PENGENDALIAN KESEIMBANGAN PADA *QUADROCOPTER* SAAT *HOVERING* MENGGUNAKAN KONTROLER PROPORSIONAL INTEGRAL DEFERENSIAL (PID) BERBASIS MIKROKONTROLER ATMEGA 168-20AU

SKRIPSI

Diajukan untuk memenuhi persyaratan

memperoleh gelar Sarjana Teknik

Disusun oleh :

ARIF DWI JAYANTO

NIM. 0810633031 – 63

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN

UNIVERSITAS BRAWIJAYA

FAKULTAS TEKNIK

JURUSAN ELEKTRO

MALANG

2012

LEMBAR PERSETUJUAN

PENGENDALIAN KESEIMBANGAN PADA QUADROCOPTER SAAT HOVERING 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 :

ARIF DWI JAYANTO

NIM. 0810633031 - 63

Telah diperiksa dan disetujui oleh

Dosen Pembimbing

Pembimbing I	Pembimbing II
Ir. Purwanto., MT	**Ir. Retnowati., MT**
NIP. 19540424 198601 1 001	**NIP. 19511224 198203 2 001**

LEMBAR PENGESAHAN

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

Disusun Oleh :

ARIF DWI JAYANTO

NIM : 0810633031 – 63

Skripsi ini telah diuji dan dinyatakan lulus

pada tanggal 26 Desember 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

Rahmadwati, ST., MT
NIP. 19771102 200604 2 003

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 Pada *Quadrocopter* Saat *Hovering* 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 Rusmiati, Ayah Eko Suwignyo 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. Kakak penulis Wiwit Wijayanti,
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. Ibu Dr.Rini Nur Hasanah., S.T., M.Sc. selaku Dosen Pembimbing akademik atas segala bimbingan, nasehat, gagasan, ide, saran, motivasi dan masukan yang telah diberikan,

9. 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,
10. Ibu Retnowati.,ST.,MT selaku Dosen Pembimbing II atas segala bimbingan, nasehat, gagasan, ide, saran, motivasi dan masukan yang telah diberikan,
11. Staff Recording Jurusan Teknik Elektro,
12. Rekan-rekan pengerjaan skripsi di Laboratorium Sistem Kontrol, Rio, Mahendra, Wahyu, Seif, Shidqi, Irfan, Mas Aldo, Karisma,
13. Rekan-rekan pengerjaan skripsi di rumah kontrakan, Safril, Andi, Fariz, Mifta, Umar, Aldy, Okta, Anas, Arnas, Liky, Jefri, Dimas aditya, Dodit,
14. Seluruh Keluarga Besar Anggota Tim Robot Seni, Gladi, Aka, Andi, Abu, Tansu, Muammar terima kasih atas segala semangat dan bantuan yang telah diberikan,
15. Seluruh Keluarga Besar Asisten Laboratorium Sistem Kontrol Jurusan Teknik Elektro atas segala masukan-masukannya,
16. Seluruh Keluarga Besar Tim Robot UB Jurusan Teknik Elektro atas segala bantuan alat, bahan dan masukan-masukannya yang telah di berikan,
17. 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 11 Desember 2012

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

Abstrak–-*Quadrocopter* merupakan miniatur pesawat terbang yang memiliki empat buah *propeller*. Miniatur pesawat terbang jenis ini kebanyakan digunakan untuk pengambilan *visualisasi* suatu kegiatan dari udara. Dengan keempat baling-balingnya diharapkan miniatur *quadrocopter* ini dapat melakukan *hover* di udara.

Untuk mencapai *hover* di udara, keempat motor harus di jaga kecepatannya saat di udara dengan variasi gangguan berupa angin di udara. Sistem kontrol yang digunakan untuk menstabilkan adalah kontroler PID (proporsional, integral dan diferensial). keuntungan sistem kontrol PID yaitu dengan sistemnya yang sederhana sehingga lebih cepat dalam mengambil sebuah keputusan dan mudah dalam analisis *tuning* nilai Kp, Ki, dan Kd.

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

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pada dasarnya beberapa golongan sangat membutuhkan pencitraan dari udara, seperti halnya untuk mengabadikan kegiatan *outdoor*. Dalam pencitraan dari udara sangat memerlukan biaya yang sangat besar tanpa adanya miniatur alat yang dapat melakukan gerakan aerodinamis di udara. Alat yang dapat mengudara identik dengan pesawat terbang. Dengan itu penulis mengangkat judul ini dan membuat suatu alat yang disebut dengan *quadrocopter*.

Quadrocopter merupakan salah satu bentuk dari sebuah pesawat terbang yang dilengkapi dengan empat baling-baling (*propeller*) pada ke empat lengan-lengannya. Dari ke empat baling-baling tersebut memiliki daya dorong ke bawah yang membuat *quadrocopter* dapat melayang di udara. Karena *quadrocopter* ini merupakan sebuah miniatur, daya jelajah dan ketahanan keseimbangan terhadap *disturbance* tergolong rendah, tetapi ketahanan miniatur *quadrocopter* untuk menjaga keseimbangannya lebih baik dari miniatur pesawat *rotary wing* yang hanya menggunakan satu baling – baling saat melakukan *hover*.

Dalam menjaga keseimbangannya *quadrocopter* harus dilengkapi dengan sebuah sensor yang disebut dengan *gyro*, untuk mendeteksi tingkat keseimbangan sebuah *quadrocopter* yaitu, *roll axis*, *yaw axis* dan *pitch axis*. Dari ketiga tingkat keseimbangan tersebut sangat berpengaruh terhadap *gain* masing-masing aktuatornya.

Dengan perubahan pada tiap-tiap aktuatornya maka harus menggunakan sebuah kontroler, yang digunakan adalah sebuah kontroler *Proporsional Integral Deferensial* (PID). Kontroler ini cukup handal untuk mengatasi masalah dari *disturbance* yang berubah-ubah. *Roll, pitch* dan *yaw axis* merupakan beberapa hal paling penting apabila *disturbance* pada saat keadaan di udara berubah-ubah. Hal ini dikarenakan adanya perubahan gain yang terjadi pada beberapa aktuatornya,

sehingga apabila mengubah *gain* salah satu dari empat aktuator maka kondisi keseimbangan juga berpengaruh.

Kontroler PID mampu menghasilkan sinyal kontrol yang berubah-ubah, sesuai dengan perubahan kondisi dari *plant* (Ogata, K., 1991). Hal inilah yang melandasi pemikiran untuk memperbaiki kinerja *roll, pitch* dan *yaw axis quadrocopter* pada saat mengudara. Dengan menggunakan kontroler Proporsional Integral Deferensial (PID) diharapkan keseimbangan *quadrocopter* mampu stabil meskipun dengan *disturbance* yang berubah-ubah agar mendapatkan hasil dari pencitraan yang maksimal saat pengambilan dari udara.

1.2 Rumusan Masalah

1. Bagaimana merancang kontroler Proporsional Integral Deferensial (PID) untuk mempertahankan keseimbangan *roll, yaw dan pitch axis* pada *quadrocopter*?
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 respon sistem *quadrocopter* , kinerja *driver* dan elektronika tidak dibahas mendalam.
3. Gangguan berupa distribusi beban yang tidak merata dan angin yang berubah-ubah secara acak dalam batasan yang telah ditentukan.
4. Pencitraan geografis dan proses terbang tidak dibahas mendalam.

1.4 Tujuan

Menciptakan sebuah keseimbangan 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 (*Brushless Motor Direct Current*), sensor *gyro*, kontroler, kontroler Proporsional Integral Deferensial (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 Alat

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.

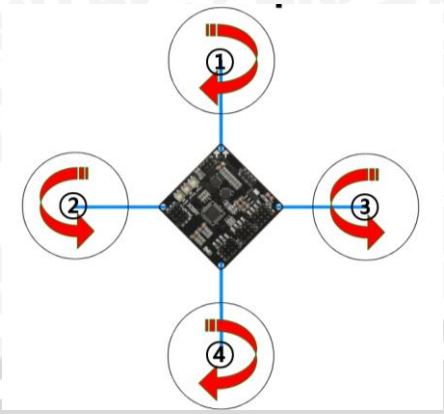
BAB II

TINJAUAN PUSTAKA

2.1 *Quadrocopter*

Quadrocopter atau biasa disebut juga dengan quadrotor atau helikopter quadrotor yang diangkat dan menggunakan empat buah *propeller*. *Quadrocopter* bisa diklasifikasikan sebagai helikopter, meskipun berbeda dengan helikopter pada umumnya. *Quadrocopter* 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, sehingga *quadrocopter* dapat setimbang atau posisi dari keempat baling-balingnya sejajar (rata-rata air) dan *quadrocopter* dapat mempertahankan posisinya atau yang biasa disebut dengan stabil sehingga *quadrocopter* dapat melakukan *hover*.

Untuk memdapatkan keseimbangan dan kesetabilan, setiap rotor harus 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 dari reaksi torsi dari masing-masing rotor pada *quadrocopter*.

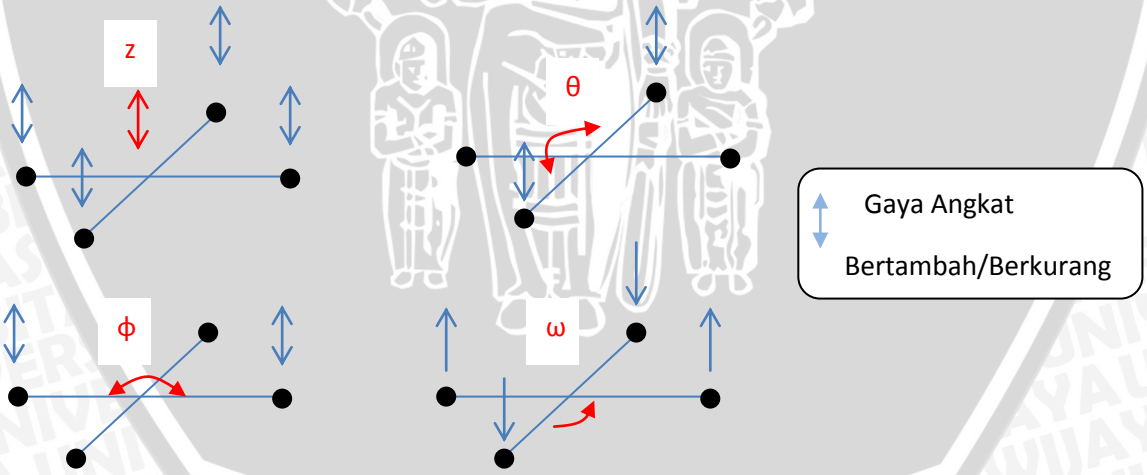


Gambar 2.1 Skema *Quadrocopter* dan *Body Frames*

Sumber : Luukonen, 2011:2

2.1.1 Skema Pergerakan *Quadrocopter*

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, 2008

Sesuai dengan skema pergerakan *quadrocopter* dalam Gambar 2.2, 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 simbol rotasi dari 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,

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

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 2-6 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 : P = posisi (pergerakan *translasi*)

Ω = rotasi (pergerakan *frame*)

θ = sumbu *pitch*

φ = sumbu *roll*

ω = sumbu *yaw*

2.2 Kontroler

Dengan adanya kontroler dalam sebuah sistem kontrol sangat berperan penting terhadap seluruh prilaku yang terjadi pada sistem. Pada dasarnya semua itu disebabkan oleh komponen yang digunakan sebagai perancangan system tersebut. Artinya, karakteristik plant yang digunakan harus dapat diterima sebagaimana adanya, sehingga segala pergerakan dari sistem hanya dapat dilakukan dengan menambahkan subsistem yaitu kontroler.

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

Jika perbedaan antara nilai referensi dengan nilai keluaran selisihnya relatif besar, maka kontroler yang baik harus dapat mempengaruhi plant agar memperkecil selisih nilai keluaran *plant* dengan nilai referensi sekecil mungkin secara cepat dan tepat.

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, K., 1997).

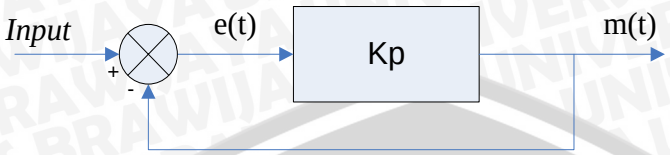
2.3 Kontroler PID (Proporsional Integral Diferensial)

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, K., 1997: 157

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

$$m(t) = Kp\ e(t) \quad \text{.....(2-1)}$$

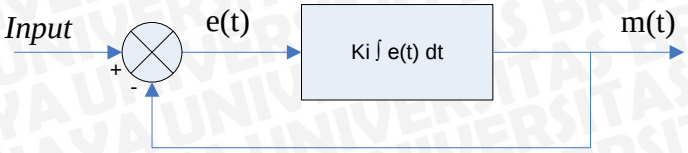
Sumber: Ogata, K,. 1997: 157

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, K., 1997: 158

Nilai keluaran kontroler *m(t)* sebanding dengan integral sinyal kesalahan e(t), Sehingga

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

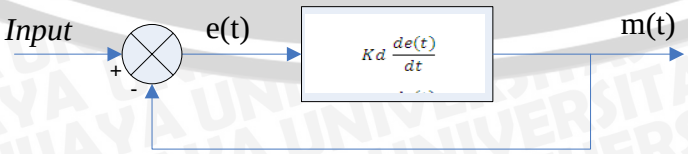
$$m(t) = Ki \int_0^t e(t)dt \quad \text{.....(2-3)}$$

Sumber: Ogata, K., 1997: 157

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 Diferensial

Kontroler diferensial 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 diferensial.



Gambar 2. 5 Diagram Blok Kontroler Diferensial

Sumber: Ogata, K., 1997: 177

Nilai keluaran kontroler m(t) sebanding laju sinyal kesalahan $\frac{de(t)}{dt}$.

Hubungan ini dapat ditulis sebagai:

$$m(t) = Kd\ \frac{de(t)}{dt} \quad \text{.....(2-4)}$$

Sumber: Ogata, 1997: 179

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

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

Dari ketiga aksi kontrol dasar di atas dapat dibuat kombinasi dari ketiganya, yaitu kontroler Proporsional Integral Deferensial (PID)

2.3.4 Kontroler Proporsional Integral Diferensial (PID)

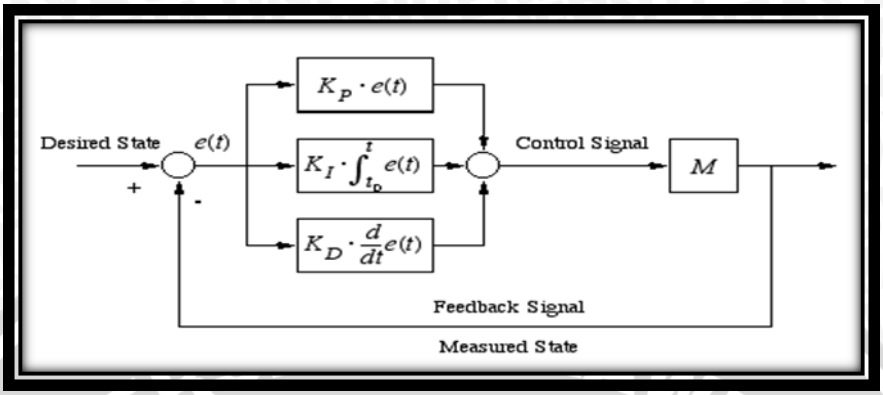
Setiap kekurangan dan kelebihan dari masing-masing kontroler Proporsional (P), Integral (I) dan Deferensial (D) dapat saling menutupi dengan menggabungkan ketiganya secara paralel menjadi kontroler proporsional integral diferensial (PID). Elemen-elemen kontroler Proporsional (P), Integral (I) dan Deferensial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

Aksi kontrolnya dinyatakan sebagai:

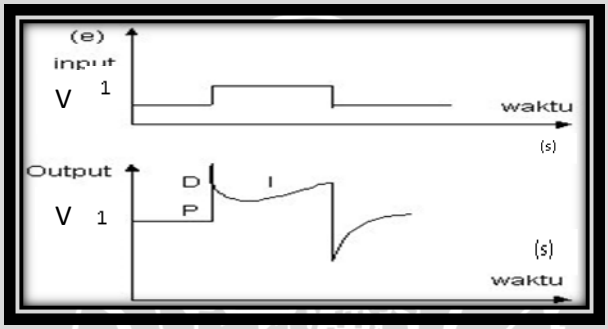
$$m(t) = K_p e(t) + \frac{Kp}{Ti} \int_0^t e(t)dt + K_p T_d \frac{de(t)}{dt} \quad \text{.....(2-5)}$$

Sumber: Ogata, K., 1997: 183

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



Gambar 2. 6 Diagram Blok Kontroler PID



Gambar 2. 7 Fungsi Waktu antara Sinyal Keluaran dan Sinyal Masukan Kontroler Proporsional Integral Deferensial (PID)

Sumber: Gunterus, 1994:8-11

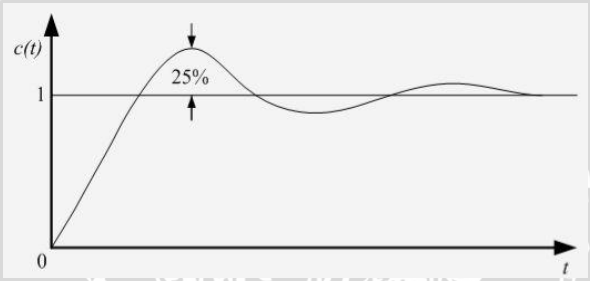
Keluaran kontroler Proporsional Integral Deferensial (PID) merupakan penjumlahan dari keluaran kontroler proporsional, integral dan diferensial. Gambar 2.6 menunjukkan hubungan tersebut. Karakteristik kontroler Proporsional Integral Deferensial (PID) sangat dipengaruhi oleh kontribusi besar dari ketiga parameter Proporsional (P), Integral (I) dan Deferensial (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 Metode Perancangan Kontroler Proporsional Integral Diferensial (PID) Menggunakan Metode Ziegler-Nichols.

Ziegler dan Nichols mengemukakan aturan-aturan untuk menentukan nilai dari gain proporsional Kp, waktu integral Ti, dan waktu derivatif Td berdasarkan karakteristik respon transien dari *plant* yang diberikan. Penentuan parameter kontroler PID atau penalaan kontroler PID tersebut dapat dilakukan dengan bereksperimen dengan plan.(Ogata, K., 1997)

Terdapat dua metode yang disebut dengan aturan penalaan Ziegler-Nichols, pada kedua metode tersebut memiliki tujuan yang sama yaitu untuk mencapai 25% *maximum overshoot* pada respon unit step, seperti ditunjukkan dalam Gambar 2.8

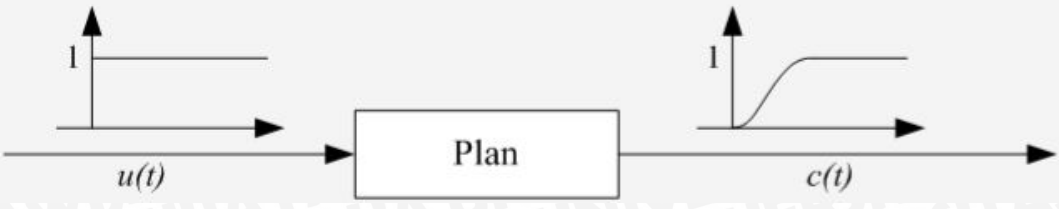


Gambar 2.8 Kurva Respon Unit Step yang Menunjukkan 25% *Maximum Overshoot*

Sumber: Ogata, K., 1997

a). Metode Pertama

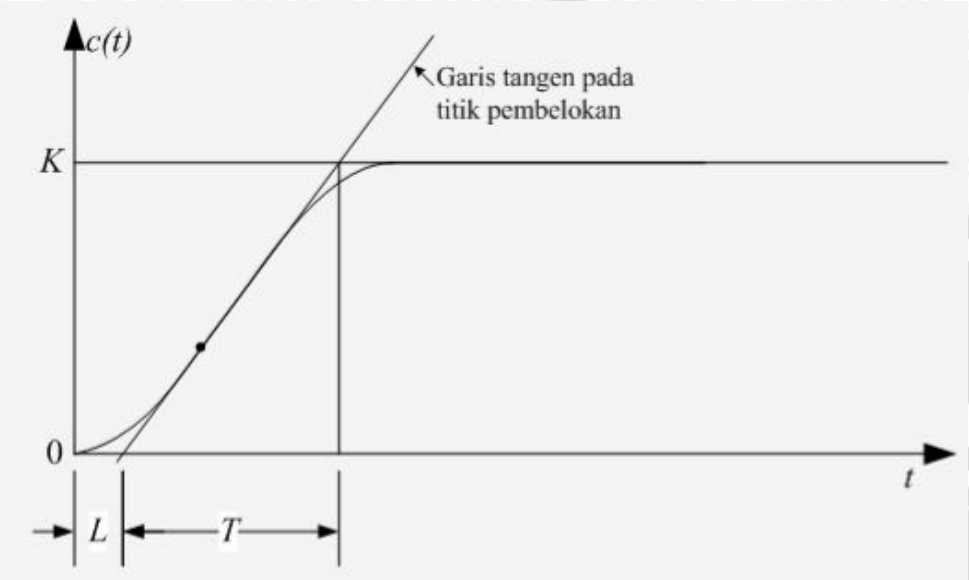
Metode pertama atau sering disebut metode kurva reaksi, respon dari plan dapat dapat diperoleh secara eksperimental dengan masukan berupa unit step, seperti yang ditunjukkan dalam Gambar 2.9



Gambar 2.9 Respon Plant Terhadap Masukan Berupa Unit Step

Sumber: Ogata, K. 1997

Jika dalam plan tersebut terdapat integrator atau *dominan complex-conjugate poles*, maka kurva respon unit step berbentuk seperti huruf S, seperti dalam Gambar 2.10 jika respon tidak memberikan bentuk kurva S, maka metode ini tidak berlaku.(Ogata, K., 1997).



Gambar 2.10 Kurva Respon yang Berbentuk S

Sumber: Ogata, K. 1997

Kurva berbentuk S tersebut dapat dikarakteristikkan menjadi dua konstanta yaitu waktu tunda *L* dan konstanta waktu *T*. Waktu tunda dan konstanta waktu ditentukan dengan menggambar sebuah garis tangen pada titik pembelokan dari kurva S, dan menentukan perpotongan antara garis tangen dengan sumbu waktu *t* dan sumbu *c(t) =K*, seperti yang telah ditunjukkan dalam Gambar 2.10 Fungsi alih *C(s)/U(s)* dapat dilakukan pendekatan dengan sistem orde satu dengan persamaan sebagai berikut:

$$\frac{C(s)}{U(s)} = \frac{Ke^{-Ls}}{Ts+1}$$

Sumber: Ogata, K. 1997

Ziegler dan Nichols menyarankan untuk menentukan nilai-nilai dari Kp, Ti dan Td berdasarkan pada formula yang ditunjukkan dalam Tabel 2.1. (Ogata, K., 1997).

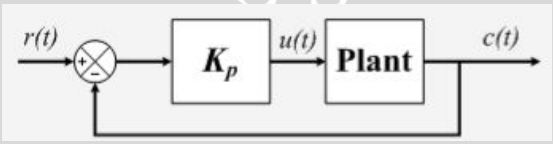
Tabel 2.1. Aturan Penalaan Ziegler-Nichols Berdasarkan Respon Unit Step Dari Plan

Tipe Kontroler	K_p	T_i	Td
P	$\frac{T}{L}$	∞	0
PI	$0{,}9\frac{T}{L}$	$\frac{L}{0{,}3}$	0
PID	$1{,}2\ \frac{T}{L}$	$2L$	$0{,}5\ L$

Sumber: Ogata, K. 1997

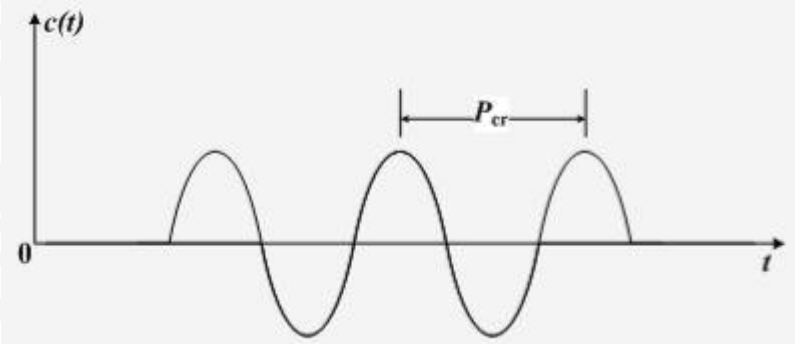
b). Metode Kedua

Dalam metode kedua ziegler-nichols, mula-mula yang dilakukan adalah membuat Ti = ∞ dan Td = 0. Kemudian hanya dengan menggunakan tindakan kontrol proporsional, harga ditingkatkan dari nol ke suatu nilai kritis *K*cr, disini mula-mula keluaran memiliki osilasi yang berkesinambungan (Jika keluaran tidak memiliki osilasi berkesinambungan untuk nilai Kp manapun yang telah diambil, maka metode ini tidak berlaku). Dari keluaran yang berosilasi secara berkesinambungan, penguatan kritis *K*cr dan periode *P*cr dapat ditentukan. Diagram blok sistem loop tertutup dengan kontroler proporsional dapat dilihat dalam Gambar 2.11. dan untuk osilasi berkesinambungan dengan periode Pcr dapat dilihat dalam gambar 2.12. Ziegler dan Nichols menyarankan penyetelan nilai parameter Kp, Ti , Td dan berdasarkan rumus yang diperlihatkan dalam Tabel 2.2 (Ogata, K., 1997).



Gambar 2.11 Sistem Loop Tertutup dengan Kontroler Proporsional

Sumber: Ogata, K., 1997



Gambar 2.12 Osilasi Berkesinambungan dengan Periode *P*cr

Sumber : Ogata, K., 1997

Tabel 2.2 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, K., 1997

2.3.6 *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 $K_d = 0$.

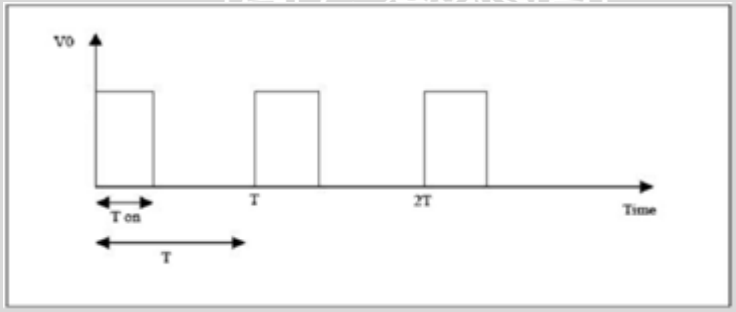
2. Mengatur nilai K_p hingga didapatkan respons yang diinginkan, dengan mengabaikan *offset* dari *setpoint*.
3. Dengan terus menaikkan nilai K_p, nilai dari K_d dinaikkan untuk mengurangi *overshoot* yang terjadi.
4. Naikkan nilai K_i untuk mengurangi *offset*.

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 2.13 berikut:



Gambar 2.13 Gambar Sinyal PWM Secara Umum
Sumber : electronics-scheme.com

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

Dengan :

Ton = Periode logika tinggi

T = Periode keseluruhan

$Vdc = Dutycycle\ x\ Vcc\ \ldots(V)$ ……(2-14)

Sedangkan frekuensi sinyal dapat ditentukan dengan rumus berikut :

$f0n = \frac{fclk\ I/0}{N.256}\ \ldots\ (Hz)$ ……(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).

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

dengan : I_{line} = Arus tiap jalur (A)

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)

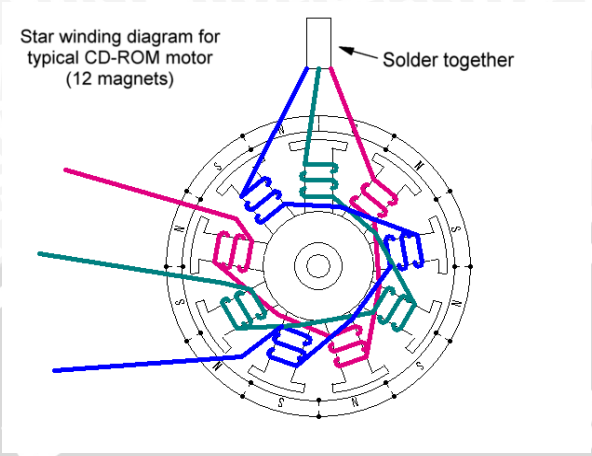
dengan : ω = kecepatan anguler (rad/s)

pi = 3.14

RPM = kecepatan putaran per menit

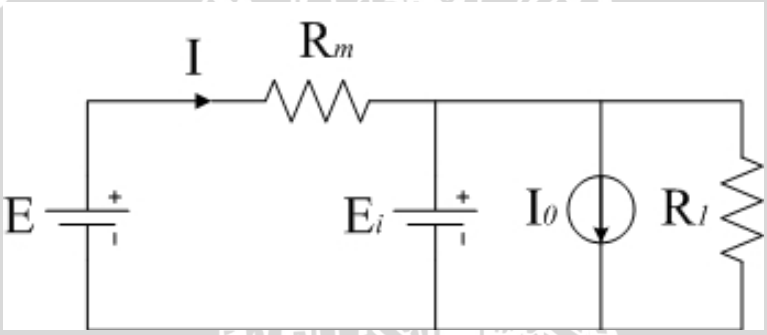
Brushless motor juga bisa dideskripsikan sebagai motor *stepper*. Dengan magnet permanen yang tetap dan mungkin juga dengan beberapa pole pada rotor daripada stator. Versi yang lain mungkin tanpa magnet permanen, hanya pole

yang diinduksi pada rotor kemudian ditarik pada lilitan stator. Dapat ditunjukkan motor DC brushless dalam Gambar 2.14 :



Gambar 2.14 Skematik *Brushless Motor*

Sumber : Buchi, 2012:12



Gambar 2.15 Rangkaian Ekivalen DC Motor
Sumber : Brown, 2002:6

Sesuai dengan rangkaian ekivalen motor BLDC di atas, dapat ditentukan hubungan (Brown, 2002:6):

$E_i = E - IR_m$ .....(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:

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

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 (Brown, 2002:6):

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

dengan: M = torsi (N.m)

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

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.

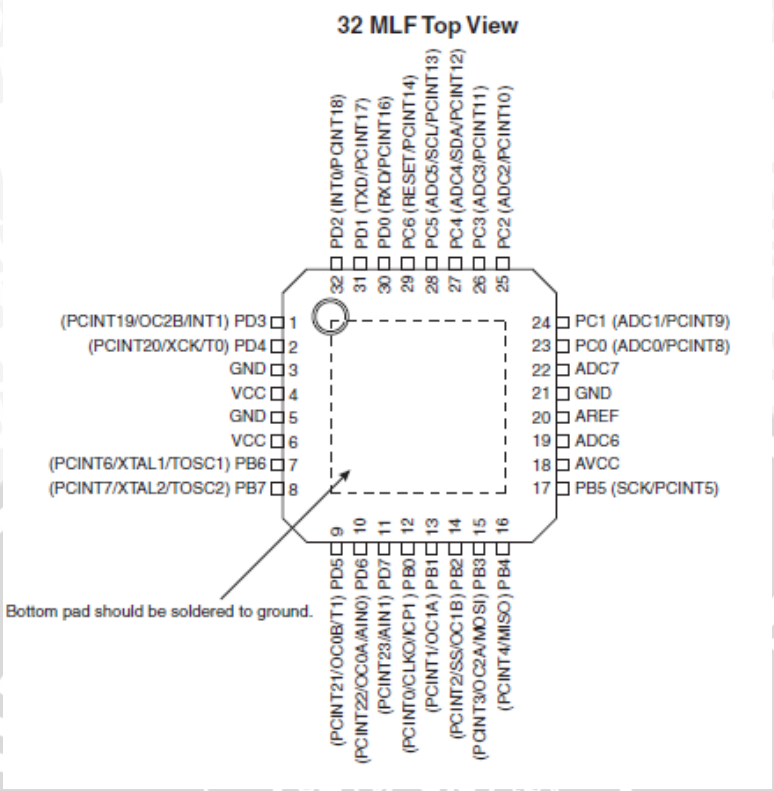
Sebuah *chip* mikrokontroler umumnya memiliki fitur:

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. Seperti dalam Gambar 2.16.



Gambar 2.16 Konfigurasi *Pin* ATMega 168-20AU

Sumber : http://atmel.com

2.6.2 Program CodeVision AVR

CodeVision AVR merupakan sebuah *cross-compiler* C, *Integrated Development Environtment* (IDE), dan *Automatic Program Generator* yang didesain untuk mikrokontroler buatan Atmel seri AVR. CodeVision AVR 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 pengiriman program kedalam *chip* mikrokontroler setelah sukses

melakukan kompilasi/asembli secara otomatis. *Software In-System Programmer* di *desain* 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, CodeVision AVR 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

CodeVision AVR juga mempunyai *Automatic Program Generator* bernama CodeWizard AVR, 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) m. Inisialisasi Bus I2C, Sensor Suhu LM75, *Thermometer/Thermostat* DS1621 dan *Real-Time Clock* PCF8563, PCF8583, DS1302, dan DS1307

n) n. Inisialisasi *Bus 1-Wire* dan Sensor Suhu DS1820, DS18S20

o) o. Inisialisasi modul LCD

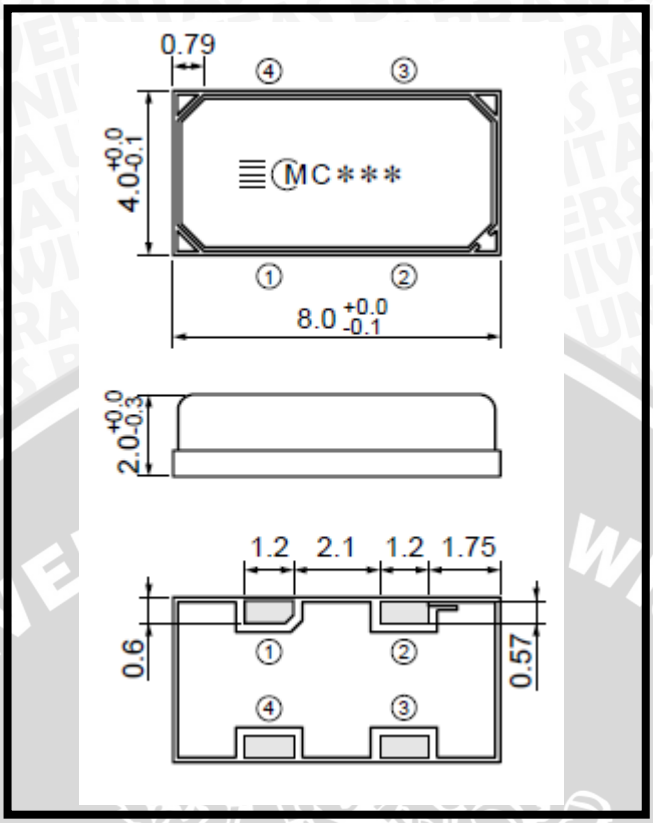
2.7 Sensor VSG (*Vibrating Structure Gyroscope*)

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*.

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 angular 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.

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

Untuk dimensi dan bentuk dari sensor VSG (*Vibrating Structure Gyroscope*) dapat dilihat pada Gambar 2.17:



Gambar 2.17 Sensor *Gyro* Murata ENC-03R

Sumber : http://murata.com

BAB III

METODOLOGI PENELITIAN

Kajian dalam skripsi ini merupakan penelitian yang bersifat aplikatif, yaitu merancangdan membuat suatu *Quadrocopter* dengan menitik beratkan keseimbangan pada sumbu (X, Y, Z) dengan pengendalian yang menggunakan kontroler Proporsional Integral Deferensial (PID) yang bertujuan agar dapat menampilkan performansi sistem sesuai dengan yang direncanakan.

Langkah-langkah yang perlu dilakukan untuk merealisasikan 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 *Quadrocopter* +.
2. *Quadrocopter* memiliki dimensi lengan 25 cm dari ujung lengan menuju *center plate*, jarak antar lengan ialah 90˚ dan menggunakan *acrylic* 5mm 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 ke tiga, 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*) 1200kV 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 Proporsional (P), Integral (I) dan Deferensial (D). Perancangan dimulai dari pembuatan *flowchart*, kemudian dilanjutkan dengan penulisan *listing code*.

3.3 Pengujian Alat

Setelah semua komponen pada alat sudah terhubung sesuai dengan diagram blok sistem yang telah dirancang dan perangkat lunak untuk mendukung sistem yang telah dibuat, maka diadakan pengujian dan analisa alat. Metode pengujiannya adalah sebagai berikut :

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 Kontrol PID

   Pengujian kontrol PID dilakukan dengan cara menggunakan suatu program yang dijalankan dengan memasukkan nilai *set value* (SV) dan *present value* (PV) selanjutnya data yang dihasilkan diproses sehingga *error* yang diperoleh sama dengan nol, atau nilai *set value = present value.*

3. 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 dalam skripsi ini bertujuan untuk merancang beberapa perangkat maupun alat secara keseluruhan. Perancangan perangkat tersebut meliputi perancangan perangkat keras maupun perancangan perangkat lunak. Sedangkan pembuatan bertujuan untuk menghasilkan semua perangkat pendukung maupun alat secara keseluruhan.

4.1 Spesifikasi Sistem

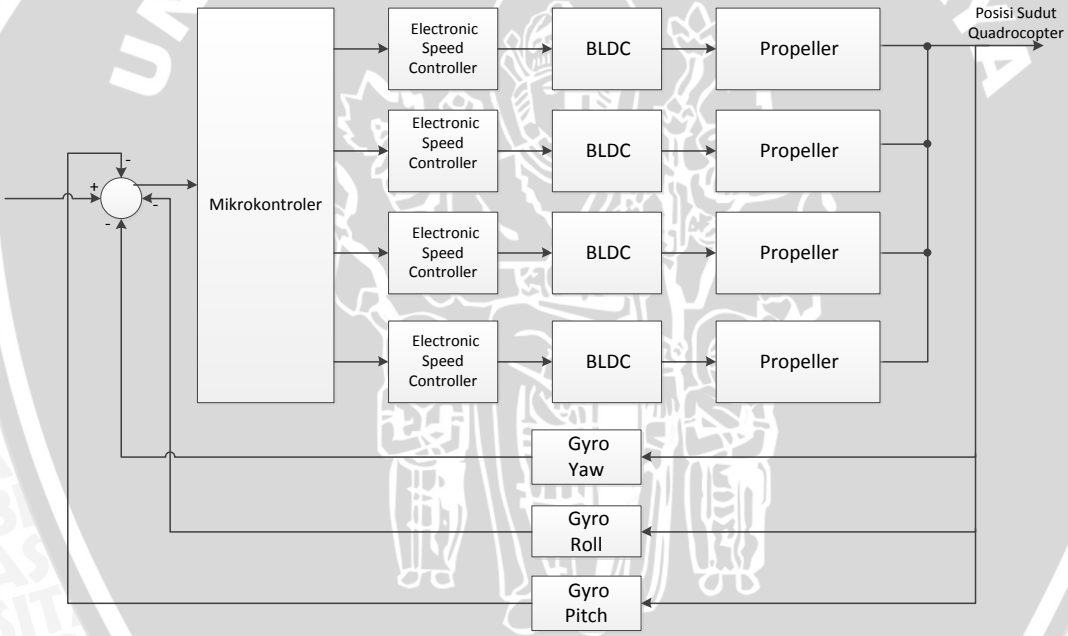
spesifikasi alat yang di rancang adalah sebagai beriku:

1. *Quadrocopter* yang dibuat menggunakan rangka dengan spesifikasi sebagai berikut (Vincent, 2010):
   - Panjang 4 (empat) buah lengan dengan masing-masing panjangnya 25 cm
   - Lengan *quadrocopter* menggunakan bahan aluminium berbetuk persegi dengan ketebalan 1 mm dan panjang masing-masing sisinya 1.2 cm.
   - Sudut antar lengan berjarak 90˚.
   - *Center plate* menggunakan bahan *acrylic* dengan tebal 5 mm sebanyak 2 (dua) buah untuk peletakan *board* elektrik utama dan mengunci masing-masing lengan bagian atas dan bawah
2. Pergerakan *quadrocopter* menggunakan 4 (empat) buah BLDC (*Brushless Motor Direct Current*) pada masing-masing lengannya.
3. kV *rating* pada BLDC (*Brushless Motor Direct Current*) adalah 1200kV dengan maksimum tegangan masukan adalah 11.1 V.
4. *Propeller* pada ujung *shaft* motor menggunakan EPP (*explore propeller plane)* 1045 (10x4.5 inchi).

5. 4 (empat) ESC (*Electronic Speed Controller)* sebagai pengkondisi sinyal masukan pada BLDC (*Brushless Motor Direct Current*) 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

Dalam skripsi ini sebelumnya sudah dibuat diagram blok agar dalam pengerjaan dapat dilakukan sesuai dengan rangcangan sistem dari *quadrocopter*. Adapun diagram blok tersebut dapat dilihat pada Gambar 4.1:



Gambar 4.1 Diagram Blok Sistem *Quadrocopter*

Keterangan dari diagram blok dalam Gambar.4.1:

- *Input* berupa sudut diberikan melalui *transmitter*, 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 (*electronic speed controller).*

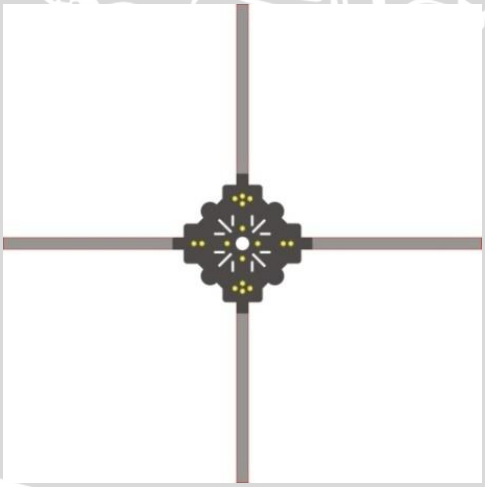
- ESC kemudian menguatkan sinyal kontrol dan diteruskan menuju BLDC (*Brushless Motor Direct Current*).
- Keluaran sudut (*present value*) kemudian di *feedback* pada tiga buah sensor *gyro*.
- Hasil pembacaan sensor kemudian dikurangkan dengan *input* sehingga mikrokontroler mampu mengkompensasi *error* yang terjadi.

4.3 Perancangan Perangkat Keras

Perangkat keras terdiri dari rangka *quadrocopter, board* mikrokontroler, pengkondisi sinyal sensor *gyro*, ESC (*electronic speed controller),* BLDC (*Brushless Motor Direct Current*) dan *propeller* (baling-baling).

4.3.1 Perancangan Rangka *Quadrocopter*

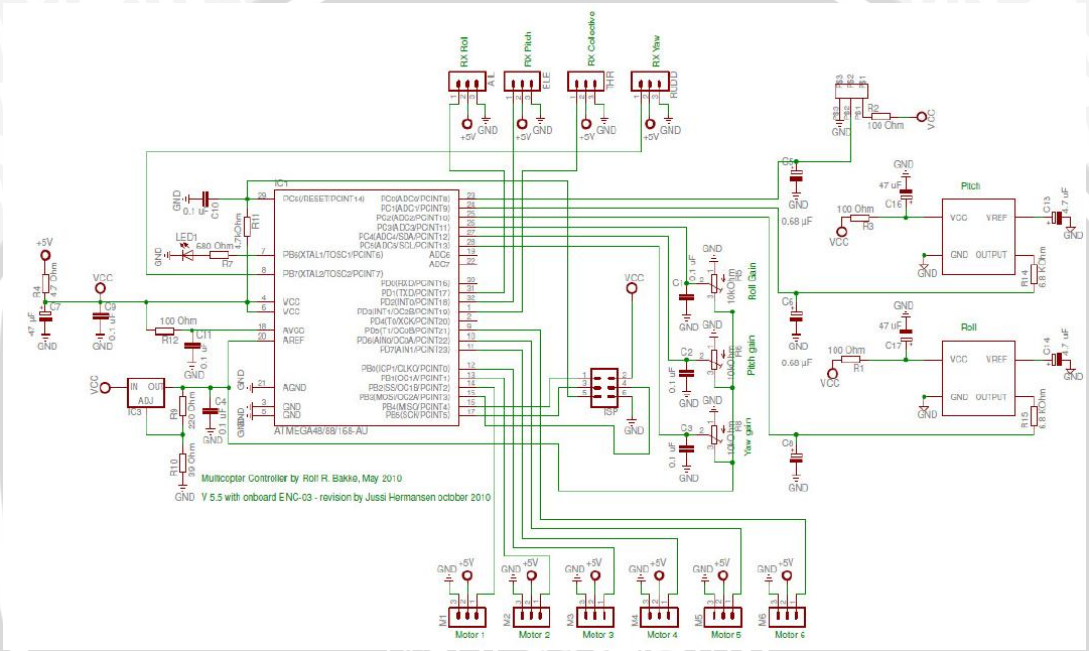
Perancangan *qudrocopter* ini terdiri dari perancangan *Center plate* dan lengan-lengan *quadrocopter*. Perancangan rangka *quadrocopter* dilakukan sebagai dasar dari komponen-komponen yang akan diletakkan. Secara umum rancangan rangka *quadrocopter* ditunjukkan dalam Gambar 4.2 :



Gambar 4.2 *Frame* (rangka) *Quadrocopter*

4.3.2 Rangkaian *Board* Mikrokontroler

Board mikrokontroler merupakan rangkaian utama dari system *quadrocopter*. Di dalam *board* ini mikrokontroler terhubung langsung dengan sensor *gyro* sehingga menjadi satu dengan *board* utama. Pada pengendalian *quadrocopter* digunakan ATMEGA 168-20AU. Rangkaian boar utama dapat dilihat dalam Gambar 4.3:



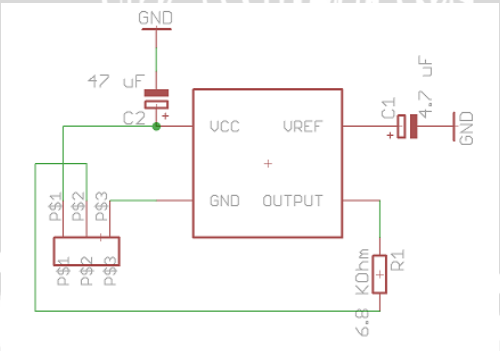
Gambar 4.3 Rangkaian *Board* Mikrokontroler

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

Rangkaian sensor *gyro* terdiri dari sensor *gyro*, *low-pass filter*, RC *tap*. *Low-pass filter* yang terdiri dari resistor 6.8kΩ dan kapasitor 0.68µF. Rangkaian tersebut berfungsi untuk mengeliminasi tegangan keluaran yang dikarenakan oleh getaran. RC *tap* berfungsi untuk menghilangkan *ripple* pada tegangan masukan *gyro*. Rangakain sensor *gyro* dapat dilihat dalam Gambar 4.4 :



Gambar 4.4 Rangkaian Sensor *Gyro*

4.3.3 *Electronic Speed Controller* (ESC)

Electronic speed controller (ESC) berfungsi sebagai penguat tegangan dari masukan *pin* motor dan juga berfungsi sebagai pemecah fasa. Karena BLDC (*Brushless Motor Direct Current*) yang digunakan merupakan motor DC dengan 3 fasa yang memiliki hubungan *star*.

4.3.4 Pemilihan BLDC dan *Propeller*

Dalam pemilihan BLDC dan *propeller* yang tepat harus mengkombinasikan keduanya agar mampu menghasilkan daya angkat (*thrust*) yang sesuai dengan berat keseluruhan *quadrocopter* (termasuk *battery*), ESC , BLDC dan komponen-komponen lainnya. Untuk mengetahui perhitungan *thrust* yang dihasilkan sebuah *propeller* dapat diperoleh melalui perhitungan dari persamaan 4.1:

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

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 dijelaskan sebelumnya, bahwa pada *quadrocopter* ini menggunakan EPP 1045 dengan BLDC 1200kV. EPP 1045 memiliki diameter 25.4 cm. Sementara BLDC 1200kV mampu menghasilkan putaran hingga 12000 rpm dengan tegangan masukan 10 V. Tabel 4.2 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
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

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)

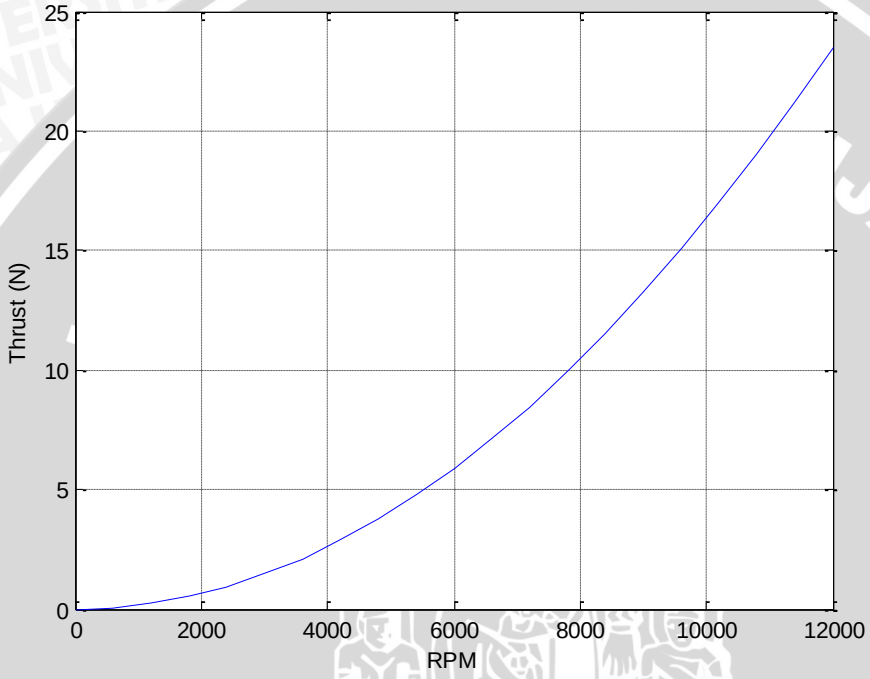
maka :

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

$= 17.568\ N$

Hal ini berarti *thrust* minimal yang harus dihasilkan oleh keempat *propeller* adalah 17.658 N. Sesuai dengan perhitungan pada Tabel 4.2, 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 *hoover*. Gambar 4.5 menunjukkan grafik hubungan linier antara RPM dan *thrust* pada EPP 1045 dan bentuk dari *propeller* yang digunakan dapat dilihat dalam Gambar 4.6.



Gambar 4.5 Hubungan *Thrust* dan RPM EPP1045

Gambar 4.6 *Propeller* EPP1045 CW dan CCW

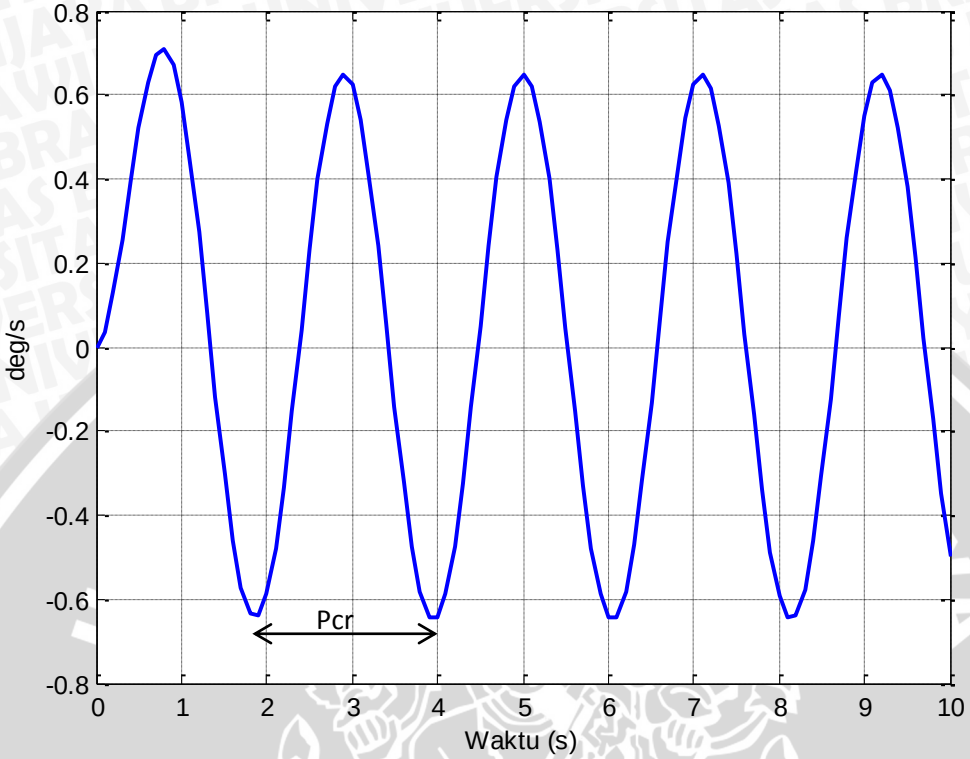
Pada *quadrocopter* ini juga menggunakan BLDC dengan kV *rating* 1200 dengan masukan tegangan maksimum 11.1 volt. Hal ini berarti BLDC mampu menghasilkan putaran hingga 13320 RPM atau setara dengan 1394.86 rad/s yang sesuai dengan Tabel 4.2. Hal ini cukup untuk memenuhi nilai RPM minimal untuk melakukan gaya angkat dari *quadrocopter* tersebut. Bentuk dan spesifikasi dari BLDC yang digunakan dapat dilihat pada Gambar 4.7:

Gambar 4.7 BLDC 1200kV

4.4 Penetuan Nilai Penguatan Kontroler

Untuk mendapatkan nilai penguatan kontroler agar *output* sesuai dengan keinginan akan dilakukan dengan cara metode *hand tuning* tetapi dengan penetuan awal nilai kontroler, terlebih dahulu akan menggunakan metode Ziegler-Nichols agar proses metode *hand tuning* tidak terlalu lama.

Dengan mengaplikasikan metode Ziegler-Nichols, pada sebuah sistem *loop* tertutup dengan penguatan K_{cr} sebesar 3, maka didapatkan respons yang di tunjukkan dalam Gambar 4.8:



Gambar 4.8 Osilasi Berkesinambungan dengan Periode Pcr

Berdasarkan Gambar 4.8, dapat diperoleh nilai *critical period* (P_{cr}) sebesar 2.1 s. Untuk dapat menetukan nilai parameter kontroler PID, langkah yang dilakukan sesuai dengan tabel aturan dasar Ziegler-Nichols metode kedua berdasarkan *critical gain* (K_{cr}) dan *critical period* (P_{cr}) yang ditunjukkan dalam Tabel 4.3:

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, K., 1997

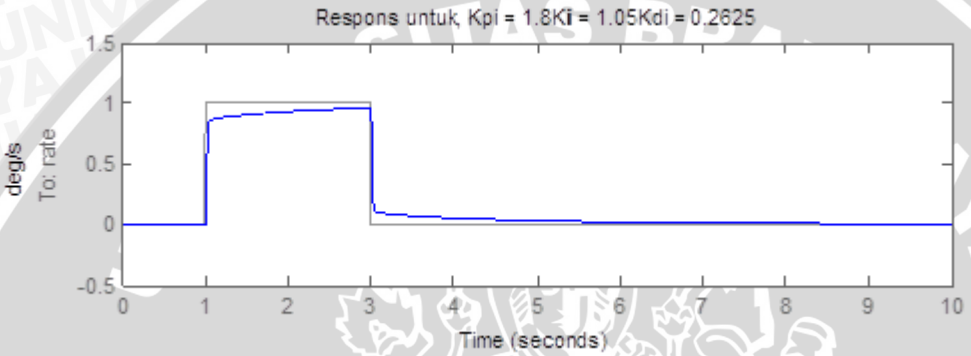
Maka didapatkan nilai parameter PID sebagai berikut :

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 seperti yang ditunjukkan dalam Gambar 4.9:

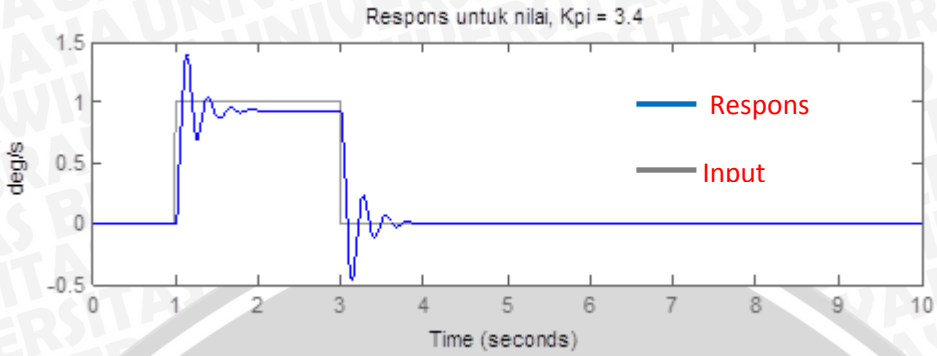


Gambar 4.9 Respon Sistem untuk Nilai PID Berdasarkan Ziegler-Nichols

Dari respons sistem dengan parameter PID menurut Ziegler-Nichols metode kedua dalam Gambar 4.9, dapat dilihat bahwa respon sistem masih memiliki *offset* atau *error steady state* yang dapat dilihat dalam Tabel 4.4 dalam bentuk prosentase, sehingga nilai parameter PID perlu diubah-ubah secara manual (*hand tuning*) sampai diperoleh respon seperti dalam Gambar 4.10.

Tabel 4.4 Penguatan Kp yang Berbeda

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

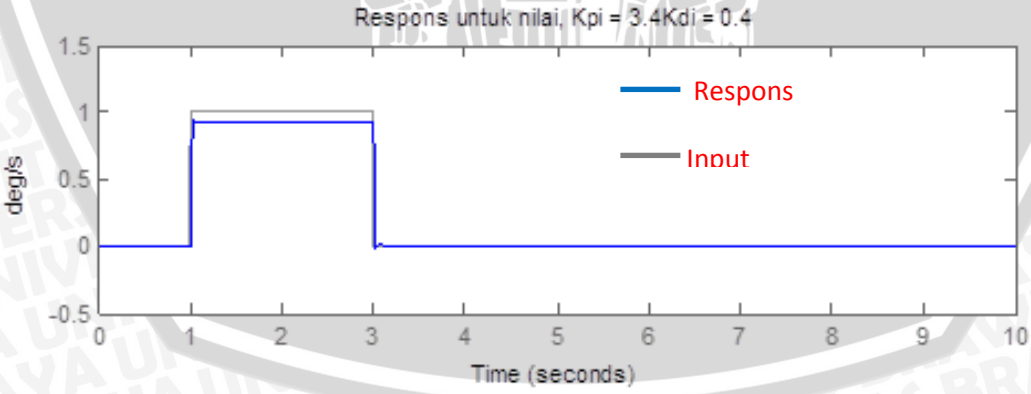


Gambar 4.10 Respons untuk nilai Kp = 3.4

Dari Gambar 4.10 respons untuk Kp = 3.4 masih terdapat *overshoot*. Untuk menghilangkan *overshoot*, nilai peguatan Kd perlu dinaikkan dengan tujuan memper kecil nilai *overshoot*. Dari perubahan nilai Kd seperti dalam Tabel 4.5 akan didapatkan prosentase nilai *overshoot* dan akan menghasilkan respon sistem seperti dalam Gambar 4.11.

Tabel 4.5 Penguatan Kd yang Berbeda

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

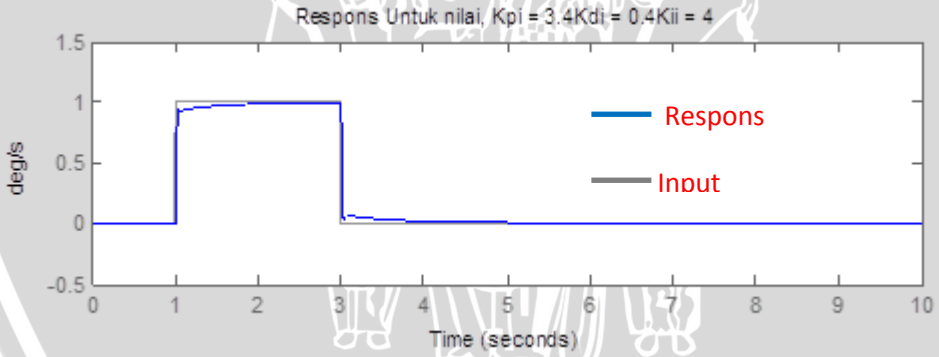


Gambar 4.11 Respons Untuk Kp = 3.4 dan Kd = 0.4

Dari Gambar 4.11 dapat dilihat bahwa respons sistem masih memiliki *offset*. Untuk menghilangkan *offset* yang terjadi, maka nilai Ki perlu di naikkan. Karena pada kontroler dengan aksi kontrol integral, nilai masukan kontroler u(t) diubah pada laju proporsional dari sinyal pembangkit kesalahan (e(t)). Dapat dilihat prosentase nilai *error steady state* dalam Tabel 4.7 yang menghasilkan respon sistem seperti dalam gambar 4.12.

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

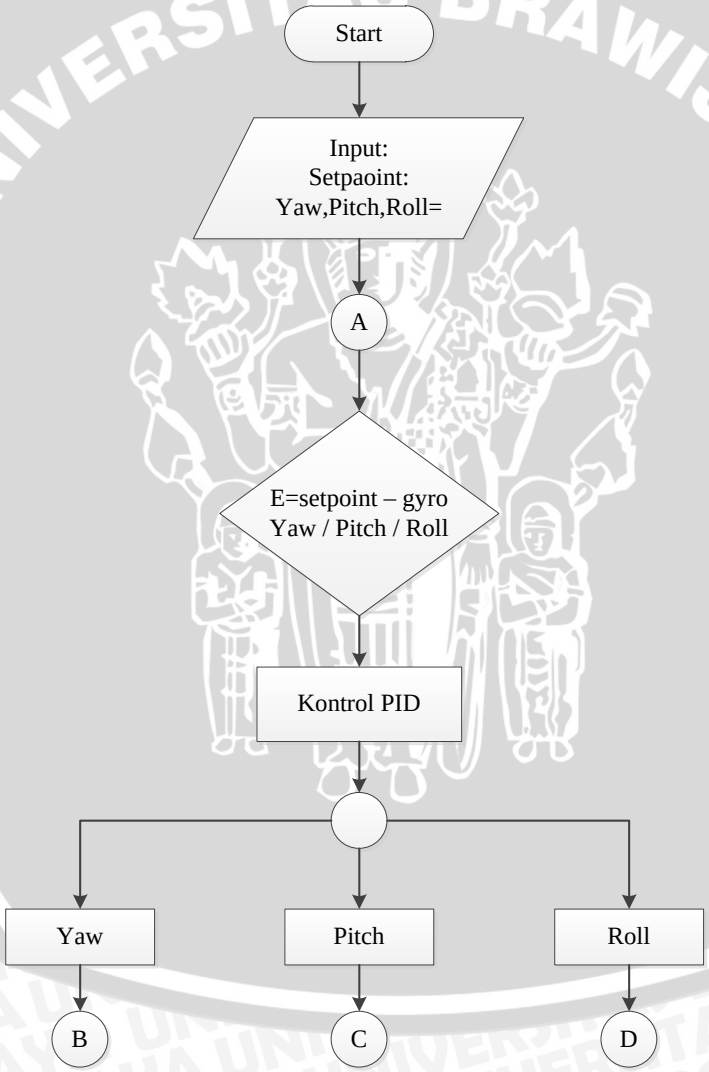


Gambar 4.12 Respon Untuk Kp = 3.4, Kd = 0.4 dan Ki = 4

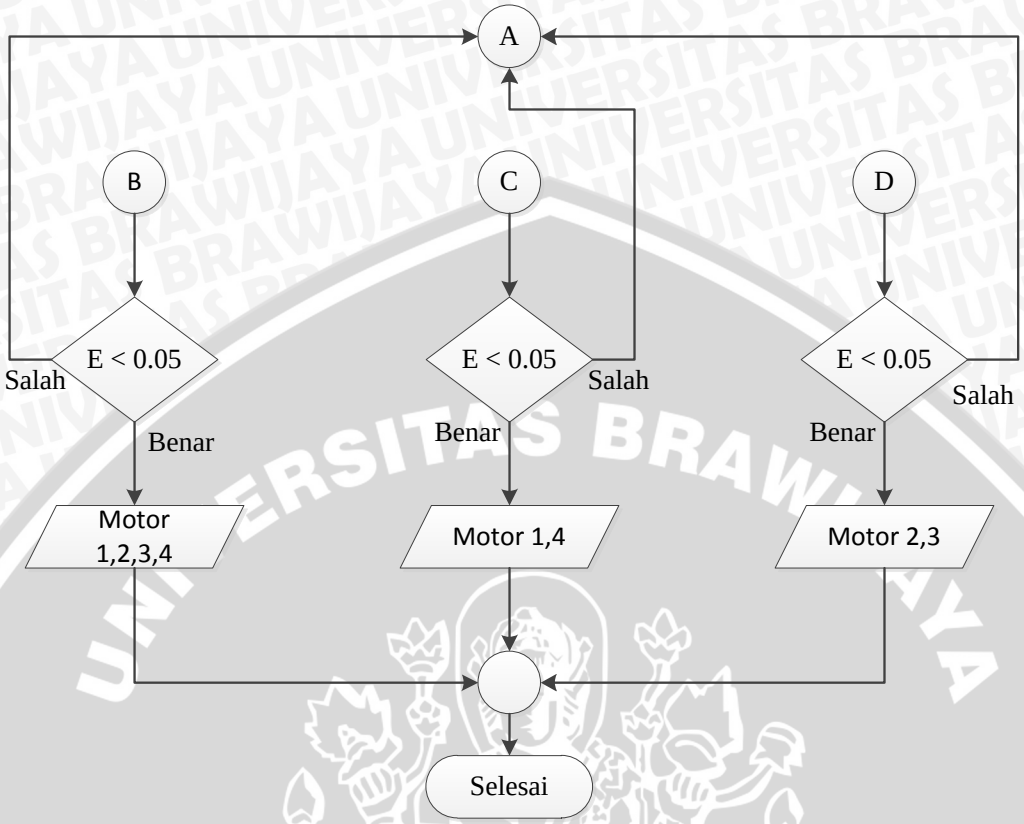
Dari penentuan nilai penguatan Kp, Ki dan Kd dari langkah-langkah yang sudah diterapkan dapat dipastikan nilai penguatan yang digunakan untuk sistem *quadrocopter* adalah Kp = 3.4, Ki = 4 dan Kd = 0.4.

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 Ziegler-Nichols yang telah dimasukkan pada mikrokontroler yang digunakan. *Flowchart* perancangan perangkat lunak dapat dilihat pada Gambar 4.13 dan Gambar 4.14:



Gambar 4.13 *Flowchart* Perangkat Lunak



Gambar 4.14 *Flowchart* Perangkat Lunak

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 yang dilakukan. Adapun langkah – langkah pengujian yang dilakukan adalah:

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

5.1 Pengujian Sensor *Gyro*

Pengujian ini bertujuan untuk mengetahui tingkat kerberhasilan dari sensor *gyro* yang digunakan untuk mendeteksi dari setiap perubahan sudut terhadap sumbu pada *quadrocopter*. Dengan setiap pergerakan dan gangguan pada *quadrocopter* diharapkan dapat setimbang secara stabil di udara untuk mencapai posisi *hover*, yang ditunjukkan dengan posisi keempat *propeller* sama tingginya. Maka harus dilakukan pengujian untuk sensor *gyro* sebelum digunakan. Untuk pengujian suatu respon sensor *gyro* digunakan fasilitas *Hyperterminal* pada Windows XP untuk melihat data keluaran dari sensor *gyro* yang digunakan.

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 yang digunakan.

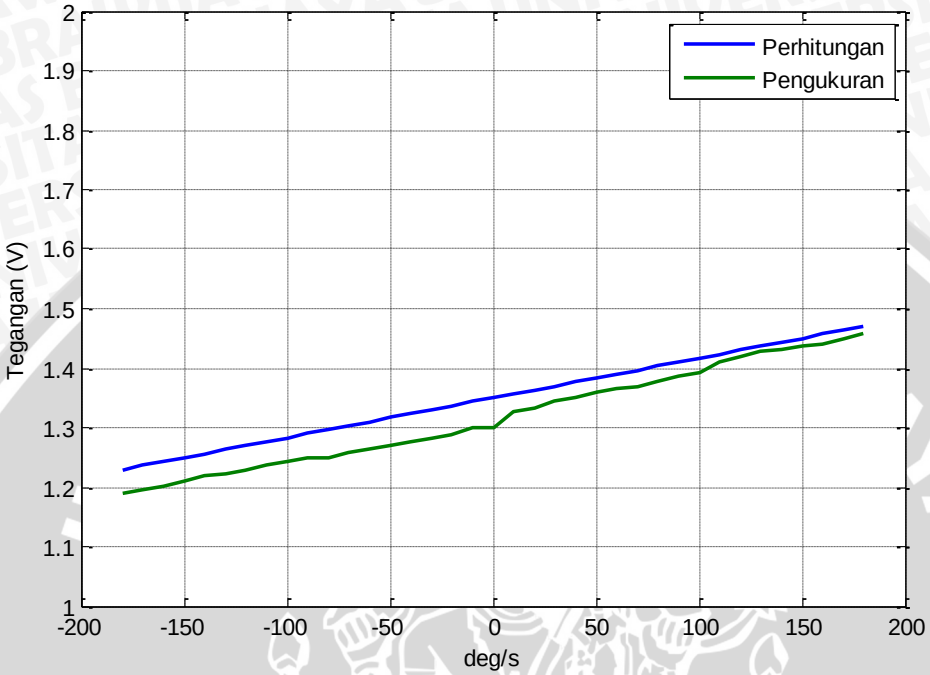
5.1.3 Hasil Pengujian

Dengan melakukan prosedur pengujian berdsasarkan tegangan, maka akan didapatkan keluaran dari *gyro* sesuai dengan persamaan (2-13), dan hasilnya dapat dilihat pada Tabel 5.1:

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

Dari hasil Tabel 5.1 akan didapatkan karakteristik dari respon gyro yang dapat dilihat dalam Gambar 5.1:



Gambar 5.1 Respon Sensor *Gyro* CW dan CCW

Dari hasil pengujian berdasarkan Tabel 5.1 dan Gambar 5.1 dapat dilihat bahwa kerja dari sensor *gyro* yang digunakan memiliki karakteristik semakin kecil kecepatan sudutnya maka semakin kecil nilai dari tegangan yang terukur. Dan nilai *error* pada antara nilai tegangan berdasarkan perhitungan dan hasil pengukuran didapatkan antara 0.006 % sampai 0.036%.

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

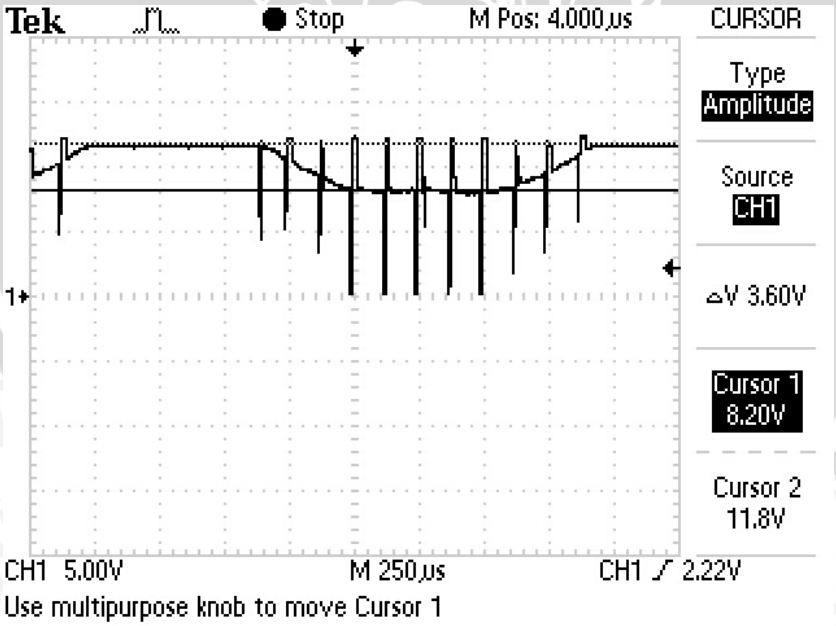
1. *Borad* Mikrokontroler
2. Osiloskop
3. ESC
4. BLDC Motor
5. *Tachometer*

5.2.2 Prosedur Pengujian

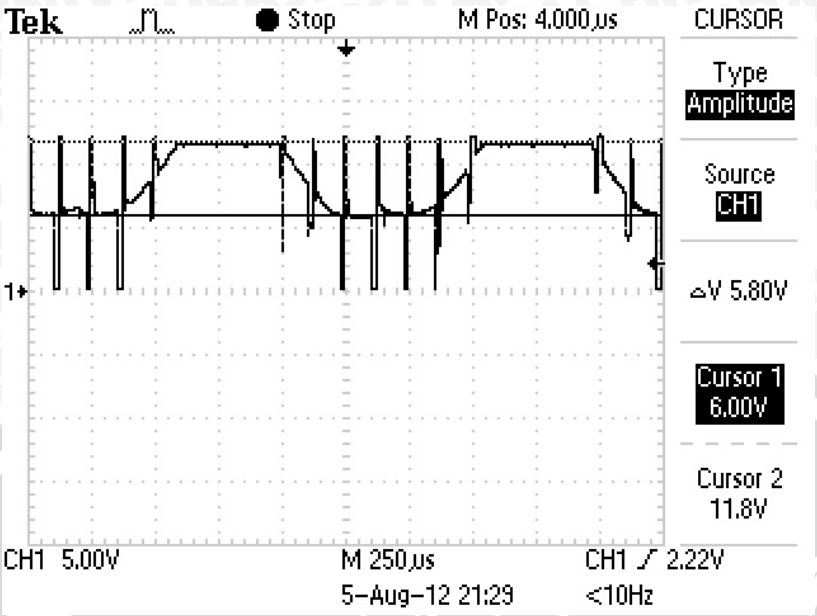
Pengujian dilakukan dengan memberikan tegangan melalui ESC dengan nilai PWM divariasi mulai dari 10% sampai dengan 100%. Salah satu *line* masukan dari BLDC dihubungkan dengan osiloskop dengan bagian luar motor diukur kecepatannya dengan menggunakan *tachometer*.

5.2.3 Hasil Pengujian

Dari prosedur pengujian yang dilakukan maka didapatkan hasil sinyal masukan yang pada motor BLDC yang digunakan seperti dalam Gambar 5.2 dengan tegangan 3.60 V dan Gambar 5.3 dengan tegangan 5.80 V :



Gambar 5.2 Sinyal Masukan dengan Tegangan 3.60V pada Motor BLDC



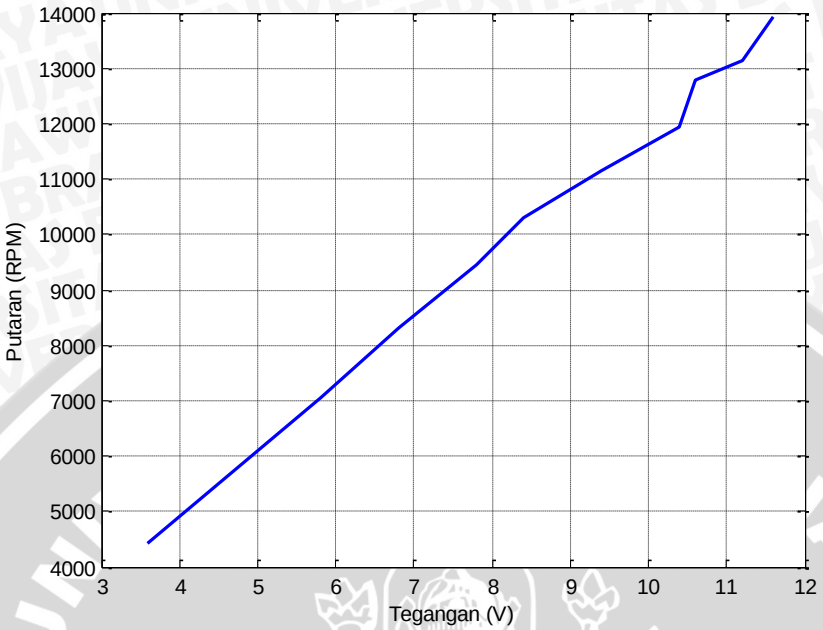
Gambar 5.3 Sinyal Masukan dengan Tegangan 5.80V pada Motor BLDC

Dari pengujian PWM (tegangan masukan dari ESC) dan kecepatan pengujian motor akan didapatkan hubungan dari keduanya yang dapat dilihat dalam Tabel 5.3:

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

Berdasarkan Tabel 5.3 akan didapatkan hasil karakteristik motor BLDC yang akan menghasilkan kecepatan putaran motor sesuai dengan tegangan masukan yang di berikan oleh ESC dapat dilihat dalam Gambar 5.4:



Gambar 5.4 Grafik Hubungan Tegangan dan RPM BLDC 1200K_v

Dari hasil pengujian motor BLDC berdasarkan Tabel 5.2 dan Gambar 5.4 didapatkan karakteristik dari motor BLDC semakin besar tegangan yang digunakan maka akan semakin besar pula kecepatan putaran yang di hasilkan oleh BLDC.

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. Laptop
3. BLDC
4. ELAB

5. Battery Li-Po 11.1 V
6. *Transmitter*
7. Osiloskop

5.3.2 Prosedur Pengujian

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

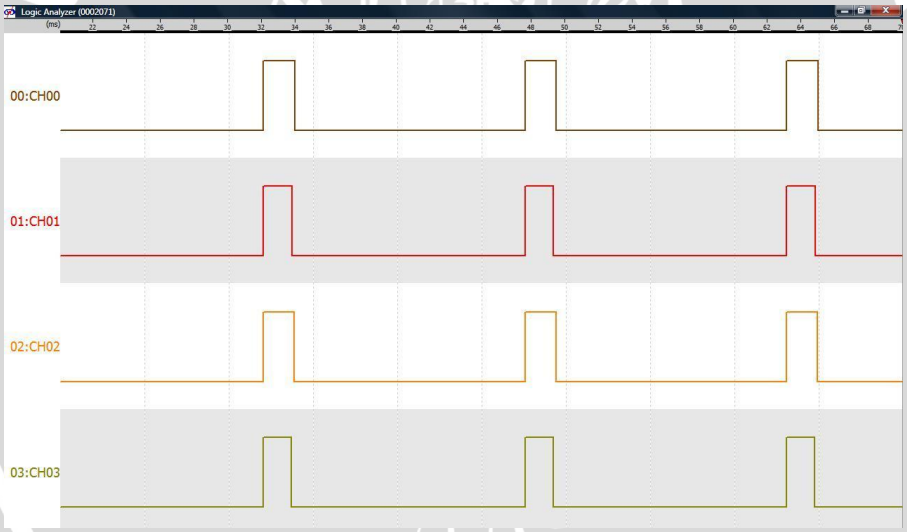
Untuk pengujian dengan menggunakan ELAB, empat *pin* keluaran motor dihubungkan sekaligus pada ELAB, dengan konfigurasi *channel* sebagai berikut:

- *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 dalam Gambar 5.5 dan Gambar 5.6 sesuai dengan lebar sinyal yang telah di berikan.

Gambar 5.5 Sinyal PWM Waktu Kondisi Diam

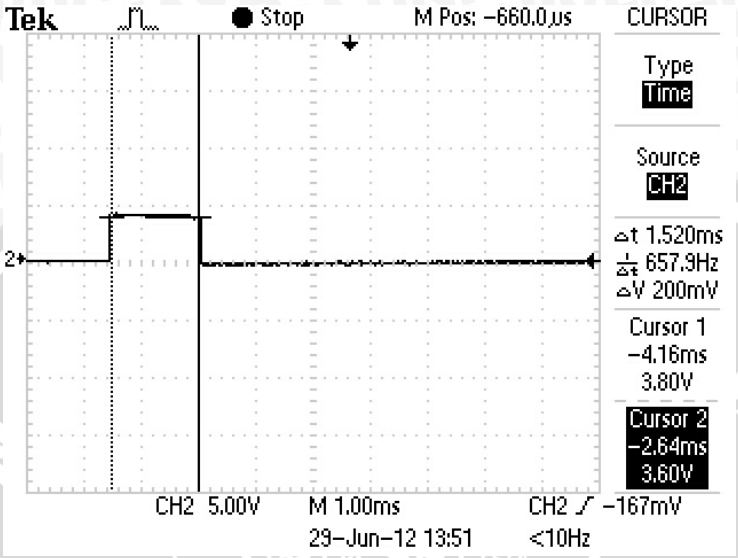


Gambar 5.6 Sinyal PWM Waktu Kondisi Berputar

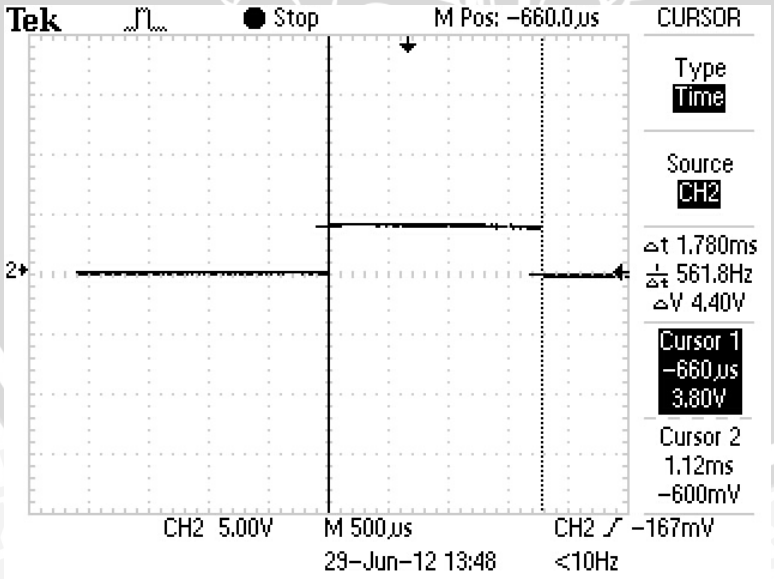
Pada kedua Gambar 5.5 dan Gambar 5.6 di atas, dapat dilihat bahwa mikrokontroler menghasilkan sinyal PWM secara bersamaan. Artinya bahwa keempat motor BLDC bekerja secara bersamaan dan dengan lebar sinyal yang sama. Dengan demikian akan menghasilkan kesetimbangan saat melakukan *hover*.

Untuk melihat besar lebar sinyal yang diberikan ESC pada motor BLDC dapat dilakukan dengan menyambungkan *pin* PWM yang dihasilkan oleh mikrokontroler ke

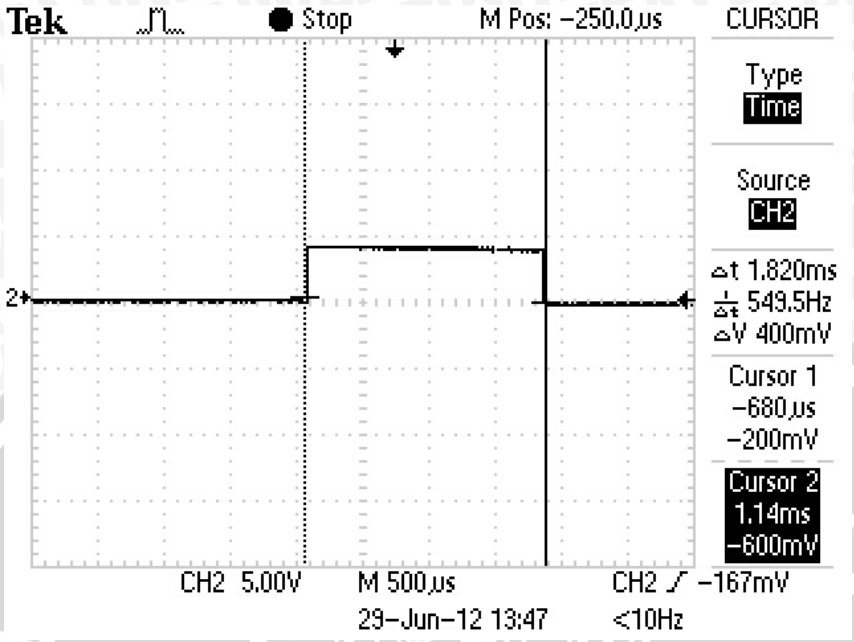
osiloskop. Maka akan terlihat seperti dalam Gambar 5.7, Gambar 5.8 dan Gambar 5.9 dengan lebar pulsa *high* yang berbeda:



Gambar 5.7 Sinyal Kontrol dengan Lebar Pulsa *High* 1520ms



Gambar 5.8 Sinyal Kontrol dengan Lebar Pulsa *High* 1780ms



Gambar 5.9 Sinyal Kontrol dengan Lebar Pulsa *High* 1820ms

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 motor BLDC. Pengujian ini juga bertujuan untuk melihat perbandingan antara *thrust* dari perhitungan secara teori dan dari pengambilan data pada saat pengujian sistem.

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 dan mengaktifkan 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.

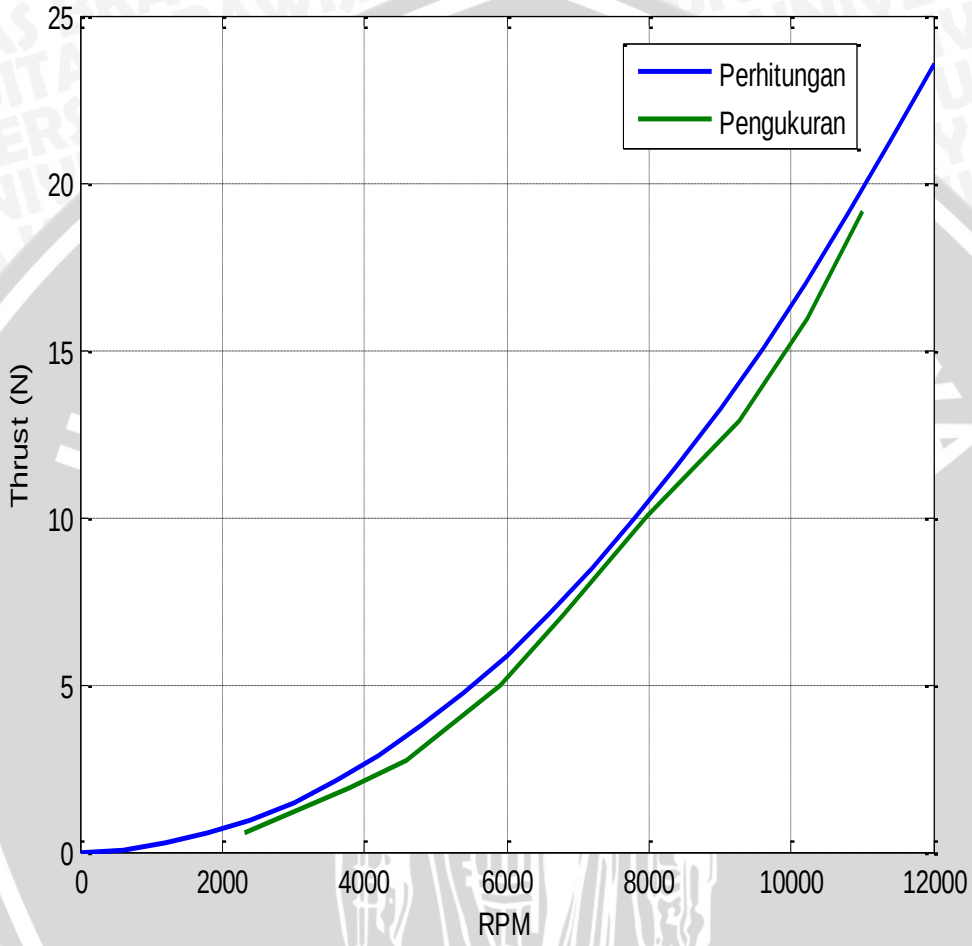
5.4.3 Hasil Pengujian

Setelah melakukan pengujian yang sesuai dengan prosedurnya, maka didapatkan hasil seperti dalam Tabel 5.4:

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

Dengan membandingkan hasil pengujian di atas dengan hasil perhitungan teori sesuai dengan persamaan (4-1), maka didapatkan hasil seperti dalam Gambar 5.10 :



Gambar 5.10 Grafik Perbandingan *Thrust* Perhitungan dan Hasil Pengujian

Dari Tabel 5.4 dan Gambar 5.10 dapat di simpulkan bahwa semakin besar kecepatan putaran dari BLDC maka trust yang di hasilkan dari setiap BLDC semakin besar.

5.5 Pengujian Keseluruhan

Pengujian ini bertujuan untuk melihat respon kesetimbangan sistem *quadrocopter* secara keseluruhan yang mencakup gerak rotasional *yaw*, *pitch*, *roll* yang di butuhkan saat *quadrocopter* melakukan *hover* di udara.

5.5.1 Peralatan Pengujian

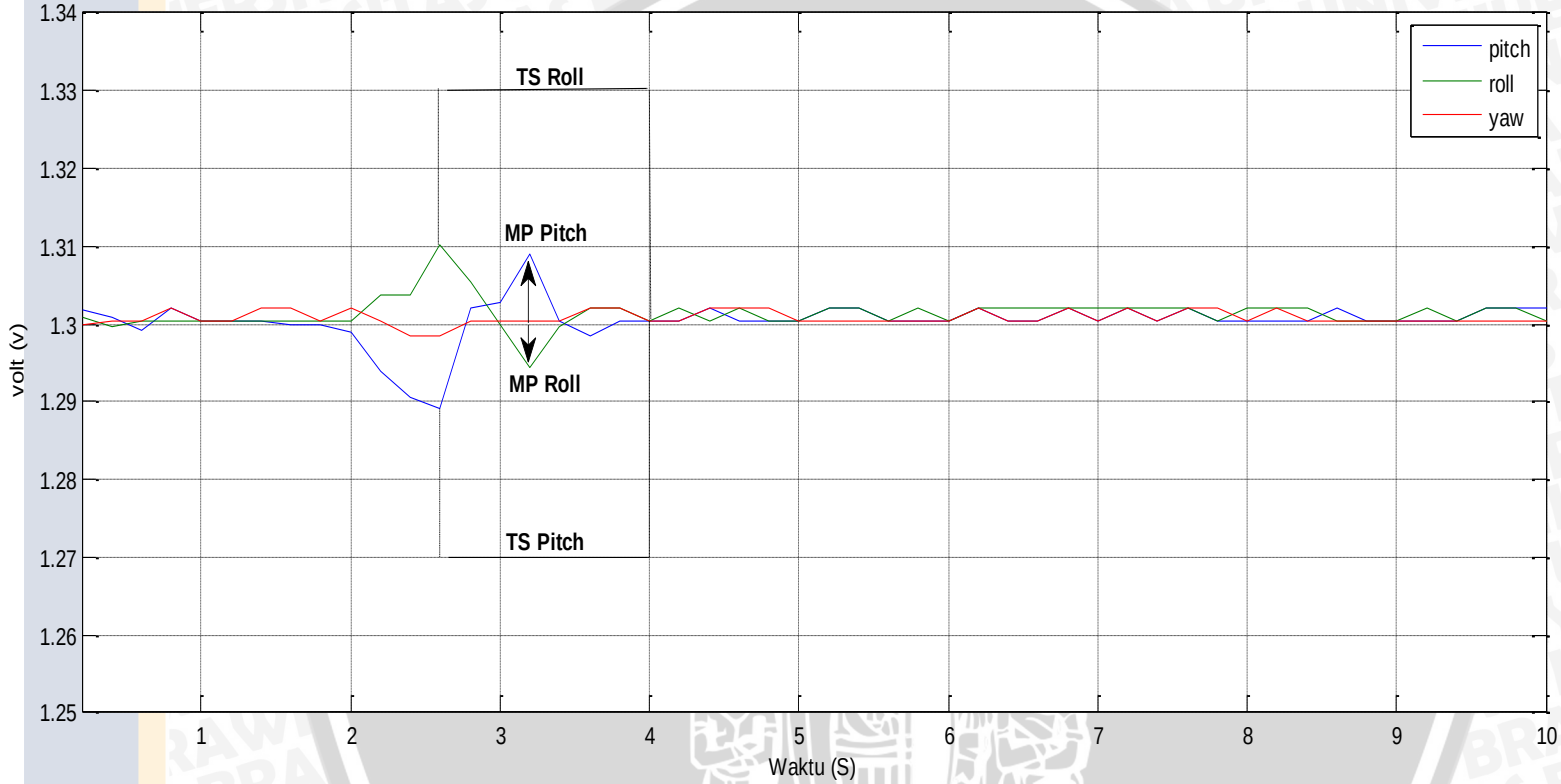
1. *Converter* USB *to* RS232 PL2303
2. Komputer
3. Minimum Sistem ATMega8
4. Beberapa buah kabel

5.5.2 Prosedur Pengujian

Quadrocopter dipegang dan digerakkan sesuai keinginan dan merasakan *trust* yang telah dihasilkan oleh keenpat motor, sehingga dapat melihat respon pengujian kerja sensor *gyro* pada tiap-tiap *axis*-nya. Untuk dapat melihat respon dari sensor *gyro*, 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 5.11.



Gambar 5.11 Grafik Hasil Pengujian Keseluruhan

Dari gambar 5.11 dapat diketahui performansi sistem sebagai berikut:

a. *Time Settling* (ts)

- *Time settling* pada respons kesetimbangan *pitch axis* adalah 1.35s.
- *Time settling* pada respons kesetimbangan *roll axis* adalah 1.35s.

b. *Error Steady State* (ESS)

Nilai osilasi maksimum dalam grafik 5.11 di setiap *axis*-nya adalah 1.302 dengan *setpoint* adalah 1.3, maka :

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

Dengan nilai ESS sebesar 0.15% maka *quadrocopter* mampu melakukan *hover* dengan baik dan akan mendapatkan hasil pencitraan yang maksimal.

c. *Maximum Overshoot* (Mp)

- Untuk *Pitch axis*

  Nilai tertinggi dari gambar 5.11 adalah 1.309, maka:

$$Mp = \frac{1.309 - 1.3}{1.3}\ x\ 100\%$$
$$= 0.7\ \%$$

- Untuk *Roll axis*

  Nilai tertinggi dari gambar 5.11 adalah 1.295, maka:

$$Mp = \frac{1.294 - 1.3}{1.3}\ x\ 100\%$$
$$= 0.5\ \%$$

Dengan prosentase nilai MP yang terjadi, membuktikan bahwa kerja dari *quadrocopter* mampu mempertahankan posisi setimbangnya dari gangguan yang telah diberikan.

BAB VI
PENUTUP

6.1 Kesimpulan

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

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 akan mampu membuat *quadrocopter* terbang pada kecaparan putaran *propeller* sebesar 5400 RPM .

6.2 Saran

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

1. Penambahan data *logger* agar mampu merekam data pada saat benar-benar terbang, sehingga dapat menganalisa sistem *quadrocopter*.
2. *Frame* diganti dengan serat karbon agar bisa lebih ringan dan kuat.

3. Algoritma PID perlu digabungkan dengan kalman filter dengan untuk sistem yang lebih *robust*.
4. Penambahan sensor *accelerometer* agar mampu mealakukan *auto level* saat melakukan *hover* di udara.
5. Penambahan LCD pada *quadrocopter* agar saat pengaturan sensor *gyro*, kerja sensor tersebut tidak terbalik.

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. 1994. *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.

Smith, L. C. 1979. *Fundamentals of control theory.* Deskbook issue.

Jantzen, J. 2001. *A robustness study of fuzzy control rules,in* EUFIT (ed.), *Proceedings Fifth European Congres of fuzzy and technologies*, ELITE Foundation, Promenade 9, D-52076 Aachen, pp. 1222–1227

LAMPIRAN 1

Gambar Alat

Quadrocopter Tampak Samping

Quadrocopter Tampak Atas

LAMPIRAN 2

Datasheet Komponen

Features

- High performance, low power Atmel® AVR® 8-bit microcontroller
- Advanced RISC architecture
  - 131 powerful instructions – most single clock cycle execution
  - 32 x 8 general purpose working registers
  - Fully static operation
  - Up to 20 MIPS throughput at 20MHz
  - On-chip 2-cycle multiplier
- High endurance non-volatile memory segments
  - 4/8/16 Kbytes of in-system self-programmable flash program memory
  - 256/512/512 bytes EEPROM
  - 512/1K/1Kbytes internal SRAM
  - Write/erase cyles: 10,000 flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C[1]
  - Optional boot code section with independent lock bits
    - In-system programming by on-chip boot program
    - True read-while-write operation
  - Programming lock for software security
- QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix acquisition
  - Up to 64 sense channels
- Peripheral features
  - Two 8-bit timer/counters with separate prescaler and compare mode
  - One 16-bit timer/counter with separate prescaler, compare mode, and capture mode
  - Real time counter with separate oscillator
  - Six PWM channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
  - 6-channel 10-bit ADC in PDIP Package
  - Programmable serial USART
  - Master/slave SPI serial interface
  - Byte-oriented 2-wire serial interface (Philips I²C compatible)
  - Programmable watchdog timer with separate on-chip oscillator
  - On-chip analog comparator
  - Interrupt and wake-up on pin change
- Special microcontroller features
  - DebugWIRE on-chip debug system
  - Power-on reset and programmable brown-out detection
  - Internal calibrated oscillator
  - External and internal interrupt sources
  - Five sleep modes: Idle, ADC noise reduction, power-save, power-down, and standby
- I/O and packages
  - 23 programmable I/O lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating voltage:
  - 1.8V - 5.5V for Atmel ATmega48V/88V/168V
  - 2.7V - 5.5V for Atmel ATmega48/88/168
- Temperature range:
  - -40°C to 85°C
- Speed grade:
  - ATmega48V/88V/168V: 0 - 4MHz @ 1.8V - 5.5V, 0 - 10MHz @ 2.7V - 5.5V
  - ATmega48/88/168: 0 - 10MHz @ 2.7V - 5.5V, 0 - 20MHz @ 4.5V - 5.5V
- Low power consumption
  - Active mode:
    - 250µA at 1MHz, 1.8V
    - 15µA at 32kHz, 1.8V (including oscillator)
  - Power-down mode:
    - 0.1µA at 1.8V

Note: 1. See "Data retention" on page 8 for details.

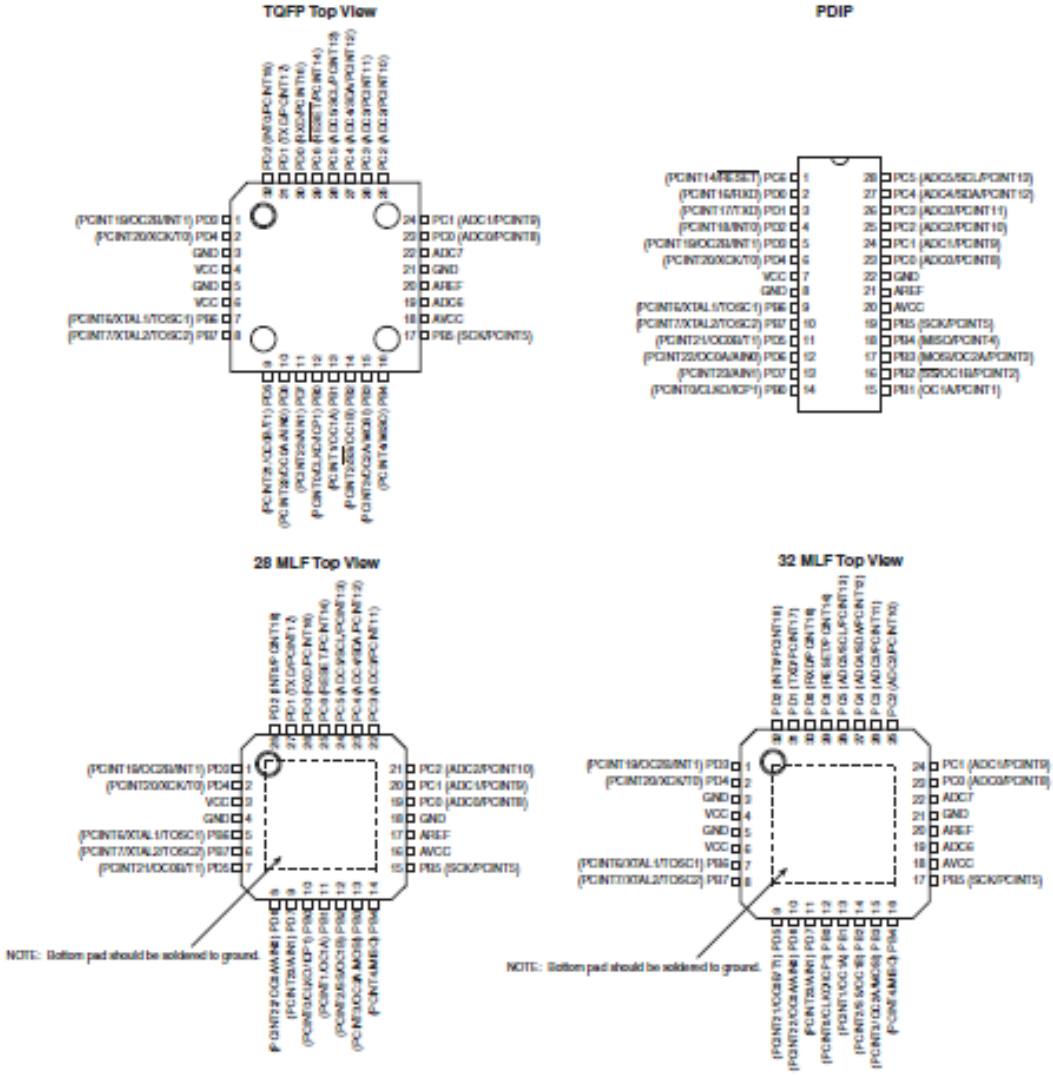
8-bit Atmel Microcontroller with 4/8/16K Bytes In-System Programmable Flash

ATmega48/V
ATmega88/V
ATmega168/V

Rev. 2545T–AVR–05/11

1. Pin configurations

Figure 1-1. Pinout Atmel ATmega48/88/168.



1.1 Pin descriptions

1.1.1 VCC

Digital supply voltage.

1.1.2 GND

Ground.

1.1.3 Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB7..6 is used as TOSC2..1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

The various special features of Port B are elaborated in "Alternate functions of port B" on page 78 and "System clock and clock options" on page 27.

1.1.4 Port C (PC5:0)

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5..0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

1.1.5 PC6/RESET

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. The minimum pulse length is given in Table 29-3 on page 307. Shorter pulses are not guaranteed to generate a Reset.

The various special features of Port C are elaborated in "Alternate functions of port C" on page 81.

1.1.6 Port D (PD7:0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up

resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

The various special features of Port D are elaborated in "Alternate functions of port D" on page 84.

1.1.7 AV_{CC}

AV_{CC} is the supply voltage pin for the A/D Converter, PC3:0, and ADC7:6. It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter. Note that PC6..4 use digital supply voltage, V_{CC}.

1.1.8 AREF

AREF is the analog reference pin for the A/D Converter.

1.1.9 ADC7:6 (TQFP and QFN/MLF package only)

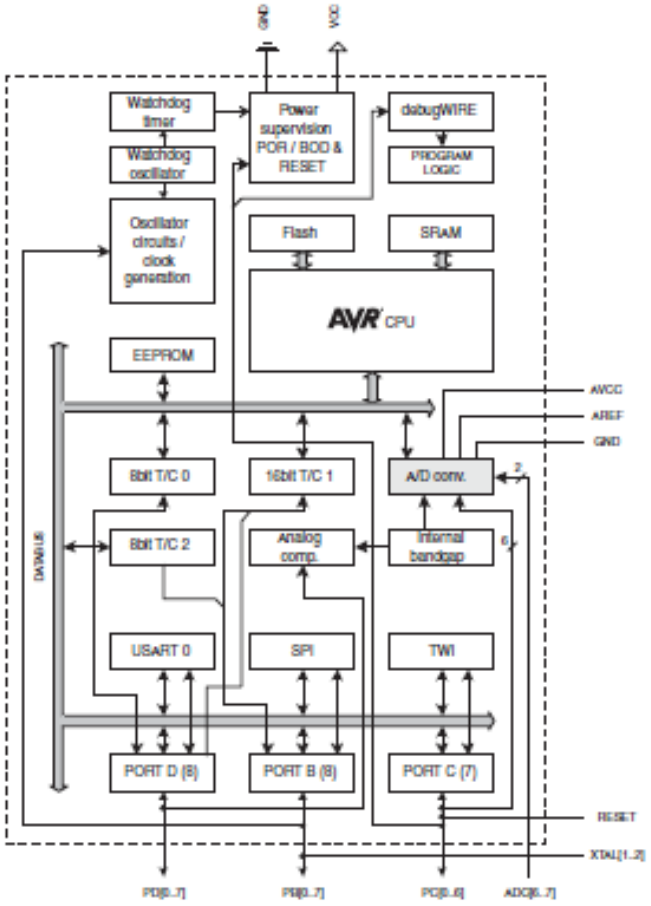
In the TQFP and QFN/MLF package, ADC7:6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

2. Overview

The Atmel ATmega48/88/168 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega48/88/168 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

2.1 Block diagram

Figure 2-1. Block diagram.



The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting

architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The Atmel ATmega48/88/168 provides the following features: 4K/8K/16K bytes of In-System Programmable Flash with Read-While-Write capabilities, 256/512/512 bytes EEPROM, 512/1K/1K bytes SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte-oriented 2-wire Serial Interface, an SPI serial port, a 6-channel 10-bit ADC (8 channels in TQFP and QFN/MLF packages), a programmable Watchdog Timer with internal Oscillator, and five software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, USART, 2-wire Serial Interface, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption.

Atmel offers the QTouch Library for embedding capacitive touch buttons, sliders and wheels functionality into AVR microcontrollers. The patented charge-transfer signal acquisition offers robust sensing and includes fully debounced reporting of touch keys and includes Adjacent Key Suppression® (AKS®) technology for unambigiuous detection of key events. The easy-to-use QTouch Suite toolchain allows you to explore, develop and debug your own touch applications.

The device is manufactured using the Atmel high density non-volatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional non-volatile memory programmer, or by an On-chip Boot program running on the AVR core. The Boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega48/88/168 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega48/88/168 AVR is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, Program Debugger/Simulators, In-Circuit Emulators, and Evaluation kits.

2.2 Comparison between Atmel ATmega48, Atmel ATmega88, and Atmel ATmega168

The ATmega48, ATmega88 and ATmega168 differ only in memory sizes, boot loader support, and interrupt vector sizes. Table 2-1 summarizes the different memory and interrupt vector sizes for the three devices.

Table 2-1. Memory size summary.

Device	Flash	EEPROM	RAM	Interrupt vector size
ATmega48	4Kbytes	256Bytes	512Bytes	1 Instruction word/vector
ATmega88	8Kbytes	512Bytes	1Kbytes	1 Instruction word/vector
ATmega168	16Kbytes	512Bytes	1Kbytes	2 Instruction words/vector

ATmega88 and ATmega168 support a real Read-While-Write Self-Programming mechanism. There is a separate Boot Loader Section, and the SPM instruction can only execute from there. In ATmega48, there is no Read-While-Write support and no separate Boot Loader Section. The SPM instruction can execute from the entire Flash.

3. Resources

A comprehensive set of development tools, application notes and datasheets are available for download on http://www.atmel.com/avr.

4. Data retention

Reliability Qualification results show that the projected data retention failure rate is much less than 1 PPM over 20 years at 85°C or 100 years at 25°C.

5. About code examples

This documentation contains simple code examples that briefly show how to use various parts of the device. These code examples assume that the part specific header file is included before compilation. Be aware that not all C compiler vendors include bit definitions in the header files and interrupt handling in C is compiler dependent. Please confirm with the C compiler documentation for more details.

For I/O Registers located in extended I/O map, "IN", "OUT", "SBIS", "SBIC", "CBI", and "SBI" instructions must be replaced with instructions that allow access to extended I/O. Typically "LDS" and "STS" combined with "SBRS", "SBRC", "SBR", and "CBR".

6. Capacitive touch sensing

The Atmel QTouch Library provides a simple to use solution to realize touch sensitive interfaces on most Atmel AVR microcontrollers. The QTouch Library includes support for the QTouch and QMatrix acquisition methods.

Touch sensing can be added to any application by linking the appropriate Atmel QTouch Library for the AVR Microcontroller. This is done by using a simple set of APIs to define the touch channels and sensors, and then calling the touch sensing API's to retrieve the channel information and determine the touch sensor states.

The QTouch Library is FREE and downloadable from the Atmel website at the following location: www.atmel.com/qtouchlibrary. For implementation details and other information, refer to the Atmel QTouch Library User Guide - also available for download from the Atmel website.

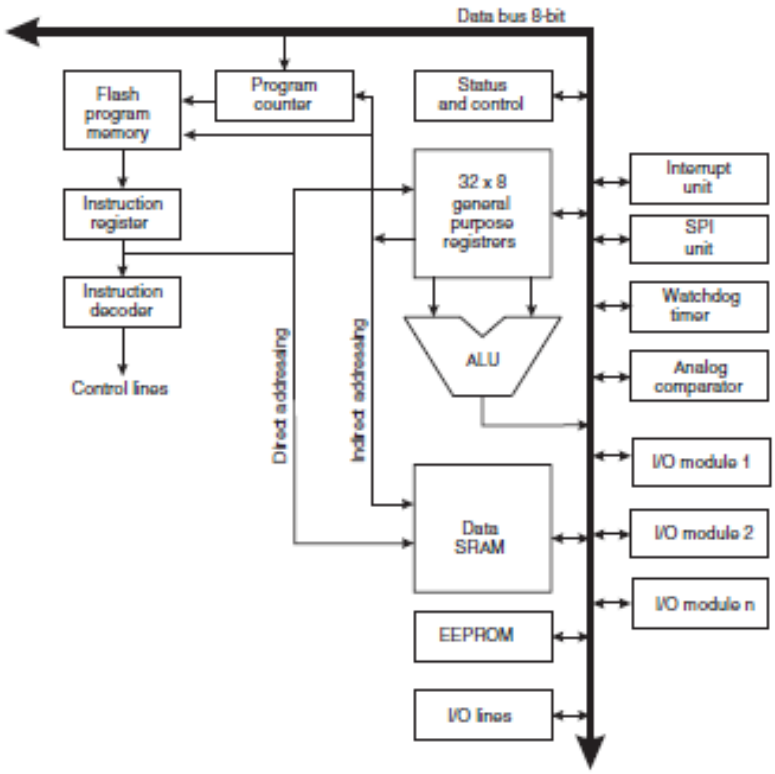
7. AVR CPU core

7.1 Overview

This section discusses the AVR core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts.

7.2 Architectural overview

Figure 7-1. Block diagram of the AVR architecture.



In order to maximize performance and parallelism, the AVR uses a Harvard architecture – with separate memories and buses for program and data. Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is In-System Reprogrammable Flash memory.

The fast-access Register File contains 32 × 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two operands are output from the Register File, the operation is executed, and the result is stored back in the Register File – in one clock cycle.

Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing – enabling efficient address calculations. One of the these address pointers can also be used as an address pointer for look up tables in Flash program memory. These added function registers are the 16-bit X-register, Y-register, and Z-register, described later in this section.

The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the Status Register is updated to reflect information about the result of the operation.

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16-bit or 32-bit instruction.

Program Flash memory space is divided in two sections, the Boot Program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM instruction that writes into the Application Flash memory section must reside in the Boot Program section.

During interrupts and subroutine calls, the return address Program Counter (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM, and consequently the Stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the Reset routine (before subroutines or interrupts are executed). The Stack Pointer (SP) is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture.

The memory spaces in the AVR architecture are all linear and regular memory maps.

A flexible interrupt module has its control registers in the I/O space with an additional Global Interrupt Enable bit in the Status Register. All interrupts have a separate Interrupt Vector in the Interrupt Vector table. The interrupts have priority in accordance with their Interrupt Vector position. The lower the Interrupt Vector address, the higher the priority.

The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers, SPI, and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the Register File, 0x20 - 0x5F. In addition, the ATmega48/88/168 has Extended I/O space from 0x60 - 0xFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

7.3 ALU – Arithmetic Logic Unit

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format. See "Instruction set summary" on page 347 for a detailed description.

7.4 Status register

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the Status Register is updated after all ALU operations, as specified in the Instruction Set Reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

7.4.1 SREG – AVR Status Register

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

- **Bit 7 – I: Global interrupt enable**

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

- **Bit 6 – T: Bit copy storage**

The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the Register File can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.

- **Bit 5 – H: Half carry flag**

The Half Carry Flag H indicates a Half Carry in some arithmetic operations. Half Carry Is useful in BCD arithmetic. See the "Instruction Set Description" for detailed information.

- **Bit 4 – S: Sign bit, S = N ⊕ V**

The S-bit is always an exclusive or between the Negative Flag N and the Two's Complement Overflow Flag V. See the "Instruction Set Description" for detailed information.

- **Bit 3 – V: Two's complement overflow flag**

The Two's Complement Overflow Flag V supports two's complement arithmetics. See the "Instruction Set Description" for detailed information.

- **Bit 2 – N: Negative flag**

The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the "Instruction Set Description" for detailed information.

- **Bit 1 – Z: Zero flag**

The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the "Instruction Set Description" for detailed information.

- **Bit 0 – C: Carry flag**

The Carry Flag C indicates a carry in an arithmetic or logic operation. See the "Instruction Set Description" for detailed information.

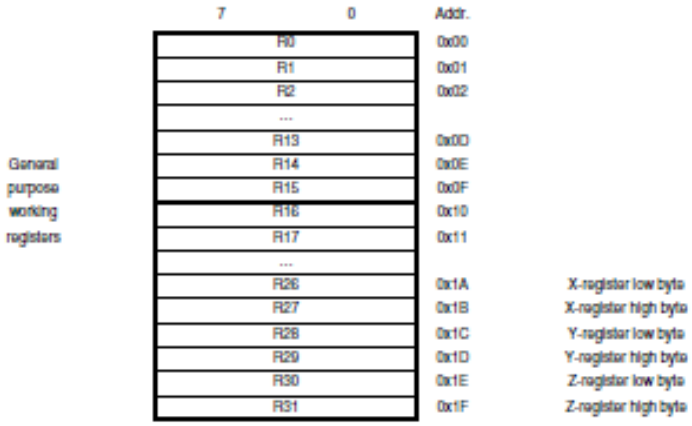
7.5 General purpose register file

The register file is optimized for the AVR enhanced RISC instruction set. In order to achieve the required performance and flexibility, the following input/output schemes are supported by the register file:

- One 8-bit output operand and one 8-bit result input
- Two 8-bit output operands and one 8-bit result input
- Two 8-bit output operands and one 16-bit result input
- One 16-bit output operand and one 16-bit result input

Figure 7-2 shows the structure of the 32 general purpose working registers in the CPU.

Figure 7-2. AVR CPU general purpose working registers.



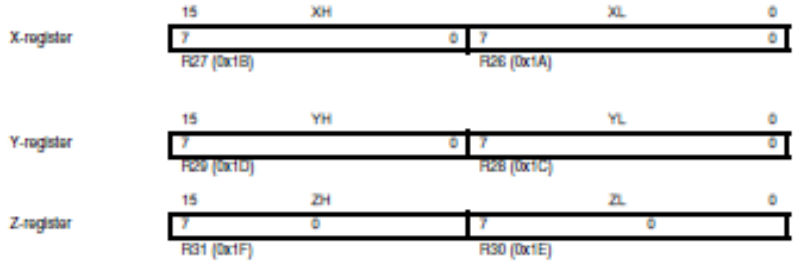
Most of the instructions operating on the register file have direct access to all registers, and most of them are single cycle instructions.

As shown in Figure 7-2, each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user Data Space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y- and Z-pointer registers can be set to index any register in the file.

7.5.1 The X-register, Y-register, and Z-register

The registers R26..R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the data space. The three indirect address registers X, Y, and Z are defined as described in Figure 7-3.

Figure 7-3. The X-, Y-, and Z-registers.



In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the instruction set reference for details).

7.6 Stack pointer

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack Pointer Register always points to the top of the Stack. Note that the Stack is implemented as growing from higher memory locations to lower memory locations. This implies that a Stack PUSH command decreases the Stack Pointer.

The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. This Stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. The Stack Pointer must be set to point above 0x0100, preferably RAMEND. The Stack Pointer is decremented by one when data is pushed onto the Stack with the PUSH instruction, and it is decremented by two when the return address is pushed onto the Stack with subroutine call or interrupt. The Stack Pointer is incremented by one when data is popped from the Stack with the POP instruction, and it is incremented by two when data is popped from the Stack with return from subroutine RET or return from interrupt RETI.

The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present.

7.6.1 SPH and SPL – Stack pointer high and stack pointer low register

Bit	15	14	13	12	11	10	9	8	
0x3E (0x5E)	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
0x3D (0x5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	
	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	

7.7 Instruction execution timing

This section describes the general access timing concepts for instruction execution. The AVR CPU is driven by the CPU clock clk_{CPU}, directly generated from the selected clock source for the chip. No internal clock division is used.

Figure 7-4 shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast-access Register File concept. This is the basic pipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

Figure 7-4. The parallel instruction fetches and instruction executions.

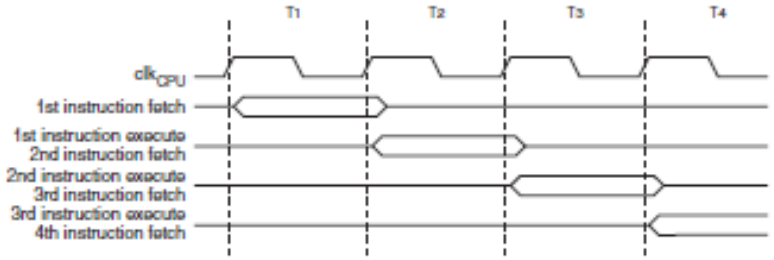
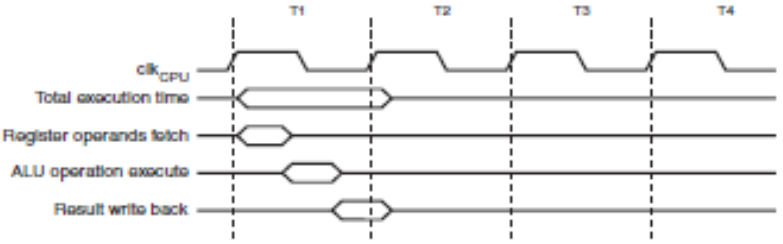


Figure 7-5 shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

Figure 7-5. Single cycle ALU operation.



7.8 Reset and interrupt handling

The AVR provides several different interrupt sources. These interrupts and the separate Reset Vector each have a separate program vector in the program memory space. All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt. Depending on the Program Counter value, interrupts may be automatically disabled when Boot Lock bits BLB02 or BLB12 are programmed. This feature improves software security. See the section "Memory programming" on page 285 for details.

The lowest addresses in the program memory space are by default defined as the Reset and Interrupt Vectors. The complete list of vectors is shown in "Interrupts" on page 56. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 – the External Interrupt Request 0. The Interrupt Vectors can be moved to the start of the Boot Flash section by setting the IVSEL bit in the MCU Control Register (MCUCR). Refer to "Interrupts" on page 56 for more information. The Reset Vector can also be moved to the start of the Boot Flash section by programming the BOOTRST Fuse, see "Boot loader support – Read-while-write self-programming, Atmel ATmega88 and Atmel ATmega168" on page 269.

When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed.

There are basically two types of interrupts. The first type is triggered by an event that sets the Interrupt Flag. For these interrupts, the Program Counter is vectored to the actual Interrupt Vector in order to execute the interrupt handling routine, and hardware clears the corresponding Interrupt Flag. Interrupt Flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the Interrupt Flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the Global Interrupt Enable bit is cleared, the corresponding Interrupt Flag(s) will be set and remembered until the Global Interrupt Enable bit is set, and will then be executed by order of priority.

The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have Interrupt Flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered.

When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.

Note that the Status Register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

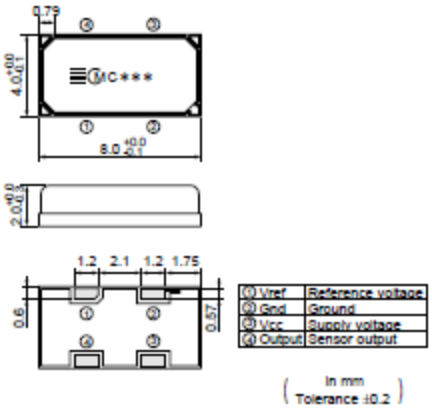
When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction. The following example shows how this can be used to avoid interrupts during the timed EEPROM write sequence.

Angular Rate Sensors (ENC Series)

ENC-03R

This product is an angular rate sensor that uses the phenomenon of Coriolis force, which is generated when a rotational angular rate is applied to the vibrator.
Murata's original, small ceramic bimorph vibrator and simple Cap-Base structure realize their ultra-small size, under 0.1cc. Their small shape and light weight increase flexibility of installment and help to downsize your equipment.
This surface mountable device can, be mounted by an automatic surface mounter.

■ Features
1. Ultra-small and ultra-lightweight
2. Quick response
3. Low driving voltage, low current consumption
4. Lead type : SMD
5. Reflow soldering (standard peak temp. 250°C)

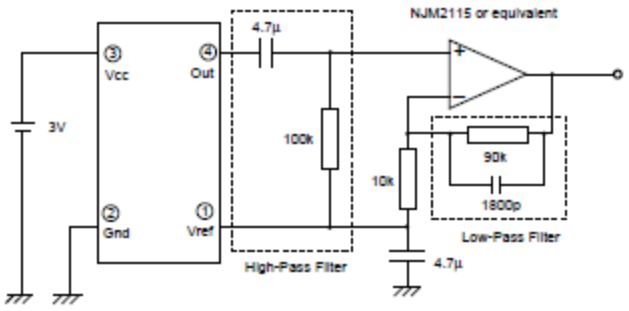


Part Number	Resonance Frequency (kHz)	Supply Voltage	Maximum Angular Velocity (deg./sec.)	Output (at Angular Velocity=0)	Scale Factor	Linearity (%FS)	Response (Hz)	Weight (g)
ENC-03RC-R	30.8	2.7-5.25V	±300	1.35Vdc	0.67mV/deg./sec.	±5	50	0.2
ENC-03RD-R	32.2	2.7-5.25V	±300	1.35Vdc	0.67mV/deg./sec.	±5	50	0.2

Operating Temperature Range: -5°C to 75°C Storage Temp. Range: -30°C to 85°C

1

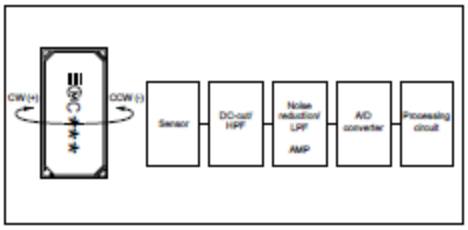
■ Sample Amplifier Circuit



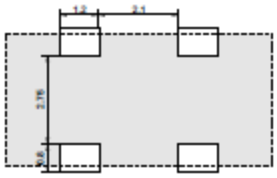
The high-pass filter's cut-off frequency in this circuit is approx. 0.3Hz.
The low-pass filter's cut-off frequency in this circuit is approx. 1kHz.

■ Application

1. One sensor detects rotation on one axis. If two axes are to be detected in the same equipment, two different types of sensors (EMC-03RC and EMC-03RD) should be used.
2. To reduce the effect of temperature drift (due to change in ambient temperature), a high-pass filter must be connected to sensor output to eliminate the DC component.
3. To suppress the output noise component at around 30-33kHz (resonant frequency of sensor element), a low-pass filter that has a higher cut-off frequency than the required response frequency must be connected to the sensor output.



■ Dimensions of Land Pattern

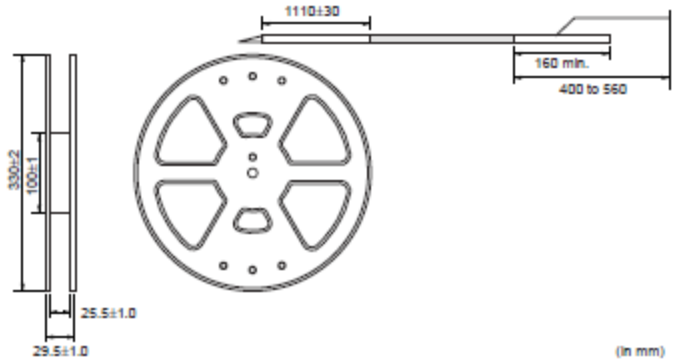


(in mm)

Packaging

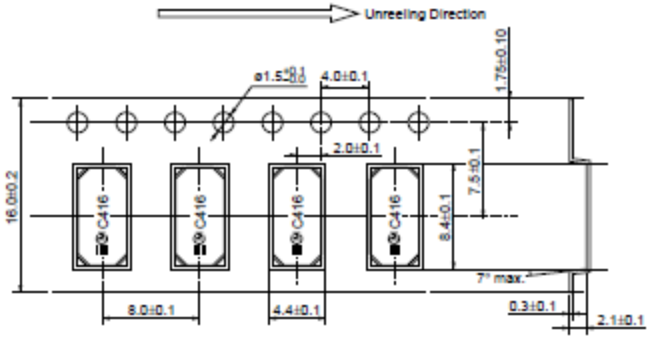
1

■ Dimensions of Reel



(In mm)

■ Dimensions of Plastic Tape



(In mm)

1 Notice

■ Storage and Operating Conditions

1. Incorrect handling may affect sensor characteristics. Please note the following precautions:
   A. Do not subject the sensor to shock or vibration that exceeds the rated limit.
   B. Do not install or store the sensor in a location where condensation is likely to form on it.
   C. Do not install or store the sensor in a location where water may splash directly on it.
   D. Do not install or store the sensor in a location in which it is likely to be exposed to salt water or corrosive vapor.
2. Do not use or store the products in a corrosive atmosphere, especially where chloride gas, sulfide gas, acid, alkali, salt or the like are present. Also avoid exposure to moisture. Store the products where the temperature and relative humidity do not exceed 5 to 40°C and 20 to 75%R.H. Srore the products in a sealed bag and use with in 6 months.

■ Soldering and Mounting

1. Do not mount the sensor on an electric circuit line arranged on the circuit board, because the sensor has an electric circuit on its back.
2. Please ensure that the interference between the resonance frequency of the gyro and any other signals.
3. This product does not support flow soldering.

■ Handling

1. Precision electronic parts, such as ICs, are used for the sensor; therefore, it is necessary to take anti-electrostatic precautions when handling.
2. Do not wash the sensor, as it is not water-resistant.
3. Do not disassemble.
4. Do not touch the terminal directly.

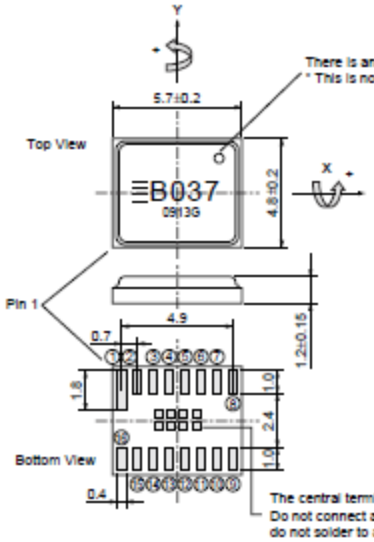
Angular Rate Sensors (ENC Series)

muRata

ENC-03W

This product is an angular rate sensor that uses the phenomenon of Coriolis force, which is generated when a rotational angular rate is applied to the vibrator.
Their small shape and light weight increase flexibility of installment and help to downsize your equipment.
This surface mountable device can be mounted by an automatic surface mounter.

■ Features
1. 2 axes in 1 package!
2. High performance with calibration function in ASIC
   - Improvement of zero-rate level drift
   - The reduction of zero-rate level (bias)
   - Reduction of the change of the sensitivity - temperature characteristic
3. Built in amplifier (AMP2)
   - Can set an amplifier gain and the filter characteristic in external CR freely.
4. Built in charge switch (HPF reset)



Pin Number	Symbol	Function
1	NC	-
2	NC	-
3	NC	-
4	APO X	The AMP2 output of the X-axis
5	AFB X	The return of AMP2 of the X-axis
6	AIN X	The input of AMP2 of the X-axis
7	OUT X	The output of the X-axis (Not AMP2)
8	Vref	Reference voltage
9	GND	Ground
10	OUT Y	The output of the Y-axis (Not AMP2)
11	AIN Y	The input of the AMP2 of the Y-axis
12	AFB Y	The return of AMP2 of the Y-axis
13	APO Y	The AMP2 output of the Y-axis
14	SCT	The control terminal of charge switch
15	Vcc	Power supply
16	NC	-

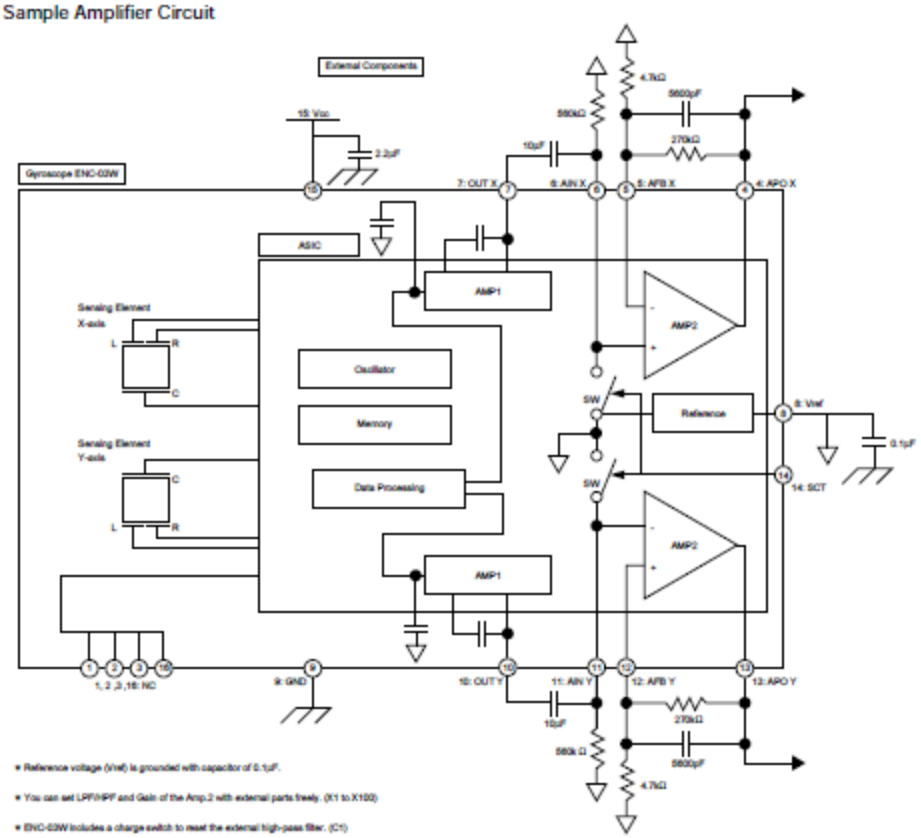
* NC: No Connect (Do not connect to any other circuit line.)

Part Number	Supply Voltage	Maximum Angular Velocity (deg./sec.)	Output (at Angular Velocity=0)	Scale Factor	Linearity (%FS)	Response (Hz)	Weight (g)
ENC-03WB-R	2.7-3.6V	±300	1.35Vdc	0.67mV/deg./sec.	±5	50	0.1

Operating Temperature Range: -5°C to 75°C Storage Temp. Range: -30°C to 85°C

2

■ Sample Amplifier Circuit



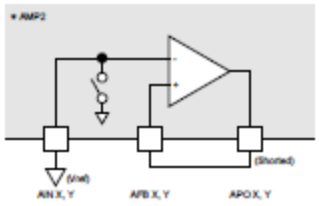
• Reference voltage (Vref) is grounded with capacitor of 0.1µF.

• You can set LPF/HPF and Gain of the Amp.2 with external parts freely. (X1 to X100)

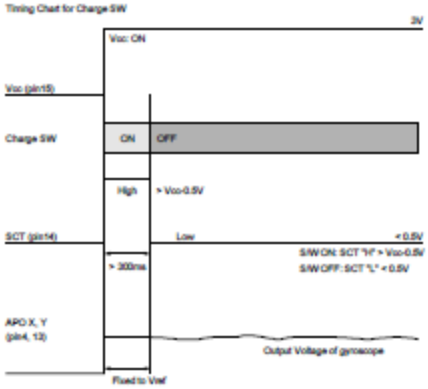
• ENC-03W includes a charge switch to reset the external high-pass filter. (C1)

• When you do not use AMP2, please short-circuit in AFB and APO, and connect AIN to Vref.

X-axis:
④ APO X ->① AFB X shorted
① AFB X ->④ APO X shorted
① AIN X ->① Vref connected
Y-axis:
⑬ AIN Y ->⑬ Vref shorted
⑫ AFB Y ->⑫ APO Y shorted
⑬ APO Y ->⑫ AFB Y connected



Timing Chart for Charge SW

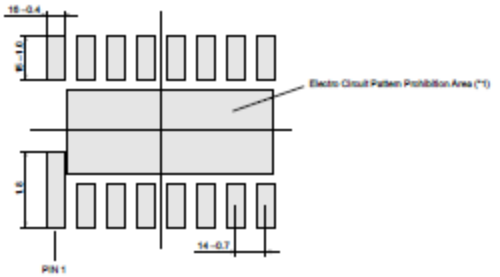


Continued on the following page.

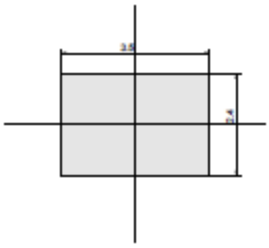
Continued from the preceding page.

■ Dimensions of Land Pattern

The Land Pattern Dimensions for SMD. (1st layer)



The GND Pattern for Electric Shield. (2nd layer) (*2)



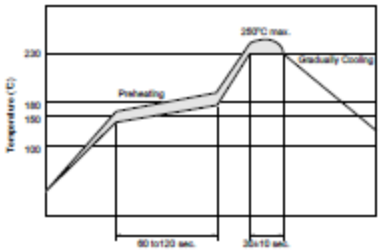
(in mm)

(*1) Do not make the electro circuit pattern at the prohibition area to prevent contact with the central terminals of product's back side and to prevent interference with the other signal.
(Prohibition area: the central hatching area at 1st layer)

(*2) Please make the GND pattern for electric shield (2nd layer) to reduce interference with the other signal.

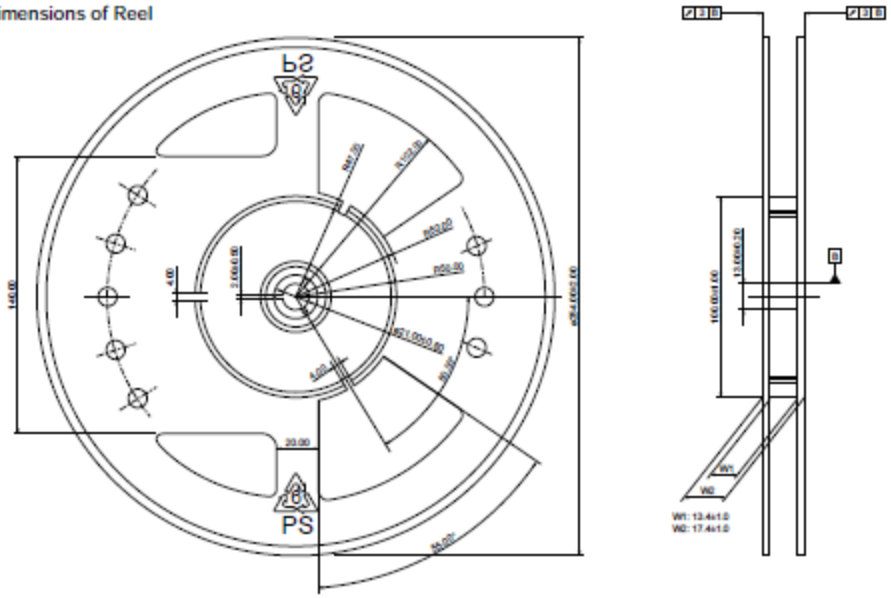
* Mechanical stress to the mounted board, such as bending and pushing, affects the zero-rate level of the product. Zero-rate level can change slightly at the moment when mechanical stress is being changed. If the hardness of mounted board is high, the zero-rate level changes very little, and the influence of the zero-rate level is changed by the condition of fixed position and shape of the mounted board. Therefore, please ensure that your product has been evaluated in view of your specifications with our product being mounted to your product.

■ Reflow Chart



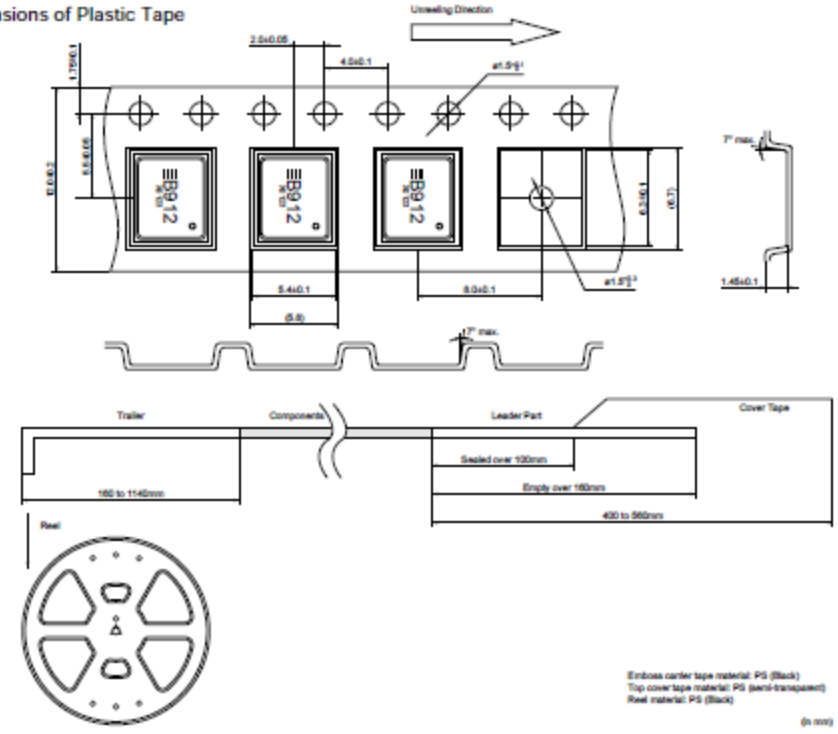
Packaging

■ Dimensions of Reel



(in mm)

■ Dimensions of Plastic Tape



Emboss carrier tape material: PS (Black)
Top cover tape material: PS (semi-transparent)
Reel material: PS (Black)

(in mm)

Notice

■ Storage and Operating Conditions

1. Incorrect handling may affect sensor characteristics. Please note the following precautions:
   A. Do not subject the sensor to shock or vibration that exceeds the rated limit.
   B. Do not install or store the sensor in a location where condensation is likely to form on it.
   C. Do not install or store the sensor in a location where water may splash directly on it.
   D. Do not install or store the sensor in a location in which it is likely to be exposed to salt water or corrosive vapor.
2. Do not use or store the products in a corrosive atmosphere, especially where chloride gas, sulfide gas, acid, alkali, salt or the like are present. Also avoid exposure to moisture. Store the products where the temperature and relative humidity do not exceed 5 to 40°C and 20 to 75%R.H. Store the products in sealed bag and use with in 6 months. In addition, this product should be treated as MSL3 of JEDEC J-STD-020D.1. After unsealing the bag, at less than 30°C/60%R.H, do reflow soldering on this product within 7 days.

■ Soldering and Mounting

1. Do not mount the sensor on an electric circuit line arranged on the circuit board, because the sensor has an electric circuit on its back.
2. Please ensure that the interference between the resonance frequency of the gyro and any other signals.
3. This product does not support hand soldering.
4. This product does not support flow soldering.

■ Handling

1. Precision electronic parts, such as ICs, are used for the sensor; therefore, it is necessary to take anti-electrostatic precautions when handling.
2. Do not wash the sensor, as it is not water-resistant.
3. Do not disassemble.
4. Do not touch the terminal directly.

■ Minimum Quantity

Part Number	ø330mm Reel	ø254mm Reel
ENC-03R	2000 pcs	–
ENC-03W	–	1500 pcs

● Part Numbering

Angular Rate Sensors (ENC Series)

(Part Number) EN ❶ | C-03RC ❷ | ☐ ❸ | -R ❹

❶Product ID
❷Type
❸Individual Specification Code
❹Packaging

EU RoHS Compliant

- All the products in this catalog comply with EU RoHS.
- EU RoHS is "the European Directive 2002/95/EC on the Restriction of the Use of Certain Hazardous Substances in Electrical and Electronic Equipment."
- For more details, please refer to our website 'Murata's Approach for EU RoHS' (http://www.murata.com/info/rohs.html).

⚠Note:

1. Export Control
<For customers outside Japan>
No Murata products should be used or sold, through any channels, for use in the design, development, production, utilization, maintenance or operation of, or otherwise contribution to (1) any weapons (Weapons of Mass Destruction [nuclear, chemical or biological weapons or missiles] or conventional weapons) or (2) goods or systems specially designed or intended for military end-use or utilization by military end-users.
<For customers in Japan>
For products which are controlled items subject to the "Foreign Exchange and Foreign Trade Law" of Japan, the export license specified by the law is required for export.

2. Please contact our sales representatives or product engineers before using the products in this catalog for the applications listed below, which require especially high reliability for the prevention of defects which might directly damage a third party's life, body or property, or when one of our products is intended for use in applications other than those specified in this catalog.
① Aircraft equipment
② Aerospace equipment
③ Undersea equipment
④ Power plant equipment
⑤ Medical equipment
⑥ Transportation equipment (vehicles, trains, ships, etc.)
⑦ Traffic signal equipment
⑧ Disaster prevention / crime prevention equipment
⑨ Data-processing equipment
⑩ Application of similar complexity and/or reliability requirements to the applications listed above

3. Product specifications in this catalog are as of May 2011. They are subject to change or our products in it may be discontinued without advance notice. Please check with our sales representatives or product engineers before ordering. If there are any questions, please contact our sales representatives or product engineers.

4. Please read rating and ⚠CAUTION (for storage, operating, rating, soldering, mounting and handling) in this catalog to prevent smoking and/or burning, etc.

5. This catalog has only typical specifications because there is no space for detailed specifications. Therefore, please review our product specifications or consult the approval sheet for product specifications before ordering.

6. Please note that unless otherwise specified, we shall assume no responsibility whatsoever for any conflict or dispute that may occur in connection with the effect of our and/or a third party's intellectual property rights and other related rights in consideration of your use of our products and/or information described or contained in our catalogs. In this connection, no representation shall be made to the effect that any third parties are authorized to use the rights mentioned above under licenses without our consent.

7. No ozone depleting substances (ODS) under the Montreal Protocol are used in our manufacturing process.

Murata Manufacturing Co., Ltd.
http://www.murata.com/

Head Office
1-10-1, Higashi Kotari, Nagaokakyo-shi, Kyoto 617-8555, Japan
Phone: 81-75-951-9111

International Division
3-29-12, Shibuya, Shibuya-ku, Tokyo 150-0002, Japan
Phone: 81-3-5469-6123 Fax: 81-3-5469-6155 E-mail: intl@murata.co.jp

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()
```

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