Default_Config();
                // write to eeProm
                Save_Config_to_EEPROM();
        } else {
                // read eeprom
```

```
                eeprom_read_block(&Config, (void *)EEPROM_DATA_START_POS, sizeof(struct
config));
        }
}

static void Set_EEPROM_Default_Config()
{
        Config.RollGyroDirection        = GYRO_REVERSED;
        Config.PitchGyroDirection       = GYRO_REVERSED;
        Config.YawGyroDirection                 = GYRO_NORMAL;
}

static void Save_Config_to_EEPROM()
{
        eeprom_write_block_changes(
                (const void *)&Config, (void *)EEPROM_DATA_START_POS,
                sizeof(struct config));
}
```