TUGAS AKHIR – TM 141585

MEKANISME PENGENDALIAN KECEPATAN *SPIN* GIROSKOP PADA SISTEM *SELF BALANCING BIKE*

MARIO MUHAMMAD HAFIDH
NRP. 02111240000122

Dosen Pembimbing
Arif Wahjudi, S.T., M.T., P.hD.

Laboratorium Perancangan dan Pengembangan Produk
Departemen Teknik Mesin
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
Surabaya
2018

TUGAS AKHIR – TM 141585

MEKANISME PENGENDALIAN KECEPATAN *SPIN* GIROSKOP PADA SISTEM *SELF BALANCING BIKE*

MARIO MUHAMMAD HAFIDH
NRP. 02111240000122

Dosen Pembimbing
Arif Wahjudi, S.T., M.T., P.hD.

Laboratorium Perancangan dan Pengembangan Produk
Departemen Teknik Mesin
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
Surabaya
2018

[Halaman Ini Sengaja Dikosongkan]

FINAL PROJECT – TM 141585

GYROSCOPE'S SPIN VELOCITY MECHANISM CONTROL ON A SELF BALANCING BIKE

MARIO MUHAMMAD HAFIDH
NRP. 02111240000122

Academic Supervisor
Arif Wahjudi, S.T., M.T., P.hD.

Product Design and Development Labroratory
Mechanical Engineering Department
Faculty of Industrial Technology
Sepuluh Nopember Institute of Technology
Surabaya
2018

[Halaman Ini Sengaja Dikosongkan]

MEKANISME PENGENDALIAN KECEPATAN *SPIN* GIROSKOP PADA SISTEM *SELF BALANCING BIKE*

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat Memperoleh Gelar
Sarjana Teknik
Pada
Program studi S-1 Departemen Teknik Mesin
Fakultas Teknik Industri
Institut Teknologi Sepuluh Nopember

Oleh :

MARIO MUHAMMAD HAFIDH
NRP. 021112 40000 122

Disetujui oleh tim penguji tugas akhir :

1. Arif Wahjudi, S.T., M.T., P.hD. ..... (Pembimbing)
   NIP. 197303222001121001
2. Dr. Unggul Wasiwitono, S.T., M.Eng.Sc. ..... (Penguji I)
   NIP. 197805102001121001
3. Latifah Nurahmi, S.T., M.Sc., P.hD. ..... (Penguji II)
   NIP. 1986201712037

SURABAYA
JULI, 2018

MEKANISME PENGENDALIAN KECEPATAN *SPIN* GIROSKOP PADA SISTEM *SELF BALANCING BIKE*

Nama	**: Mario Muhammad Hafidh**
NRP	**: 0211140000122**
Departemen	**: Teknik Mesin FTI-ITS**
Dosen Pembimbing	**: Arif Wahjudi, ST., MT., PhD.**

Abstrak

Pada banyak penelitian, keseimbangan menjadi aspek yang paling disoroti dalam penggunaan sepeda. Posisi tubuh, kecepatan kayuh dan arah kendali stang sepeda mempengaruhi keseimbangan saat sepeda melaju. Masalahnya, keseimbangan yang merupakan kunci utama ini justru menjadi sebuah hambatan seseorang dalam mengendarai sepeda. Dalam beberapa kasus, ketidakseimbangan bahkan menjadi sumber penyebab kecelakaan sepeda. Maka, diperlukan sebuah mekanisme terotomatisasi yang mampu menjaga kondisi sepeda untuk tetap dalam keadaan seimbang pada saat statis maupun dinamis. Tugas akhir ini bertujuan untuk membuat sistem pengendalian yang berfungsi mengatur kecepatan giroskop sehingga mampu membantu menjaga keseimbangan sepeda.

Mekanisme pengendalian keseimbangan sepeda yang teorotomasi dibuat dengan memanfaatkan efek putaran *gyroscope* untuk menjaga kestabilannya yang menggunakan motor DC *brushless* sebagai aktuatornya. Untuk menjaga keseimbangan maka besar kecepatan giroskop diatur menggunakan PID *controller* sehingga putarannya menyesuaikan sudut jatuhnya sepeda. Mekanisme ini memanfaatkan input *error* berupa sudut jatuhnya sepeda dari pembacaan sensor IMU MPU6050 yang kemudian diteruskan ke dalam mikrokontroller. Selanjutnya dilakukan proses *tunning* nilai Kp dan Ki sementara untuk mendapatkan nilai F terendah. Selanjutnya nilai Kp dan Ki yang relevan didapatkan melalui pembuatan sistem *neural network* yang akhirnya diinputkan kembali ke dalam mikrokontroller untuk

mengatur kecepatan putar gimbal serta menjaga keseimbangan posisi sepeda.

Keluaran yang dihasilkan dari tugas akhir ini adalah perancangan sistem kendali yang telah dibangun mampu mengendalikan *prototype* sepeda. Pada pengendalian PI-NN dapat mengurangi rata-rata *overshoot* sebesar 6,8178% menjadi 6,567% (turun 3,679%), menurunkan *rise time* sebesar 1,2555 detik menjadi 1,2166 detik (turun 3,069%) dan *steady state error* sebesar 0,543 derajat menjadi 0,246 derajat (15,15%).

Kata kunci: sepeda; keseimbangan; *gyroscope*; *neural network*; *proportional-integral-derivative control*; mikrokontroller

GYROSCOPE'S SPIN VELOCITY MECHANISM CONTROL ON A SELF BALANCING BIKE

Name : **Mario Muhammad Hafidh**
Reg. Number : **0211140000122**
Department : **Teknik Mesin FTI-ITS**
Academic Supervisor : **Arif Wahjudi, ST., MT., PhD.**

Abstrac

For many researches, balancing is one of important things in utilization on a bike. Stability on a body, velocity on a paddle bar and controlling direction on a handle bar are affecting balance on a bike. Bad balancing is one of the reasons accident involving bike. So, we need an automation mechanism taht could balance a bike while on static or dyanmics condition. The purpose in this final project is to control the gyroscope's spin velocity, so it can help stabilizing the balancing of a bike.

Balancing mechanism system on a self balancing bike is being created by using rotational effect on a gyroscope and using brushless DC as their actuator. For maintaining the balance, we control rotational speed of gyroscope using PID controller until the rotation is adjusting falling angle on bike. This mechanism utilitize error input in form in falling angle on a bike from the results on IMU MPU6050. This results is transferred to microcontroller an then we tunned the value of Kp and Ki for the purpose to get F as lowest possible. To get most relevant Kp and Ki value is by creating neural network system. Then, the value is inputting on microcontroller for controlling rotational speed on a gyroscope and maintain the balance of a bike.

The results from this project are designed control system can balance the bike. On PI-NN controlling could minimize average overshoot from 6,8178% to 6,567% (down 3,6789%), minimizing rise time from 1,2555 second to 1,2166 second (down 3,069 %) and steady state error from 0,246^{o} to 0,543^{o} (down 15,15%).

Key word: bike; balancing; gyroscope; neural network; proportional-integral-derivative control; microcontroller.

KATA PENGANTAR

Puji dan syukur marilah kita panjatkan kehadirat Tuhan Yang Maha Esa. Karena setiap hidayah dan karunia-Nya, penulis mampu menyelesaikan Tugas Akhir yang berjudul **Mekanisme Pengendalian Kecepatan *Spin* Giroskop pada Sistem *Self Balancing Bike***.

Tidak lupa ucapan terima kasih penulis sampaikan kepada berbagai pihak yang telah membantu dalam penyelesaian Tugas Akhir ini. Tanpa orang-orang tersebut, mustahil laporan tugas akhir ini dapat diselesaikan dengan baik. Adapun ucapan terima kasih penulis sampaikan kepada:

1. **Allah SWT** yang senantiasa memberi petunjuk, berkat, rahmat dan anugerahnya kepada penulis.
2. **Bapak Sri Baroto S dan Ibu Dewi PPS** selaku orang tua selalu mendoakan, mendidik, dan memberi semangat putra-putrinya untuk melakukan yang terbaik.
3. **Khaisar Muhammad H** selaku adik yang selalu memberi semangat dan menjadi tempat berbagi.
4. **Bapak Arif Wahjudi, S.T., M.T., Ph.D**., selaku dosen pembimbing Tugas Akhir yang selalu memberikan ilmu, kritik dan saran, serta bimbingannya selama proses penyelesaian Tugas Akhir ini.
5. **Bapak Dr. Unggul Wasiwitono, S.T., M.Eng.Sc** dan **Ibu Latifah Nurahmi, S.T., M.Sc., P.hD** selaku dosen penguji Tugas Akhir, yang telah menyempatkan berbagi ilmu, sehingga tugas akhir ini dapat terselesaikan dengan baik.
6. **Ibu Dr. Ir. Helena Caroline Kis Agustin, DEA** selaku dosen wali yang telah membimbing penulis selama masa studi di Teknik Mesin ITS.
7. **Mas Satrio** selaku sahabat dan partner yang selalu memberi dukungan baik secara fisik maupun psikis dalam pembuatan tugas akhir ini.

8. **Bunga Arsya Amanda** yang selalu menemani serta memberi dorongan motivasi, nasehat dan doa hingga tugas akhir ini selesai.
9. **Satrio Ramadan dan Yunico Rixy Setyawan** selaku partner dalam pembuatan Tugas Akhir ini.
10. **Alm. Haditya Zulkarnain, Ghinadia P.A., Teditya Nico L, Ramadhani Ayu S.N., Ayzam S., Amelia A.M** dan teman-teman laskar tamring yang telah mengisi hari-hari kuliah penulis dengan banyak pengalaman dan keseruan.
11. **Muhammad Arif, Deris Triana Noor, Tubagus Bima, Kholiq Deliasgarin, Febriana P.P, Rahadian Chandra, Mas'ud A, Hafizh N.P, Abi Nubli, Sulthoni Awan, Rosadila F, Ray Raditya** dan segenap keluarga besar Laboratorium Perancangan dan Pengembangan Produk yang selalu memberikan semangat dan motivasi kepada penulis.
12. Dosen dan karyawan Jurusan Teknik Mesin FTI-ITS yang telah memberikan ilmu, pengalaman, dan bantuannya selama masa studi.
13. Serta semua pihak yang secara langsung ataupun tidak langsung terlibat dalam proses penyelesaian tugas akhir ini.

Dengan selesainya Tugas Akhir ini, penulis berharap laporan ini dapat bermanfaat khususnya bagi penulis sendiri dan umumnya untuk kita semua.

Surabaya, 21 Juli 2018

Mario Muhammad Hafidh

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

[Halaman Ini Sengaja Dikosongkan]

BAB I
PENDAHULUAN

1.1 Latar Belakang

Sepeda kayuh roda dua merupakan salah satu alat transportasi yang paling populer di mata seluruh kalangan masyarakat dunia. Bahkan, masyarakat beberapa negara di dunia menggunakan sepeda kayuh sebagai kendaraan untuk mendukung mobilitas mereka dalam aktivitasnya sehari-hari, seperti berangkat ke kantor, sekolah ataupun sekedar berolahraga. Harganya yang terjangkau, beragam model desainnya yang mampu memenuhi kebutuhan berbagai macam lapisan masyarakat dan ramah lingkungan, membuat transportasi ini banyak dimiliki. Banyak institusi baik pemerintahan maupun swasta mulai mengembangkan konsep sepeda agar semakin nyaman digunakan oleh semua orang, salah satunya yaitu dengan sistem *self balancing bike.* Konsep *self balancing bike* mulai ramai dikembangkan mengingat faktor keseimbangan menjadi hal paling penting dalam mengendarai sepeda. Pengembangan konsep ini diharapkan mampu mengurangi angka kecelakaan dalam bersepeda khususnya akibat kehilangan keseimbangan pada saat mengendarainya.

Pada tahun 2017 telah dilakukan penelitian mengenai penerapan konsep *self balancing bike* pada *prototype* sepeda yang terbuat dari besi lubang dengan panjang 55 cm. Penerapan konsep *self balancing bike* ini menggunakan efek giroskop untuk menunjang kestabilan dari sistem keseimbangan sepeda. Pemanfaatan giroskop pada keseimbangan sepeda telah banyak dikembangkan serta menjadi produk untuk menyeimbangkan posisi sepeda dan membantu orang-orang mengurangi kecelakaan jatuh dalam mengendarai transportasi roda dua ini. Setelah perancangan penempatan giroskop sebagai sistem keseimbangan telah selesai dilakukan, maka dibutuhkan suatu mekanisme pengendalian untuk menjaga posisi sepeda agar tetap pada keadaan setimbang. Mekanisme pengendalian akan mengatur kecepatan motor DC *brushless* yang menggerakkan putaran spinning pada

giroskop menggunakan PID *controller* sehingga sepeda tetap terjaga keseimbangannya.

PID *controller* sejauh ini adalah *controller* yang paling banyak digunakan di industri. Dalam pengendalian PID, terdapat parameter-parameter yang harus ditentukan. Terdapat tiga parameter yaitu nilai *proportional gain* (Kp), *integral gain* (Ki), dan *derivative gain* (Kd). Bagian *proportional* menunjukkan nilai error, *integral* menunjukkan nilai rata-rata dari error, dan *derivative* adalah nilai prediksi dari error dengan pendekatan *linier extrapolation*. Pengaruh nilai Kp, Ki, dan Kd pada *transient response* adalah untuk mempercepat respon, mengurangi *steady-state error*, dan memperkecil *overshoot*. Jika sistem berperilaku dinamis, maka nilai dari parameter-parameter tersebut diperoleh dari persamaan differensial. Namun, terdapat beberapa kesulitan jika menggunakan persamaan differensial. Salah satunya adalah sistem yang cukup kompleks sehingga sulit untuk menemukan persamaan yang dapat mewakili sistem yang ingin dikendalikan. Ketika persamaan dari sistem tidak dapat mewakili sistem seceara keseluruhan, maka hasil yang diperoleh tidak dapat digunakan. Selain itu, butuh waktu yang cukup lama untuk menurunkan persamaan dari sistem.

Untuk mencari solusi dari permasalahan di atas adalah dengan menggunakan salah satu metode *intelligent* yaitu *neural network*. *Neural network* adalah gabungan koneksi-koneksi dari elemen proses (*synapse*), *units* atau *nodes* yang saling terhubung yang didasarkan pada fungsi *neuron* hewan. Kemampuan proses dari *network* terdapat pada nilai dari koneksi antar-unit (*weights*) yang diperoleh dari proses adaptasi, atau mempelajari pola dari suatu data input dan output.

Output dari proses adaptasi ini adalah untuk mendapatkan persamaan dari *proportional gain* yang digunakan pada sistem pengendali proporsional. Untuk mendapatkan persamaan *gain* yang sesuai dengan output yang diinginkan, maka dibutuhkan proses adaptasi dari *neural network*. Oleh karena itu, tugas akhir ini diajukan untuk mendapatkan nilai *proportional gain* (Kp) dan

integral gain (Ki) sebenarnya melalui *neural network* yang diinputkan ke dalam *controller* untuk mengatur kecepatan rotasi giroskop.

1.2 Rumusan Masalah

Berdasarkan latar belakang diatas, maka dapat dirumuskan permasalahan sebagai berikut:
1. Bagaimana hubungan *error* kemiringan sepeda dengan parameter pengendalian nilai Kp dan Ki agar menghasilkan putaran motor penggerak giroskop yang optimal?
2. Bagaimana sistem pengendali adaptif yang tepat untuk diterapkan pada mekanisme *self balancing bike*?

1.3 Tujuan Penelitian

Tujuan dari penelitian ini ialah sebagai berikut:
1. Mengetahui hubungan *error* kemiringan sepeda dengan parameter pengendalian nilai Kp dan Ki agar menghasilkan putaran motor penggerak giroskop yang optimal.
2. Mengetahui sistem pengendali adaptif yang tepat untuk diterapkan pada mekanisme *self balancing bike.*

1.4 Batasan Masalah

Agar tujuan penulisan dari tugas akhir ini lebih terarah dan sisematik, maka diperlukan adanya batasan masalah sebagai berikut:
1. Rancang bangun *self balancing bike* tidak dibahas.
2. Variabel yang dikontrol adalah kecepatan spin giroskop.
3. Spesifikasi motor DC Brushless memiliki kecepatan maksimal 7200 rpm.
4. Besar sudut kemiringan sepeda yang dikontrol maksimal 8^{o}.
5. Pembacaan sudut menggunakan sensor sudut MPU 6050.
6. Penelitian dilakukan pada saat posisi sepeda statis.

1.5 Manfaat Tugas Akhir

Tugas akhir ini diharapkan mampu memberikan manfaat diantaranya sebagai berikut:

1. Mengembangkan sistem pengendalian untuk meningkatkan kestabilan pada kendaraan roda dua.
2. Memperkaya dan memperdalam wacana dalam bidang ilmu mekatronika khususnya dalam *auto tunning controller*.
3. Diperoleh pengetahuan sistem kendali *auto tunning* yang diterapkan pada sistem *self balancing bike.*

BAB II
TINJAUAN PUSTAKA

2.1 Penelitian Terdahulu

Penelitian mengenai sistem pengendalian menggunakan giroskop pada mekanisme *self balancing bike* telah banyak dilakukan. Metode pengendalian yang digunakan pada setiap penelitian pun beragam jenisnya. Berikut ini adalah beberapa penelitian mengenai sistem pengendali yang diterapkan pada mekanisme *self balancing bike*. Penelitian yang dilakukan oleh San Kaplan dan Aaron Fineman (Kaplan, 2008) menjelaskan desain pengendalian PID controller pada *self balancing bike* menggunakan mikrokontroler PIC 32 produksi Microchip Technology Inc. *Controller* tersebut diberikan untuk meningkatkan respon dari sistem. Proporsional *control* berfungsi untuk memperkuat sinyal kesalahan (sinyal error), sehingga mempercepat keluaran sistem mencapai titik referensi. *Integral control* berfungsi untuk menurunkan *steady-state error*. Sedangakan *derivative control* berfungsi untuk mengurangi efek *overshoot*. Ketiga *control* tersebut digabung menjadi satu kesatuan.

Pada *control* PID, terdapat parameter yang berpengaruh terhadap fungsi *controller*, yaitu nilai Kp, Ki, dan Kd. Nilai tersebut didapat dari *tuning* dengan metode *root locus*. *Root locus* pada suatu sistem secara umum dapat diperbolehkan untuk memilih penguat (*gain*) yang ideal dengan spesifikasi respon *transient* yang diinginkan. Metode tersebut masih meng*gun*akan *tuning* manual dengan mencari nilai Kp, Ki, dan Kd secara perhitungan.

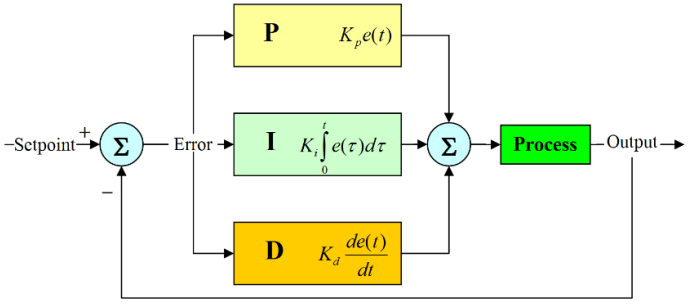
Metode pengendalian pada *self balancing bike* lainnya juga pernah dilakukan oleh Stephen C. Spry dan Anouck R. Girard (Spry, 2008) dengan menggunakan *controller* Atmel AVR Mega 128. Penelitian ini menggunakan dua metode pengaplikasian giroskop, yaitu menggunakan giroskop tunggal dan ganda. Pada penelitian ini, uji coba sistem pengendalian dibagi menjadi dua percobaan, yaitu pertama dengan membiarkan sudut giroskop

bergerak bebas natural dengan kecepatan rotasi maksimal dan kedua, mengontrol jatuhnya sudut giroskop.

2.2 Proportional Integral Derivatif (PID)

Di dalam suatu sistem kontrol kita mengenal adanya beberapa macam aksi kontrol, diantaranya yaitu aksi kontrol *proporsional*, aksi kontrol *integral* dan aksi kontrol *derivative.* Masing-masing aksi kontrol ini mempunyai keunggulan keunggulan tertentu, dimana aksi kontrol *proporsional* mempunyai keunggulan *rise time* yang cepat, aksi kontrol integral mempunyai keunggulan untuk memperkecil *error* ,dan aksi kontrol *derivative* mempunyai keunggulan untuk memperkecil *error* atau meredam *overshot/undershot*. Untuk itu agar kita dapat menghasilkan output dengan *risetime* yang cepat dan *error* yang kecil kita dapat menggabungkan ketiga aksi kontrol ini menjadi aksi kontrol PID.

Parameter pengontrol *Proporsional Integral derivative* (PID) selalu didasari atas tinjauan terhadap karakteristik yang di atur (*plant*). Dengan demikian bagaimanapun rumitnya suatu *plant*, perilaku *plant* tersebut harus di ketahui terlebih dahulu sebelum pencarian parameter PID itu dilakukan.

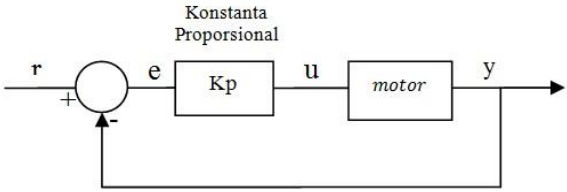


Gambar 2.1 Blok Diagram PID *Controller* (Xingjia, 2010)

2.3 Proportional Control

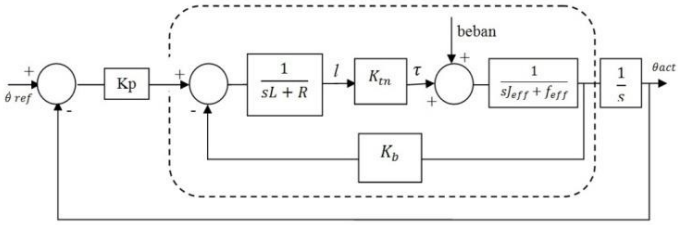
Control proporsional berfungsi untuk memperkuat sinyal kesalahan penggerak (sinyal error), sehingga akan mempercepat keluaran sistem mencapai titik referensi. Hubungan antara input *controller* r dengan sinyal error e terlihat pada persamaan berikut:

$$u = KP.e. \quad (2.1)$$



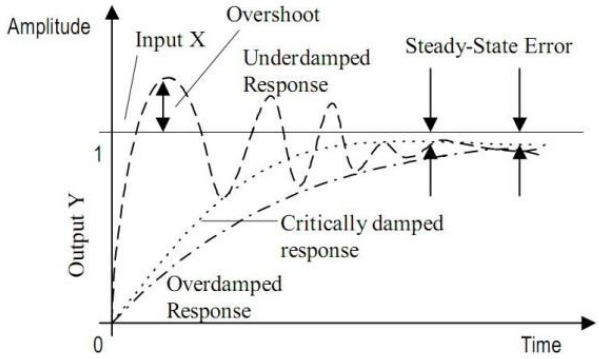
Gambar 2.2 diagram kontrol proporsional (Xingjia, 2010)

Gambar 2.2 menunjukkan blok diagram yang menggambarkan hubungan antara besaran setting, besaran aktual dengan besaran keluaran pengontrol proporsional. Sinyal keasalahan (*error*) merupakan selisih antara besaran setting dengan besaran aktualnya. Selisih ini akan mempengaruhi pengontrol, untuk mengeluarkan sinyal positif (mempercepat pencapaian harga setting) atau negatif (memperlambat tercapainya harga yang diinginkan).



Gambar 2.3 Diagram control proporsional Motor DC (Xingjia, 2010)

Untuk lebih jelasnya pada contoh kasus *control* kecepatan pada motor DC meng*gun*akan *control* P seperti pada gambar 2.3. Dalam uji simulasi ini, seluruh parameter motor dimasukan ke dalam blok sistem simulasi sesuai dengan diagram *control* pada gambar diatas. Kecepatan putar referensi yang digunakan adalah 2400 rpm. Skema *control* P di atas diuji dengan memberikan nilai K_P yang berbeda.



Gambar 2.4 Respon output *controller* P (Wang, 2001)

Nampak dalam gambar 2.4 bahwa semakin kecil KP maka *offset* atau *steady state error* semakin besar. Namun nilai KP yang terlalu besar akan menyebabkan osilasi pada saat start. *Control* P dapat di*gun*akan sendirian dalam aplikasi. Beberapa aplikasi seperti *control* temperatur pada *heater*, sistem penghemaan energi pada sistem *air conditioning* berdaasarkan *control* kecepatan motor kompresor sudah cukup memadai menggunakan *control* P saja.

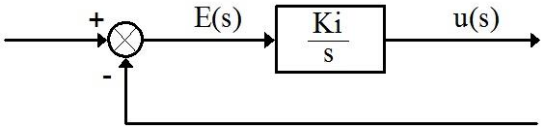
2.4 Integral Control

Kontroller *proporsional* tidak akan mampu menjamin output dari sistem akan menuju ke keadaan yang diinginkan kalau sebuah plant tidak memiliki unsur *integrator*. Fungsi dasar kontrol I adalah menurunkan *steady-state error*. Kontrol I jarang

digunakan sendirian dalam aplikasi. Biasanya selalu dikombinasikan dengan kontrol P untuk memperbaiki respon guna mencapai eror minimum. Pengontrol *integral* berfungsi menghasilkan respon sistem yang memiliki kesalahan keadaan stabil nol. Jika sebuah Plant tidak memiliki unsur integrator (1/s), pengontrol proposional tidak akan mampu menjamin keluaran sistem dengan kesalahan keadaan stabilnya nol. Dengan pengontrol *integral*, respon sistem dapat diperbaiki, yaitu mempunyai kesalahan keadaan stabilnya nol. Pengontrol *Integral* memiliki karaktiristik seperti halnya sebuah *integral*. Keluaran sangat dipengaruhi oleh perubahan yang

sebanding dengan nilai sinyal kesalahan. Keluaran pengontrol ini merupakan penjumlahan yang terus menerus dari perubahan masukannya.

Bila sinyal kesalahan tidak mengalami perubahan, keluaran akan menjaga keadaan seperti sebelum terjadinya perubahan masukan. Pada diagram blok kontroller integral, menunjukan hubungan antara nilai *error* dengan *output*. Kontroler *integral* membantu menaikan respon sehingga menghasilkan output yang diinginkan.

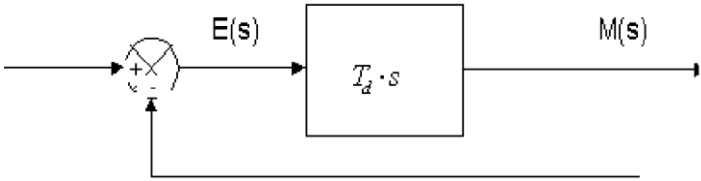


Gambar 2.5 Diagram blok kontrol *integral*
(Xingjia, 2010)

2.5 Derivative Control

Keluaran pengontrol *Derivative* memiliki sifat seperti halnya suatu operasi differensial. Perubahan yang mendadak pada masukan pengontrol, akan mengakibatkan perubahan yang sangat besar dan cepat. Gambar 2.6 menunjukkan blok diagram yang

menggambarkan hubungan antara sinyal kesalahan dengan keluaran pengontrol.

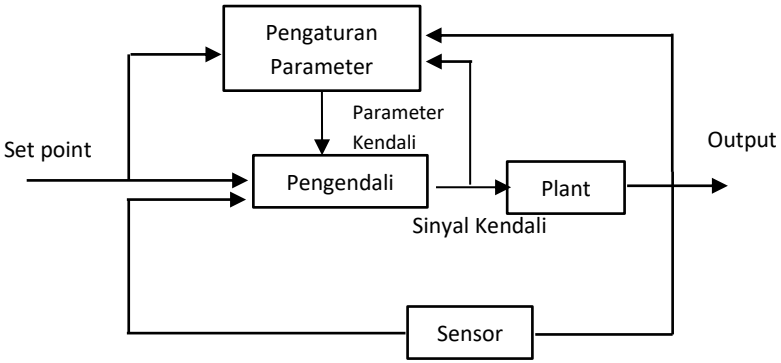


Gambar 2.6 Blok diagram pengontrol *derivative* (Xingjia, 2010)

Ketika masukannya tidak mengalami perubahan, keluaran pengontrol juga tidak akan mengalami perubahan. Dengan sifat ini ia dapat digunakan untuk memperbaiki *transent response* dengan memprediksi error yang akan terjadi. Kontrol *derivative* hanya berubah saat ada perubahan error sehingga saat error statis kontrol ini tidak akan bereaksi, hal ini pula yang menyebabkan kontroler *derivative* tidak dapat dipakai sendiri.

2.6 Teknik Pengendalian Adaptif

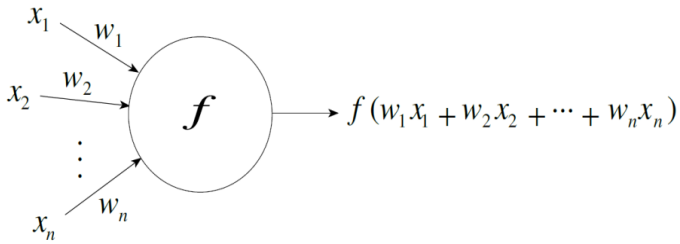
Teknik kendali adaptif merupakan teknik kendali yang dilengkapi dengan algoritma pembelajaran. Teknik kendali adaptif didefinisikan juga sebagai sistem kendali dimana paramater-parameternya dapat diatur dan juga memiliki mekanisme untuk mengatur parameter-parameter tersebut. Skema dari sistem kendali adaptif terdiri dari dua bagian. Bagian yang pertama adalah bagian umpan balik biasa dengan pengendali dan *plant*, sedang bagian kedua adalah bagian pengaturan parameter. Blok diagram dari sistem kendali adaptif ditunjukkan pada Gambar 2.7.



Gambar 2.7 Diagram sistem kendali adaptif (Umam, 2014)

2.7 Neural Network

Neural network adalah salah satu model komputasi yang terdiri dari *neuron-neuron* buatan untuk memproses informasi dari input ke output. *Neuron-neuron* terdiri dari *synapses* dan *nodes*. Kumpulan dari node-node disebut *layer*. *Layer* dapat dibuat lebih dari satu. *Node* digunakan untuk menghimpun fungsi-fungsi.



Gambar 2.8 *Node* dari *neuron* (Gurney, 1997)

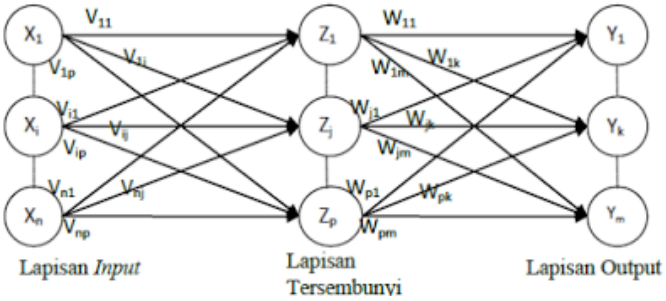
Gambar 2.8 menunjukkan struktur dari *neuron* dengan n input. Setiap *synapse* dari input dapat mentransmisikan nilai real x.

Fungsi f dibangun secara acak didalam *neuron*. Biasanya *synapse* dari input mempunyai nilai *weight*, yang berarti nilai pada input (X_i) dikalikan dengan *weight* (w_i) pada *synapse* tersebut. Nilai-nilai yang ditransmisikan pada *neuron* kemudian dibuat sebuah fungsi f. Ada tiga faktor yang perlu diperhatikan dalam membuat model *neural network*, yaitu:

1. Struktur dari *node*
2. Topologi dari *network*
3. *Learning algorithm* yang digunakan untuk mendapatkan nilai *weight*.

Salah satu penggunaan *neural network* adalah untuk menentukan sebuah fungsi yang mendekati fungsi aslinya. Fungsi asli tersebut *bias*anya tidak diketahui karena sistem terlalu rumit untuk diturunkan ke dalam sebuah persamaan. Penggunaan *neural network* dimaksudkan untuk memperoleh persamaan *gain* dari sistem *control*. *Gain* tersebut akan mempengaruhi sistem sehingga kecepatan respon dari sistem akan meningkat.

Di bawah ini adalah contoh dari *neural network* dengan tiga input dan satu output:



Gambar 2.9 Struktur dari *neural network* (Sujanarko, 2012)

Gambar 2.9 menunjukkan pada *neural network* terdapat tiga *layer*, yaitu *layer* input, *hidden layer* dan *layer* output. Jika sebagai input didefinisikan sebagai $X = [x_1, x_2, \dots, x_n]^T$ dan H =

$[h_1, h_2, \dots, h_m]^T$, maka h_j adalah fungsi Gaussian yang didefinisikan sebagai berikut:

$$h_j = \exp\left(-\frac{\|X - C_j\|^2}{2b_j^2}\right), \text{j} = 1,2, \dots \text{m} \qquad (2.2)$$

dimana $C_j = [c_{j1}, c_{j2}, \dots, c_{ji} \dots, c_{jm}]^T$, B $= [b_1, b_2, \dots, b_m]^T$, dan *weight* dari *network* adalah

W $= [w_1, w_2, \dots, w_m]^T$, maka output dari *network* tersebut adalah:

$$y_m(k) = w_1 h_1 + w_2 h_2 + \dots + w_m h_m \qquad (2.3)$$

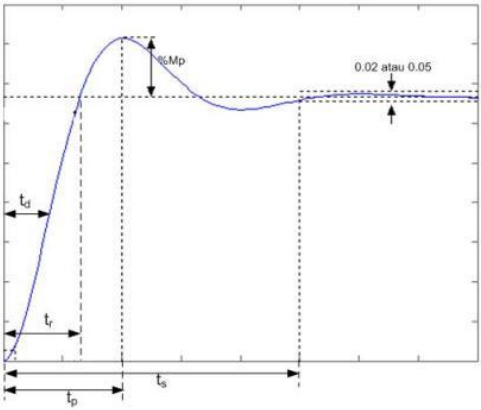
2.8 Parameter Response Kontrol PI

Ada beberapa parameter dalam menentukan suatu sistem *close loop*, yaitu *rise time*, *overshoot*, *settling time*, dan *steady state error*. *Rise time* adalah waktu yang dibutuhkan oleh *output plant* yang melebihi 90% dari tingkat yang diinginkan saat pertama kali sisem dijalankan. *Overshoot* adalah seberapa besar peak level lebih tinggi dari *steady state*, untuk membuat normal lagi *steady state*. *Settling time* adalah waktu yang dibutuhkan sistem untuk meng-konvergenkan *steady state*. *Steady state error* adalah perbedaan antara *steady state* ouput dengan output yang diinginkan.

Tabel 2.1 Parameter respons kontrol PI (Syawal, 2015)

Respon Lup Tertutup	*Rise Time*	*Overshoot*	*Settling Time*	*Steady-State Error*
Proporsional	Menurunkan	Meningkatkan	Perubahan kecil	Menurunkan/ mengurangi
Integral	Menurunkan	Meningkatkan	Meningkatkan	Mengeliminasi
Derivatif	Perubahan Kecil	Menurunkan	Menurunkan	Perubahan kecil

Kp berguna untuk mengurangi *rise time*. Sedangkan Ki berguna untuk menghapuskan *steady state error*.



Gambar 2.10 Response system (Pitowarno, 2006)

2.9 Indeks Performa (F)

Sebelum PI-NN dapat digunakan untuk memodelkan sistem kendali adaptif sebagai parameter jaringan cerdas, maka proses pelatihan atau pembelajaran sangatlah diperlukan.

Pengambilan data input dan output diperlukan karena metode *neural network* dibutuhkan pelatihan untuk mendapatkan fungsi dari *gain*. Data input dan output diperoleh dari pengambilan data pada sistem kontroler yang dibuat dan telah dipasang sistem pengendali. Untuk mendapatkan nilai K_P dan K_I optimum dibutuhkan persamaan yang menyatakan performa dari respon tersebut. Adapun persamaan tersebut adalah *performance index* (F) yang diperoleh dari persamaan:

$$F = (k_1 \times \%Overshoot + k_2 \times RiseTime + k_3 \times steady\ state\ error)$$ (Wang, 2001) (2.3)

2.10 Arduino Uno

Arduino Uno adalah arduino board yang menggunakan mikrokontroler ATmega328. Arduino Uno memiliki 14 pin digital (6 pin dapat digunakan sebagai output PWM), 6 input analog, sebuah 16 MHz osilator kristal, sebuah koneksi USB, sebuah konektor sumber tegangan, sebuah header ICSP, dan sebuah tombol reset. Arduino Uno memuat segala hal yang dibutuhkan untuk mendukung sebuah mikrokontroler. Hanya dengan menhubungkannya ke sebuah komputer melalui USB atau memberikan tegangan DC dari baterai atau adaptor AC ke DC sudah dapat membuanya bekerja. Arduino Uno menggunakan ATmega16U2 yang diprogram sebagai USB-to-serial converter untuk komunikasi serial ke computer melalui port USB. Tampak atas dari arduino uno dapat dilihat pada gambar 2.11.

Gambar 2.11 Papan Arduino Uno (Syawal, 2015)

Uno berbeda dari semua board mikrokontroler diawal-awal yang tidak menggunakan chip khusus driver FTDI USB-to-serial. Sebagai gantinya penerapan USB-to-serial adalah ATmega16U2 versi R2 (versi sebelumnya ATmega8U2). Versi Arduino Uno Rev.2 dilengkapi resistor ke garis ground.

Tabel 2.2 Spesifikasi Arduino Uno (Syawal, 2015)

Mikrokontroller	Atmega328
Operasi *Voltage*	5V
Input *Voltage*	7-12 V (Rekomendasi)
Input *Voltage*	6-20 V (limits)
I/O	14 pin (6 pin untuk PWM)
Arus	50 mA
Flash Memory	32KB
Bootloader	SRAM 2 KB
EEPROM	1 KB
Kecepatan	16 Mhz

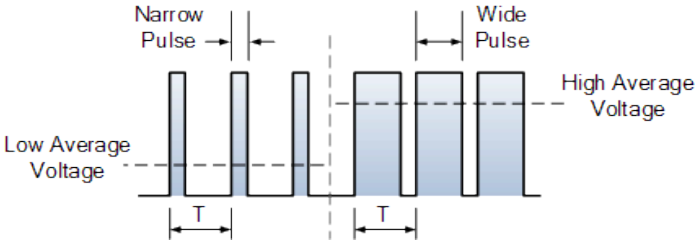
2.11 Pulse Width Modulation (PWM)

Pulse Width Modulation (PWM) secara umum adalah sebuah cara memanipulasi lebar sinyal yang dinyatakan dengan pulsa dalam suatu perioda, untuk mendapatkan tegangan rata-rata yang berbeda. Beberapa contoh aplikasi PWM adalah pemodulasian data untuk telekomunikasi, pengontrolan daya atau tegangan yang masuk ke beban, regulator tegangan, audio effect dan penguatan, serta aplikasi-aplikasi lainnya.

Aplikasi PWM berbasis mikrokontroler biasanya berupa, pengendalian kecepatan motor DC, pengendalian motor servo dan pengaturan nyala terang LED.

2.11.1 Prinsip Kerja PWM

Sinyal PWM pada umumnya memiliki amplitudo dan frekuensi dasar yang tetap, namun memiliki lebar pulsa yang bervariasi. Lebar pulsa PWM berbanding lurus dengan amplitudo sinyal asli yang belum termodulasi. Artinya, sinyal PWM memiliki frekuensi gelombang yang tetap namun *duty cycle* bervariasi (antara 0% hingga 100%).



Gambar 2.12 Sinyal PWM (Prayogo, 2012)

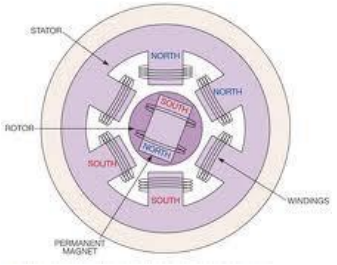
Pulse Width Modulation (PWM) merupakan salah satu teknik untuk mendapatkan signal analog dari sebuah piranti digital. Sebenarnya Sinyal PWM dapat dibangkitkan dengan banyak cara, dapat menggunakan metode analog dengan menggunakan rangkaian op-amp atau dengan menggunakan metode digital. Metode analog dapat membuat setiap perubahan PWM-nya dengan sangat halus, sedangkan menggunakan metode digital setiap perubahan PWM dipengaruhi oleh resolusi dari PWM itu sendiri. Resolusi adalah jumlah variasi perubahan nilai dalam PWM tersebut. Misalkan suatu PWM memiliki resolusi 8 bit berarti PWM ini memiliki variasi perubahan nilai sebanyak 28 = 256 variasi mulai dari 0 – 255 perubahan nilai yang mewakili duty cycle 0 – 100% dari keluaran PWM tersebut.

2.12 Motor DC Brushless (BLDCM)

BLDC motor atau dapat disebut juga dengan BLAC motor merupakan motor listrik *synchronous* AC 3 fasa. Perbedaan pemberian nama ini terjadi karena BLDCM memiliki BEMF berbentuk *trapezoid*, sedangkan BLACM memiliki BEMF berbentuk sinusoidal. Walaupun demikian keduanya memiliki struktur yang sama dan dapat dikendalikan dengan metode six-step maupun PWM sinusoidal. Dibandingkan dengan motor DC, BLDCM memiliki biaya perawatan yang lebih rendah dan kecepatan yang lebih tinggi akibat tidak digunakannya *brush*. Dibandingkan dengan motor induksi, BLCM memiliki efisiensi yang lebih tinggi karena rotor dan torsi awal yang lebih tinggi

karena rotor terbuat dari magnet permanen. Walaupun memiliki kelebihan dibandingkan dengan motor DC dan motor induksi, pengendalian BLDCM jauh lebih rumit untuk kecepatan dan torsi yang konstan karena tidak adanya *brush* yang menunjang proses komutasi dan harga BLDCM jauh lebih mahal.

Secara umum BLDCM terdiri dari dua bagian, yakni rotor, bagian yang bergerak, yang terbuat dari permanen magnet dan stator, bagian yang tidak bergerak, yang terbuat dari kumparan 3 fasa. Walaupun merupakan motor listrik *synchronous* AC 3 fasa, motor ini tetap disebut dengan BLDCM karena pada implementasinya BLDCM menggunakan sumber DC sebagai sumber energi utama yang kemudian diubah menjadi tegangan AC dengan menggunakan inverter 3 fasa. Tujuan dari pemberian tegangan AC 3 fasa pada stator BLDCM adalah menciptakan medan magnet putar stator untuk menarik magnet rotor.



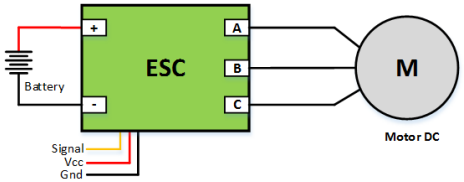
Gambar 2.13 Penampang Motor BLDCM (Ismeal, 2012)

Oleh karena tidak adanya *brush* pada motor BLDC, untuk menentukan *timing* komutasi yang tepat pada motor ini sehingga didapatkan torsi dan kecepatan yang konstan, diperlukan 3 buah sensor *Hall* dan atau *encoder*. Pada sensor *Hall*, *timing* komutasi ditentukan dengan cara mendeteksi medan magnet rotor dengan menggunakan 3 buah sensor *Hall* untuk mendapatkan 6 kombinasi *timing* yang berbeda, sedangkan pada *encoder*, *timing* komutasi ditentukan dengan cara menghitung jumlah pola yang ada pada *encoder*.

2.13 Electronic Speed Control (ESC)

Electronic Speed Control (ESC) adalah rangkaian elektronik yang berfungsi sebagai pengatur kecepatan putaran motor pada pesawat RC atau helikopter RC, cara kerjanya yaitu dengan cara menterjemahkan sinyal yang diterima receiver dari transmitter. Di pasaran terdapat berbagai merk ESC dengan kekuatan arus (*current rating*) dan kekuatan voltase (*voltage rating*) serta fitur yang ditawarkan. Pada dasarnya, ESC merupakan sebuah perangkat yang mengontrol motor melalui sebuah sistem radio.

Penggunaan ESC tipe motor *Brushless* untuk segala jenis motor *RimFire* lebih dianjurkan dibandingkan tipe motor *Brushed*, sebab ESC tipe *Brushless* mampu bekerja hingga motor *Rimfire* berkapasitas 28mm. Hal tersebut dapat menghindarkan kerusakan baik pada motor maupun ESC. Seperti halnya konfigurasi H-bridge 4 buah transistor atau IC L298 yang digunakan sebagai driver motor DCMP dalam pembuatan robot jenis lain (misalnya robot line follower), ESC juga memiliki fungsi sebagai driver motor DCBL dalam pembuatan multirotor.



Gambar 2.14 Blok diagram ESC (Tefay, 2011)

ESC pada motor BLDC mengontrol putaran pada motor dengan cara menyeleksi energi pada setiap fasenya. Saat arus listrik dikendalikan melalui gulungan stator, muatan listrik yang bergerak dengan medan magnet lalu menghasilkan gaya pada rotor yang menyebabkan motor dapat berputar. Kontroler dapat mengoperasikan motor BLDC selayaknya motor stepper tanpa membutuhkan *feedback*. Ketika posisi rotor dapat diketahui,

kontroler dapat menghitung jumlah gulungan yang optimal untuk menghasilkan energi pada setiap titik secara berkala sehingga mampu mencapai kecepatan putaran dan nilai torsi yang diinginkan. Namun, kebanyakan jenis motor BLDC yang tersedia di pasaran tidak memiliki kemampuan secara langsung untuk mendeteksi posisi rotor.

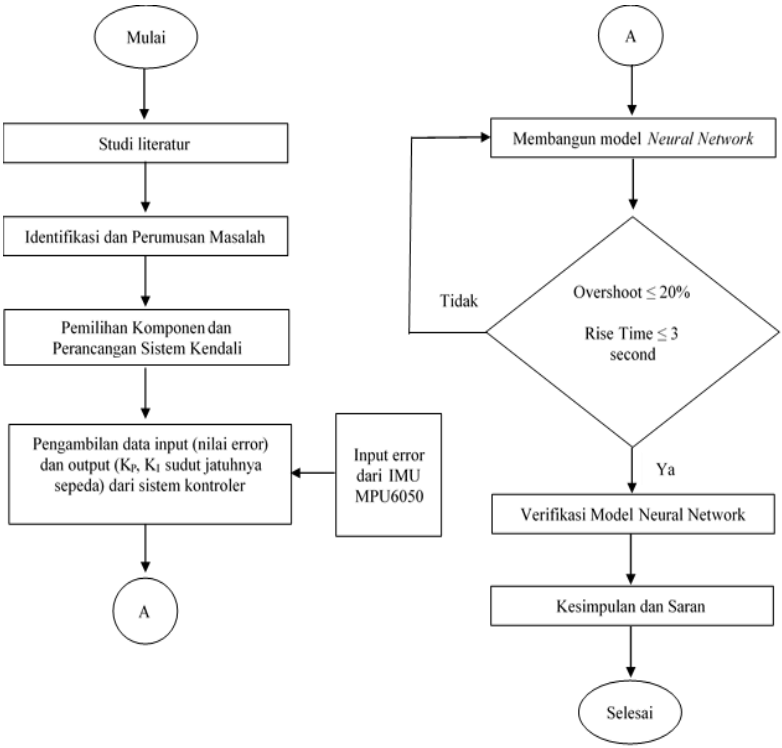
Gambar 2.15 Komponen ESC (Tefay, 2011)

Pada umumnya, satu unit ESC digunakan untuk men-drive satu unit aktuator (dalam hal ini adalah motor DCBL). Namun sekarang telah tersedia jenis ESC yang secara fisik hanya satu unit namun dapat digunakan untuk men-drive 4 buah aktuator. Seperti ESC yang didesain khusus untuk quadcopter merk Qbrain. ESC Qbrain dapat digunakan sebagai driver 4 buah motor DCBL dengan arus kerja masing-masing sebesar 20A

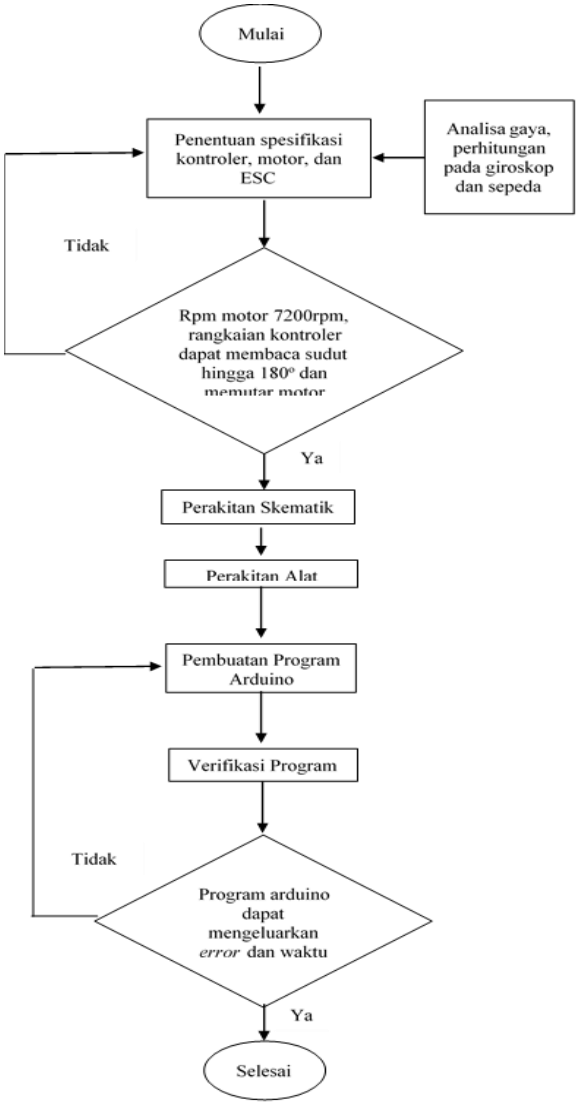
BAB III
METODOLOGI PENELITIAN

Metodologi penelitian ini menggambarkan langkah-langkah yang akan dilakukan dalam penelitian. Secara umum langkah-langkah yang akan dilakukan dalam penelitian ini ditunjukkan dalam diagram alir pada gambar 3.1, gambar 3.2 dan gambar 3.3.

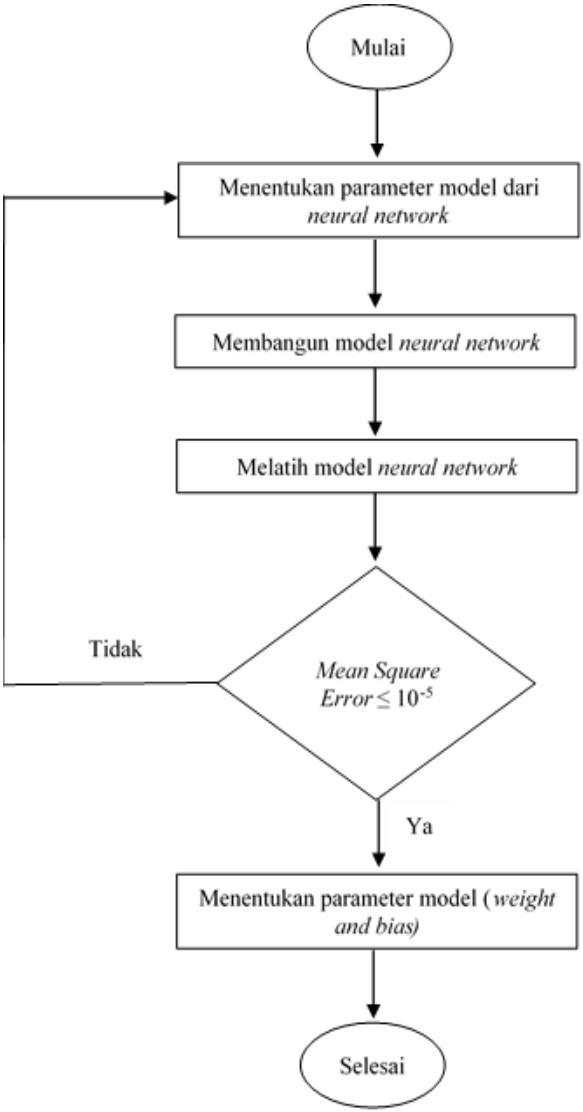
3.1 Diagram Alir Penelitian



Gambar 3.1 Diagram Alir Penelitian



Gambar 3.2 Diagram alir untuk pemilihan komponen dan perancangan sistem kendali



Gambar 3.3 Diagram alir untuk membangun model *Neural Network*

3.2 Langkah-langkah Penelitian

Langkah-langkah penelitian secara detail pada laporan tugas akhir ini adalah sebagai berikut :

1. Studi literatur

Studi literatur dilakukan sebagai tahap awal dan dasar dalam memulai proposal tugas akhir. Pada tahap ini juga dilakukan pengumpulan data terkait dengan perancangan sistem pengendali.

2. Perumusan Masalah

Langkah ini adalah menentukan permasalahan yang diangkat pada penelitian ini untuk dicari solusinya. Rumusan masalah yang diangkat pada penelitian ini adalah bagaimana menerapkan sistem pengendali pada sistem *self balancing bike*, bagaimana mendapatkan nilai K_P dan K_I melalui *neural network* pada sistem pengendali, bagaimana mengatur kecepatan serta meningkatkan respon dari sistem *control* untuk mengendalikan keseimbangan sepeda.

3. Pemilihan Komponen

Pada tahap ini dilakukan pemilihan komponen-komponen yang sesuai terkait dengan perancangan sistem kendali otomatis. Informasi lengkap mengenai perangkat keras penelitian tersebut adalah sebagai berikut:

a) Laptop

Laptop dengan spesifikasi memory 4GB RAM, intel® Core ™ i5. Laptop digunakan untuk mengolah data Kp dan Ki melalui *Neural Network* dengan *software* Matlab.

b) Arduino Uno

Arduino merupakan sebuah pengendali mikro single-board yang bersifat open-source, dirancang untuk memudahkan penggunaan elektronik dalam berbagai bidang. Pengendalian kecepatan putar motor DC menggunakan Pulse Width Modulation (PWM) juga terdapat sistem kontrol pada Arduino.

c) Motor Driver L298N.

Motor driver merupakan sebuah perangkat yang dapat mengendalikan arah putaran dan kecepatan motor DC.

d) Motor DC merupakan jenis motor listrik yang bekerja menggunakan sumber tegangan DC. Motor DC menggunakan arus langsung, digunakan pada penggunaan khusus dimana diperlukan penyalaan torsi yang tinggi atau percepatan yang tetap.

4. Perancangan Sistem Kendali

Pada tahap ini dilakukan perancangan logika berpikir yang ada pada software Arduino, difokuskan pada pengontrolan kecepatan motor DC yang memutar komponen *gyroscope* sesuai besar jatuhnya sudut sepeda.

5. Perakitan Perangkat Keras Kontroler

Pada perakitan perangkat keras kontroler ini dilakukan penyusunan seluruh komponen yang telah dipilih, kemudian mengkomunikasikan motor driver dengan Arduino. Mengkomunikasikan motor driver dengan Arduino merupakan tahapan untuk menentukan besar PWM sehingga kecepatan putar *gyroscope* motor DC dapat diatur. Selanjutnya, mengkomunikasikan arduino yang mendapat input error dari *image processing* dengan motor DC merupakan tahap berikutnya yang harus dilakukan untuk mengatur kecepatan putar *gyroscope* agar sesuai dengan besar sudut jatuhnya sepeda.

6. Pengambilan data input (error dari *image processing*) dan output (Kp dan Ki) dari sistem kontroler.

Pengambilan data input dan output diperlukan karena metode *neural network* dibutuhkan pelatihan untuk mendapatkan fungsi dari *gain*. Data input dan output diperoleh dari pengambilan data pada sistem kontroler yang dibuat dan telah dipasang sistem pengendali. Untuk mendapatkan nilai K_P dan K_I optimum dibutuhkan persamaan yang menyatakan performa dari respon tersebut. Adapun persamaan tersebut adalah *performance index* (F) yang diperoleh dari persamaan:

$$F = (k_1 \times \%Overshoot + k_2 \times RiseTime + k_3 \times steady\ state\ error) \quad (2.3)$$

Dimana:

$k_1 = 50$
$k_2 = 10$
$k_3 = 50$

Performce index tersebut tidak mutlak seperti persamaan 3.1, tetapi dapat diubah seperti yang diinginkan. Pada persamaan diatas adalah menitikberatkan pada *%overshoot* dan *steady-state error*, sehingga performa yang meningkat adalah *%overshoot* dan *steady-state error*. Namun, jika performa respon pada *rise time* yang ingin ditingkatkan, maka nilai k_2 dapat dinaikkan dan atau nilai k_1 dan k_3 diturunkan sehingga nilai k_2 melebihi parameter yang lain. Tetapi jika ingin meningkatkan performa respon pada *settling time*, maka settling time ditambahkan pada persamaan diatas dan diberi konstanta pengali.

Pengambilan data dilakukan dengan menyusun kemungkinan error yang dihasilkan dari proses *image processing*. Untuk sudut jatuhnya sepeda terdapat error maksimal yaitu 60. Sehingga, untuk sudut diberikan rentang nilai untuk error dari 0 sampai 60. Kemudian nilai error tersebut dibagi menjadi beberapa nilai, yaitu dibagi menjadi 12 bagian, sehingga penambahan nilai error dari 0 sampai 60 adalah 5.

Masing-masing nilai error tersebut disimulasikan dengan memberikan nilai K_P dan K_I pada masing-masing sudut. Nilai K_P ditentukan pada rentang nilai tiga sampai sepuluh dan nilai K_I ditentukan pada rentang nilai nol sampai tiga. Karakteristik respon yang diambil pada setiap simulasi adalah *%overshoot*, *rise time*, dan *steady-state error*. Kemudian nilai-nilai tersebut dihitung dengan menggunakan persamaan 3.1 sehingga diperoleh data *performance index* (F). Data dari nilai F tersebut kemudian diurutkan pada masing-masing nilai error. Masing-masing nilai error yang mempunyai nilai F paling kecil kemudian diambil untuk dijadikan data input dari pelatihan *neural network*.

Output yang digunakan adalah nilai K_P dan K_I. Sama seperti pengambilan data input, data output juga diperoleh dari nilai K_P dan K_I pada masing-masing nilai error yang mempunyai nilai F paling kecil.

1. Membangun Model *Neural Network*

Dalam hal membangun model *neural network* terdapat langkah-langkah yang harus dilakukan yaitu sebagai berikut:

a. Menentukan parameter model dari *neural network*

Pada tahap ini, model *neural network* dibangun dengan menentukan parameter-parameter dari model *neural network*. Penentuan berapa layer dan neuron yang akan dibuat menjadi hal yang utama pada langkah ini. Parameter tersebut mempengaruhi output yang dihasilkan oleh *neural network* apakah sesuai dengan yang diinginkan atau tidak.

b. Membangun model *Neural network*

Pada tahap ini, model neural network dibangun dengan memasukkan parameter-parameter dari model *neural network*, yaitu menentukan jumlah layer dan neuron yang digunakan.

c. Melatih model *neural network*

Pelatihan model *neural network* dimaksudkan untuk mendapatkan nilai K_P dan K_I. Pada langkah ini, data input dan output dimasukkan kedalam *neural network* untuk diproses. Input untuk dimasukkan ke dalam pelatihan *neural network* diperoleh dari data error masing-masing sudut dari sistem yang sudah ada. Sementara output yang dimasukkan ke dalam pelatihan *neural network* adalah nilai K_P dan K_I sudut jatuhnya sepeda .

d. Mendapatkan parameter *weight* dan *bias*

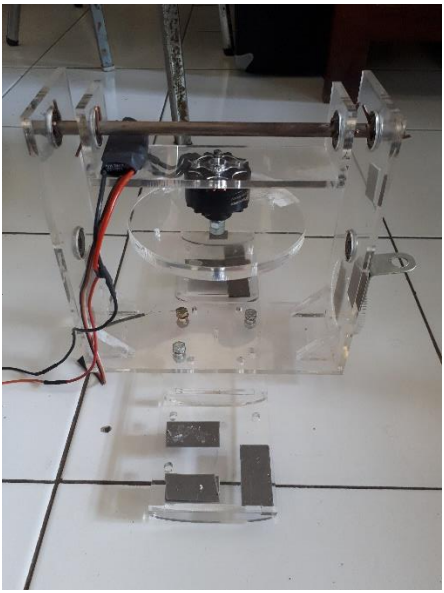
Output yang dihasilkan dari pelatihan *neural network* adalah *weight* dan *bias*. Kedua parameter tersebut adalah parameter yang mempengaruhi persamaan dari *neural network* untuk menghasilkan nilai K_P dan K_I.

2. Verifikasi Model

Untuk menyempurnakan model, diperlukan verifikasi untuk mengetahui apakah model tersebut bisa digunakan. Verifikasi tersebut dilakukan dengan pengujian pada sistem kontrol sepeda apakah output yang dikeluarkan sudah sama dengan set point.

BAB IV
PERANCANGAN SISTEM PENGENDALIAN *PROTOTYPE* SEPEDA

4.1 Rancangan dan Penentuan Spesifikasi Alat



Gambar 4.1 Desain alat

Penelitian ini menggunakan giroskop tunggal yang diaplikasikan pada *prototype* sepeda yang arah putaran gimbalnya berputar berlawanan dengan arah putaran jarum jam (*counter-clockwise*) dengan tujuan untuk memperbaiki respon keseimbangan sepeda. Dengan $\dot{\alpha}$ adalah kecepatan sudut giroskop, I momen inersia giroskop dan ω adalah kecepatan putar giroskop, maka besarnya torsi giroskopik total merupakan resultan torsi giroskopik dari gimbal.

Hampir keseluruhan alat ini dirancang menggunakan material akrilik dengan massa jenis sebesar 2,04 kg/m^3, dengan ketebalan 5 mm dan 10 mm pada bagian bodi serta 10 mm pada bagian giroskop. Pada bagian atas sebagai tempat untuk menggantungkan giroskop agar arah putaran presesinya dapat bergerak bebas ke depan dan belakang, maka komponen yang digunakan adalah poros yang terbuat dari besi dengan panjang total 20 cm serta memiliki diameter berukuran 8mm. Selain itu, agar poros bisa berputar dengan baik, maka dilengkapi dengan komponen *roller-bearing* dengan diameter dalam 8mm dan diameter luar 22,5mm. *Prototype* ini memiliki massa total sebesar 972 gram, dengan panjang total 30 cm dan tinggi 22 cm. Untuk bagian gimbal memiliki massa total sebesar 128 gram dan diameter 12 cm.

Untuk bisa memperbaiki respon keseimbangan *prototype* sepeda, maka dibutuhkan perhitungan gaya untuk mengetahui seberapa besar torsi yang dihasilkan oleh giroskop untuk menahan beban torsi yang dihasilkan oleh alat. Besarnya torsi giroskopik terhadap variasi kecepatan sudut *flywheel* ditunjukkan pada tabel 4.1, sedangkan perhitungan torsi yang dihasilkan oleh alat ditunjukkan pada tabel 4.2.

Tabel 4.1 Torsi giroskopik terhadap variasi kecepatan *spin flywheel*

Inersia (kg.m^2)	rpm spin	spin (rad/s)	rpm presesi	presesi (rad/s)	Torsi
0.000234	1000	104.76	9	0.94	0.02
0.000234	2000	209.52	9	0.94	0.05
0.000234	3000	314.29	9	0.94	0.07
0.000234	4000	419.05	9	0.94	0.09
0.000234	5000	523.81	9	0.94	0.12
0.000234	6000	628.57	9	0.94	0.14
0.000234	7000	733.33	9	0.94	0.16

Tabel 4.2 Torsi alat terhadap variasi sudut kemiringan

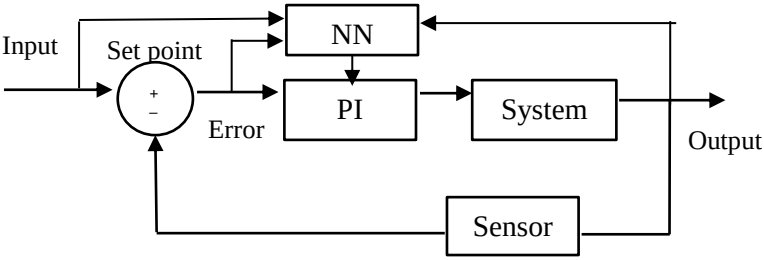
Θ (deg)	sin Θ	Massa (Kg)	g (m/s^2)	h (m)	Torsi Bodi
1	0.01945	0.972	9.81	0.138	0.025594
2	0.03489	0.972	9.81	0.138	0.045911
3	0.05233	0.972	9.81	0.138	0.06886
4	0.06975	0.972	9.81	0.138	0.091782
5	0.08715	0.972	9.81	0.138	0.114678
6	0.10452	0.972	9.81	0.138	0.137535
7	0.12186	0.972	9.81	0.138	0.160352
8	0.1391	0.972	9.81	0.138	0.183038

Variabel tetap yang nilainya tidak dapat divariasikan diantaranya adalah massa, gaya gravitasi, dan momen inersia dari giroskop. Sedangkan kecepatan *spin* giroskop dapat divariasikan dan dikontrol dengan nilai maksimal pada 7200 rpm. Untuk kecepatan putar dan sudut pada presesi giroskop tidak dilakukan pengendalian atau dengan kata lain dibiarkan bergerak secara natural berbanding lurus dengan besarnya kecepatan *spin*.

Pada penelitian ini ditentukan untuk kecepatan putar giroskop sebesar minimal 2075 rpm dan maksimal 7200 rpm. Secara teori, berdasarkan perhitungan yang ditunjukkan pada tabel 4.1, besar torsi giroskopik maksimal yang tercipta adalah sebesar 0,16 Nm. Torsi tersebut sangat mencukupi untuk melawan torsi bodi pendulum pada sudut 8^o, dengan catatan masukan nilai PWM (*Pulse Width Modulation*) pada *controller* memberikan keluaran rpm yang sesuai. Dalam penentuan spesifikasi motor *brushless* DC, motor *brushless* DC diharuskan dapat memutar giroskop seberat 128 gram dengan momen inersia sebesar 0.000234 $kg.m^2$. Dari hasil percobaan, giroskop dapat berputar dengan baik menggunakan motor berdiameter 41 mm. Spesifikasi motor *brushless* yang digunakan adalah 700kv, diameter dan tebal 41-20, panjang ass 20 mm serta menggunakan *electronic speed controller* dengan spesifikasi arus maksimum 80A dan tegangan 11,2V.

4.2 Blok Diagram Pengendalian

Pengambilan data dilakukan dengan metode tunning pada penyusunan komponen hardware yang telah dibuat pada sistem *self balancing bike* sebelumnya. Blok diagram kontrol yang digunakan pada proses pengambilan data adalah sebagai berikut :

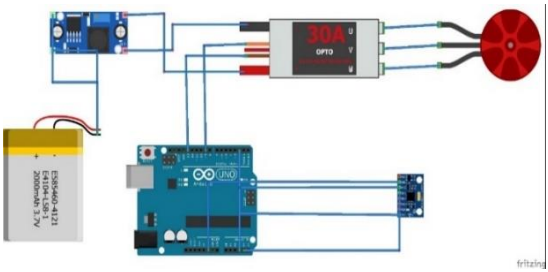


Gambar 4.2 Blok diagram pengendalian giroskop pada *self balancing bike*

Pada penelitian ini digunakan sistem kendali *Proportional Integral – Neural Network* untuk menyeimbangkan sepeda yakni menyeimbangkan sudut θ pada *set point* kemiringan 0^o. Sistem mendapatkan *input* berupa sudut kemiringan jatuhnya sepeda, dan output kecepatan *spin* giroskop. Aktuator pada penelitian ini berupa motor DC *brushless* yang akan dilakukan sistem identifikasi pada sub bab berikutnya, sedangkan *feedback* yang diberikan berupa sudut kemiringan θ yang diukur oleh sensor MPU 6050 dan kecepatan (rpm) motor DC *brushless* yang diukur dengan alat ukur *tachometer*.

Blok diagram pengendalian yang disusun sesuai gambar 4.2 menunjukkan bahwa sistem yang digunakan pada penelitian ini adalah sistem pengendalian tertutup (*close loop system controller*). *Input* didapatkan dari gangguan (*disturbance*) yang diberikan kepada *prototype* sepeda, sehingga menghasilkan variasi perubahan kemiringan sudut dari keadaan seimbang (0^o). Perubahan kemiringan sudut pada *prototype* sepeda kemudian dibaca oleh sensor pembaca sudut yaitu MPU 6050. Sudut hasil

pembacaan sensor tersebut diolah oleh perangkat keras laptop melalui software arduino sehingga masukan bisa dikomunikasikan dengan *mikrokontroller* pada alat. Setelah menerima masukan berupa sudut, maka aktuator berupa motor DC brushless akan menggerakkan putaran *spin* giroskop.



Gambar 4.3 Konfigurasi sistem kendali *prototype* sepeda dan perangkat kerasnya

4.3 Komponen Perangkat Keras Sistem Pengendalian *Prototype* Sepeda

Untuk membuat sistem pengendalian yang terotomasi pada *prototype* sepeda ini, maka diperlukan proses perangkaian yang membutuhkan kombinasi antara perangkat keras dan lunak. Aktuator yang digunakan untuk mendukung pengendalian kecepatan *spin* giroskop pada penelitian ini adalah motor DC brushless tiga fasa. Aktuator ini memerlukan beberapa komponen perangkat keras untuk meneruskan perintah dari input yang kemudian menjalankan putaran motor. Perangkat keras yang diperlukan dalam penelitian ini antara lain :

a) Mikrokonroller Arduino Uno

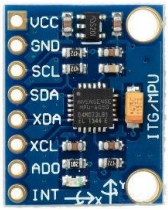
Arduino Uno adalah arduino board yang menggunakan mikrokontroler ATmega328. Arduino Uno memiliki 14 pin digital (6 pin dapat digunakan sebagai output PWM), 6 input analog, sebuah 16 MHz osilator kristal, sebuah koneksi USB, sebuah konektor sumber tegangan, sebuah header ICSP, dan sebuah tombol reset. Arduino Uno memuat segala hal yang dibutuhkan

untuk mendukung sebuah mikrokontroler. Hanya dengan menhubungkannya ke sebuah komputer melalui USB atau memberikan tegangan DC dari baterai atau adaptor AC ke DC sudah dapat membuanya bekerja. Arduino Uno menggunakan ATmega16U2 yang diprogram sebagai USB-to-serial converter untuk komunikasi serial ke computer melalui port USB. Tampak atas dari arduino uno dapat dilihat pada gambar 4.4.

Gambar 4.4 Papan Arduino Uno

b) Sensor GY-521 (MPU 6050)

GY-521 adalah sebuah modul *Inertial Measurement Unit* (IMU) yang menggunakan chip MPU-6050 dari *InvenSense*. MPU-6050 sendiri adalah chip dengan 3-axis *Accelerometer* (sensor percepatan) dan 3-axis *Gyroscope* (pengatur keseimbangan), atau dengan kata lain 6 *degrees of freedom (DOF)* IMU. Selain itu, MPU-6050 sendiri sudah memiliki *Digital Motion Processors (DMP)*, yang akan mengolah data mentah dari masing-masing sensor. Sejumlah data tersebut akan diolah menjadi data dalam bentuk *quaternions* (4 Dimensi). DMP pada MPU6050 juga berfungsi meminimalisasi error yang dihasilkan.



Gambar 4.5 Modul IMU GY-521 yang berbasis MPU 6050

MPU 6050 adalah chip IC *inverse* yang didalamnya terdapat sensor *Accelerometer* dan *Gyroscope* yang sudah terintergrasi. *Accelerometer* digunakan untuk mengukur percepatan, percepatan gerakan dan juga percepatan gravitasi. *Accelerometer* sering digunakan untuk menghitung sudut kemiringan, dan hanya dapat melakukan dengan nyata ketika statis dan tidak bergerak. Untuk mendapatkan sudut akurat kemiringan, sering dikombinasikan dengan satu atau lebih gyro dan kombinasi data yang digunakan untuk menghitung sudut. *Gyroscope* adalah perangkat untuk mengukur atau mempertahankan orientasi, yang berlandaskan pada prinsip-prinsip momentum sudut.

c) Motor DC Brushless

BLDC motor atau dapat disebut juga dengan BLAC motor merupakan motor listrik *synchronous* AC 3 fasa. Perbedaan pemberian nama ini terjadi karena BLDCM memiliki BEMF berbentuk *trapezoid*, sedangkan BLACM memiliki BEMF berbentuk sinusoidal. Walaupun demikian keduanya memiliki struktur yang sama dan dapat dikendalikan dengan metode six-step maupun PWM sinusoidal. Dibandingkan dengan motor DC, BLDCM memiliki biaya perawatan yang lebih rendah dan kecepatan yang lebih tinggi akibat tidak digunakannya *brush*. Dibandingkan dengan motor induksi, BLCM memiliki efisiensi yang lebih tinggi karena rotor dan torsi awal yang lebih tinggi karena rotor terbuat dari magnet permanen. Walaupun memiliki kelebihan dibandingkan dengan motor DC dan motor induksi, pengendalian BLDCM jauh lebih rumit untuk kecepatan dan torsi yang konstan karena tidak adanya *brush* yang menunjang proses komutasi dan harga BLDCM jauh lebih mahal.

Motor DC brushless yang digunakan pada penelitian ini adalah LD Power MT-3510 700kv multi-copter motor dengan spesifikasi fisik yaitu diameter 42 mm, panjang 26 mm, diameter stator 35 mm, panjang *shaft* 4 mm, berat 102 gram dan input baterai

3-6 cells. Penampakan fisik LD Power MT-3510 700kv ditunjukkan pada gambar 4.6.

Gambar 4.6 LD Power MT-3510 700kv

Untuk spesifikasi kecepatan, voltase masukan dan arus yang dimiliki oleh LD Power MT-3510 700kv ditampilkan melalui tabel 4.3.

Tabel 4.3 Spesifikasi oleh LD Power MT-3510 700kv

Voltage(V)	Propeller (inch)	Throttle	Current (A)	Watts(W)	Thrust(G)	RPM (RPM/Min)	Efficiency (G/W)
11.1	APC 12*3.8	50%	2.6	28.86	378	3567	13.10
		65%	4.4	48.84	516	4133	10.57
		75%	8	88.8	823	4776	9.27
		85%	11.3	125.43	1088	5189	8.67
		100%	15.5	172.05	1245	5789	7.24
	APC13*4	50%	2.5	27.75	358	3840	12.90
		65%	4	44.4	509	4589	11.46
		75%	6	66.6	725	5444	10.89
		85%	8.5	94.35	850	5789	9.01
		100%	10.6	117.66	1020	6210	8.67
	APC11*4.7	50%	4	59.2	566	4536	9.56
		65%	6.5	96.2	800	5377	8.32
		75%	11	162.8	1109	6342	6.81
		85%	14.3	211.64	1290	6789	6.10
14.8		100%	19	281.2	1556	7444	5.53

d) *Step Down* AC/DC-DC LM2596

Modul regulator LM2596 adalah suatu regulator yang berfungsi untuk menurunkan tegangan (step down). Regulator ini mampu mengeluarkan arus maksimal 3 A, dengan daya input 3.5 V sampai 40 V, dan daya output 1.2 V sampai 37 V. *Range* tegangan input minimal 1.5 volt lebih besar dari tegangan input. Dalam rangkaian ini, untuk menurunkan tegangan dari sumber listrik menggunakan modul regulator LM2596S, regulator tersebut digunakan untuk menurunkan tegangan dari battery aki atau dari *power supply* menjadi tegangan 11,2 volt yang dapat digunakan sebagai *input* arduino.

Gambar 4.7 Modul step down LM2596

Modul regulator penurun tegangan ini mengunakan bahan *solid capacitor* dan PCB berkualitas untuk menjamin kualitas tegangan yang dibutuhkan. Untuk menyesuaikan tegangan cukup dengan memutar potensio yang ada pada board. Perlu diperhatikan pada tanda *input* dan *output*, serta polaritas positif dan negatif jangan sampai terbalik karena akan merusak modul.

Pada rangkaian komponen perangkat keras ini menggunakan dua buah modul *step down* LM2596. Modul pertama digunakan untuk sambungan menuju komponen *receiver* dan modul kedua digunakan untuk sambungan menuju arduino.

e) *Power Supply*

Power Supply atau dalam bahasa indonesia disebut dengan catu daya adalah suatu alat listrik yang dapat menyediakan energi listrik untuk perangkat listrik ataupun elektronika lainnya yang ditunjukkan pada gambar 4.8. Oleh karena itu, *power supply* kadang-kadang disebut juga dengan istilah *Electric Power Converter*. Pada tugas akhir ini digunakan 1 buah *box power supply power up* 500W dengan voltase AC *input* 110V/220V, besar arus maksimum 8A dan frekuensi 50-60 Hz, di mana *power supply* ini digunakan sebagai masukan daya pada ESC untuk menggerakkan motor DC *brushless* pada *self-balancing bike.*

Gambar 4.8 *Power supply power up*

4.4 *Digital Motion Processing Filter*

Sistem kendali PID memerlukaan *feedback* untuk menentukan nilai error terhadap *setpoint*. Seperti yang telah disebutkan pada subbab 4.3 bahwa sensor yang digunakan untuk memberikan feedback pada sistem berupa sudut θ terukur adalah sensor MPU6050. Sensor MPU6050 merupakan salah satu sensor IMU(*Inertia Measurement Unit*) yang memiliki 6 output nilai keluaran. Enam nilai *output* tersebut yakni tiga *axis accelerometer* dan tiga *axis gyroscope* masing-masing berturut-turut a_x, a_y, a_z, dan g_x, g_y, g_z.

4.4.1 Masalah pada *accelerometer*

Accelerometer dapat mengukur semua gaya yang bekerja pada suatu objek. Akan terdapat banyak gaya-gaya yang dapat terukur tidak hanya gravitasi saja. Setiap gaya-gaya kecil sekalipun yang bekerja pada objek akan mengganggu pengukuran. Hal ini yang membuat nilai pengukuran menjadi sangat acak apabila terkena gaya dari luar meskipun kecil, sebagai contoh adalah getaran.

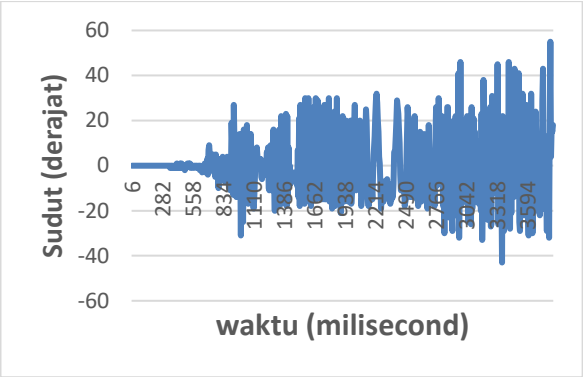
4.4.2 Masalah pada gyroscope

Gyroscope dapat memberikan pengukuran yang akurat yang tidak terpengaruh oleh gaya dari luar. Hal yang menjadi kelemahan dari *gyroscope* adalah nilai pengukuran yang didapat tidak bias kembali meskipun sistem telah kembali ke posisi semula.

4.4.3 Filter

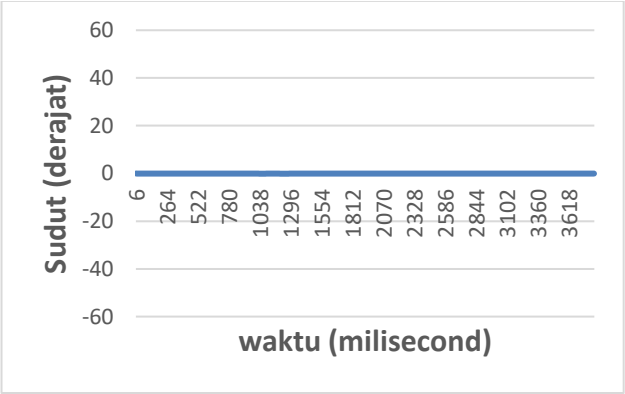
Penelitian ini menggunakan *digital motion processing filter* untuk menyaring data yang diberikan oleh sensor IMU. Prinsip dari *digital motion processing filter* yaitu menggabungkan dua kemampuan dari dari *accelerometer* dan *gyroscope* untuk mendapatkan nilai pengukuran yang baik.

Fungsi dari *digital motion processing filter* adalah mengambil nilai sudut baru dari *gyroscope* setiap waktu. *Filter* lalu mengecek nilai gaya-gaya yang terukur dari *accelerometer*. Jika nilai dari gaya-gaya tersebut terlalu besar atau terlalu kecil, *filter* menganggapnya sebagai gangguan dan tidak akan memsukkan nilai kedalam perhitungan. Nilai pengukuran tanpa menggunakan *digital motion processing filter* ditunjukkan pada gambar 4.9.



Gambar 4.9 Data sensor IMU tanpa filter

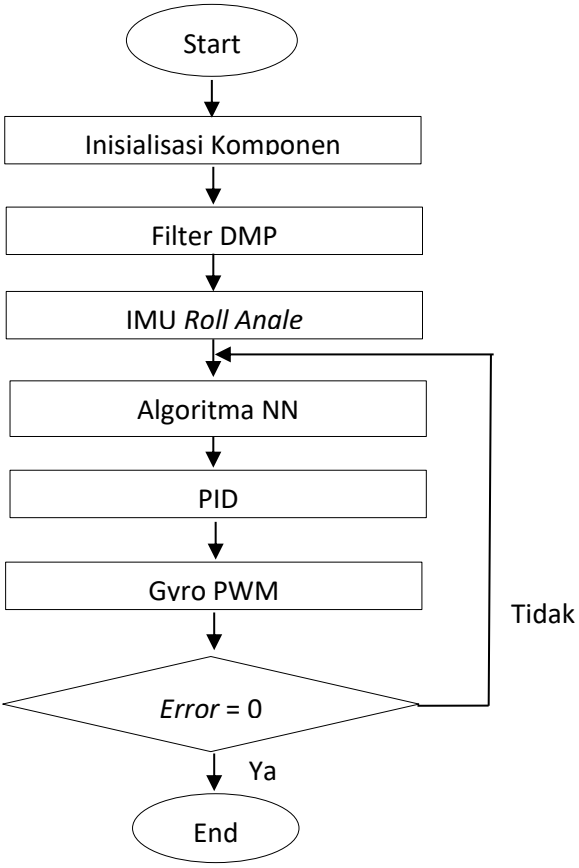
Setelah penggunaan filter DMP, maka disturbance pada sistem yang mengganggu pengukuran sudut melalui IMU disaring sehingga pembacaan yang dikeluarkan oleh sensor menjadi lebih stabil. Pengukuran tanpa menggunakan *digital motion processing filter* ditunjukkan pada gambar 4.10.



Gambar 4.10 Data sensor IMU menggunakan DM

4.5 Pemrograman Arduino Uno

Sebelum membuat program pada arduino uno, pembuatan *flowchart* proses pemrograman harus dilakukan terlebih dahulu. *Flowchart* ini berfungsi sebagai acuan dasar dalam pemrograman yang akan dilakukan lihat gambar 4.11.



Gambar 4.11 Flowchart pemrograman pada arduino

1) Inisialisasi Komponen

Tahap pertama yang dilakukan dalam pemrograman arduino adalah pengenalan variabel dan komponen yang nantinya akan diproses pada sistem pengendalian.Variabel dan komponen yang dimasukkan dalam program ini terdapat variabel untuk sensor IMU MPU6050, PI, serta input dan output untuk motor DC *brushless* .Gambar 4.12 ini adalah pemrograman awal tahap pengenalan variabel dan komponen.

```
#include <Servo.h>
Servo firstESC;
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#if I2CDEV_IMPLEMENTATION ==
I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif
MPU6050 mpu;
#define OUTPUT_READABLE_YAWPITCHROLL
#define LED_PIN 13
bool blinkState = false;
bool dmpReady = false;
uint8_t mpuIntStatus;
uint8_t devStatus;
uint16_t packetSize;
uint16_t fifoCount;
uint8_t fifoBuffer[64];
Quaternion q;           // [w, x, y, z]
VectorInt16 aa;         // [x, y, z]
VectorInt16 aaReal;     // [x, y, z]
VectorInt16 aaWorld;    // [x, y, z]
VectorFloat gravity;    // [x, y, z]
float euler[3];         // [psi, theta, phi]
float ypr[3];           // [yaw, pitch, roll]
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00,
'\r', '\n' };
```

Gambar 4.12 Pengenalan variabel dan komponen

2) Pembacaan Sensor IMU MPU6050

Pada tahap ini dilakukan pemrograman untuk sensor IMU MPU6050. MPU6050 untuk pembacaan sudut berturut-turut sambungan INT dihubungkan dengan pin 2,

SDA dengan pin A4, SCL dengan pin A3, ground dengan pin ground dan VCC dengan pin 5V pada arduino.

```
void setup() {
  firstESC.attach(9);
  int PWM = 1000;
  firstESC.writeMicroseconds(PWM);
  // join I2C bus (I2Cdev library doesn't do this automatically)
  #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    Wire.begin();
    TWBR = 24;
  #elif I2CDEV_IMPLEMENTATION ==
I2CDEV_BUILTIN_FASTWIRE
    Fastwire::setup(400, true);
  #endif

  Serial.begin(9600);
  while (!Serial);


  devStatus = mpu.dmpInitialize();

  mpu.setXGyroOffset(220);
  mpu.setYGyroOffset(76);
  mpu.setZGyroOffset(-85);
  mpu.setZAccelOffset(1788); // 1688 factory default for my test chip
```

Gambar 4.13 IMU Raw Data

3) Filter *Digital Motion Processing*

Penelitian ini menggunakan *digital motion processing filter* untuk menyaring data yang diberikan oleh sensor IMU. Prinsip dari *digital motion processing filter* yaitu menggabungkan dua kemampuan dari dari *accelerometer* dan *gyroscope* untuk mendapatkan nilai pengukuran yang baik. Penelitian ini menggunakan library DMP dari i2cDevlib. library ini menggunakan sensor fusion bawaan dari MPU6050 yang dikembangkan oleh invense. library DMP pada dasarnya digunakan untuk memperoleh sudut Yaw, Pitch dan Roll dari 6 data akselerasi pada sumbu x, y dan z, dan percepatan sudut pada sumbu x, y dan z.

```
            if (devStatus == 0) {
                    mpu.setDMPEnabled(true);

                  // enable Arduino interrupt detection
                 //  Serial.println(F("Enabling interrupt detection (Arduino
external interrupt 0)..."));
                  attachInterrupt(0, dmpDataReady, RISING);
                  mpuIntStatus = mpu.getIntStatus();

                  // set our DMP Ready flag so the main loop() function knows
it's okay to use it
                 // Serial.println(F("DMP ready! Waiting for first interrupt..."));
                  dmpReady = true;

                  // get expected DMP packet size for later comparison
                  packetSize = mpu.dmpGetFIFOPacketSize();
                } else {
                  // ERROR!
                  // 1 = initial memory load failed
                  // 2 = DMP configuration updates failed
                  // (if it's going to break, usually the code will be 1)
                 // Serial.print(F("DMP Initialization failed (code "));
                 // Serial.print(devStatus);
                  Serial.println(F(")"));
                }//

                // configure LED for output
                pinMode(LED_PIN, OUTPUT);
```

Gambar 4.14 *Digital Motion Processing* filter

4) Menampilkan otput pembacaan sudut IMU MPU6050

Setelah inputan raw data pembacaan sudut dimasukkan dan mengalami proses filter dengan DMP, maka selanjutnya output sudut roll ditampilkan melalui layar serial monitor.

```
#endif
    // subrutin yang akan diproses
    #ifdef OUTPUT_READABLE_YAWPITCHROLL
      // display Euler angles in degrees
      mpu.dmpGetQuaternion(&q, fifoBuffer);
      mpu.dmpGetGravity(&gravity, &q);
      mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
```

```
  //Serial.print("ypr\t");
  //Serial.print(ypr[0] * 180/M_PI);
  //Serial.print("\t");
  //Serial.print(ypr[1] * 180/M_PI);
  //Serial.print("\t");
  //Serial.print("Sudut Roll,millis : ");
int PWM=1000; // safety lock brushless
  //delay(100);
 // PWM=1100;
 // firstESC.writeMicroseconds(PWM);
```

Gambar 4.15 Pemrograman untuk menampilkan output sudut

5) Program *Neural Network* dan *Proportional-Integral*

Gambar 4.16 menunjukkan pemrograman untuk persamaan *neural network* yang akan digunakan. Persamaan *neural network* didapatkan dari pengelolaan data pada matlab. *Layer* dan *neuron* yang digunakan akan mempengaruhi bentuk dari persamaan *neural network*. Gambar 4.25 menunjukkan persamaan *neural network* yang menggunakan 2 layer, yaitu 1 *hidden layer* dengan 1 *layer output*. Banyaknya *layer* dan *neuron* akan menyebabkan persamaan *neural network* menjadi panjang.

```
void NN () {
  //
  M11 = 1/(1+exp(-((simpan_error*14,0005)-(13,9995))));
  M12 = 1/(1+exp(-((simpan_error*-14)+(-10,8889))));
  M13 = 1/(1+exp(-((simpan_error*-14)+(7,7778))));
  M14 = 1/(1+exp(-((simpan_error*14)+(-4,6664))));
  M15 = 1/(1+exp(-((simpan_error*14,0045)+(-1,5246))));
  M16 = 1/(1+exp(-((simpan_error*-13,9906)+(-1,6621))));
  M17 = 1/(1+exp(-((simpan_error*13,9976)+(4,673))));
  M18 = 1/(1+exp(-((simpan_error*-0,14083)-(7,7653))));
  M19 = 1/(1+exp(-((simpan_error*-13,968)+(-10,9099))));
  M110 = 1/(1+exp(-((simpan_error*13,9474)+(14,0525))));


 Mtotal1 = (M11*-0,45385)+(M12*0,4786)+(M13*-
0,10432)+(M14*0,065764)+(M15*-0,46882)+(M16*-0,10037)+(M17*-
0,81076)+(M18*0,3526)+(M19*0,65113)+(M110*0,37804)+1,2395;
```

```
Mtotal2 = (M11*-0,77157)+(M12*-0,5852)+(M13*-0,33223)+(M14*-
0,35966)+(M15*-0,82855)+(M16*-
0,46613)+(M17*0,39612)+(M18*0,38734)+(M19*0,28854)+(M110*-
0,11847)-1,3369;
```

Gmbar 4.16 Program *Neural Network*

Gambar 4.17 menunjukkan pemograman untuk sistem kontrol PI. Program PI ini disesuaikan dengan rumus dari masing - masing sistem pengendali PI yaitu sistem pengendali *proportional* dan sistem pengendali *integral* dengan nilai Kp dan Ki yang telah ditentukan.

Program PI-NN merupakan gabungan dari program NN dan PI. Output dari program NN akan diberikan pada program PI. Output tersebut adalah nilai Kp dan Ki yang telah diolah pada program NN. nilai Kp dan Ki yang diberikan akan dikalikan dengan nilai error dari selisih antara *set point* dengan hasil pembacaan MPU6050.

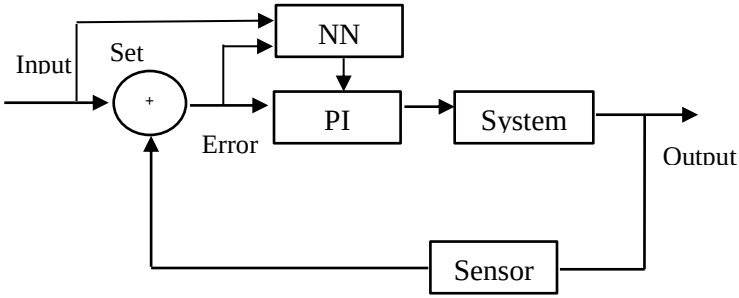
```
//Algoritma PID
void pid(){
//ROLL
kp = M16;
ki = M17;
error = simpan_error;
error_i = error-error_last ;
p = kp * error;
i = ki * error_i;
pi=abs(p+i);
error_last = error;
```

Gambar 4.17 Program *proportional-integra*

BAB V
ANALISA DATA DAN PEMBAHASAN

5.1 Pengambilan Data

Pengambilan data dilakukan dengan melakukan uji coba langsung terhadap *prototype* sepeda yang telah dilengkapi dengan komponen-komponen hardware pengendalian dibuat sebelumnya. Blok diagram yang digunakan untuk pengambilan data pada penelitian ini ditunjukkan pada gambar 5.1.



Gambar 5.1 Blok diagram NN-PI pada *prototype* sepeda

Sistem pengendali yang terdapat pada *prototype* tersebut adalah PID *controller*. Input yang digunakan untuk PID *controller* adalah error kemiringan sudut yang terjadi akibat dari pemberian gangguan (*disturbance*) pada *prototype*. Output dari *controller* adalah sudut. Sudut tersebut digunakan sebagai input untuk *prototype* sepeda. Output dari sistem adalah sudut roll yang nantinya akan memutar kecepatan motor DC *brushless*.

Blok PID *controller* memiliki persamaan PID. Persamaan pada parameter PID yang dibutuhkan pada penelitian ini adalah nilai *proportional gain* (Kp) dan *Integral gain* (Ki. Nilai tersebut adalah faktor pengali untuk nilai error yang terjadi pada sistem. Untuk kebutuhan pengambilan data pada *prototype*, K nilai yang

diubah adalah nilai Kp dan Ki yang berfungsi untuk mengubah variasi kecepatan motor DC *brushless*.

Pengambilan data dilakukan dengan menyusun kemungkinan error yang dihasilkan oleh *prototype*. Untuk variasi sudut *roll* kemiringan *prototype* sepeda yang bisa diberikan adalah pada rentang sudut 1^o hingga 8^o dengan interval pengambilan data senilai 1^o.

Masing-masing nilai error tersebut disimulasikan dengan memberikan nilai Kp dan Ki pada masing-masing sudut. Nilai Kp ditentukan pada rentang nilai 3,10 dan 20 sedangkan nilai Ki antara nilai 1, 3 dan 5. Karakteristik respon yang diambil pada setiap pengambilan data adalah *%overshoot, Rise time,* dan *Steady-state Error*. Pengambilan data dilakukan antara rentang Kp dan Ki yang ditentukan untuk setiap kondisi sudut. Kemudian nilai-nilai tersebut dihitung dengan menggunakan persamaan 3.1 sehingga diperoleh data *performance index* (F). Data dari nilai F tersebut diambil data yang memiliki nilai F terkecil pada masing-masing nilai error di setiap percobaan. Data tersebut menjadi input untuk pelatihan *neural network*. Output yang digunakan adalah nilai Kp *roll*, Ki *roll*. Data output diperoleh dari nilai Kp dan Ki pada masing-masing nilai error yang mempunyai nilai F paling kecil.

5.2 Hasil Pengambilan Data

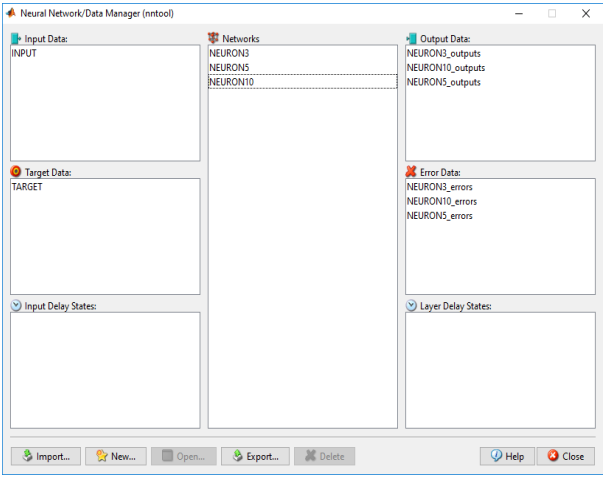
Pengambilan data dilakukan secara langsung dengan menggunakan prototype sepeda yang telah dibuat beserta blok diagram pada gambar 5.1. Sesuai dengan persamaan 3.1, parameter yang diambil adalah %*overshoot* (OS), *rise time* (RT), dan *steady-state error*. Dengan kombinasi nilai error pada masing-masing sudut, nilai K_P *roll*, K_I *roll*, diperoleh data dengan rentang sebanyak 200-300 data pada setiap variasi sudut. Data tersebut kemudian dipilih sesuai dengan nilai F yang mempunyai nilai terkecil pada setiap nilai error. Sehingga diperoleh data yang ditunjukkan tabel 5.1

Tabel 5.1 Data untuk pelatihan *neural network* untuk variasi sudut *roll*

Error	Kp	Ki	OS (%)	RT (s)	SSE (derajat)	F
1	20	5	2.857	0.427	0.056	154.19
2	10	5	5.5	2.67	0.079	332.35
3	20	5	6.68	1.067	0.17	363.84
4	20	5	1.256	1.386	0.301	105.57
5	20	3	4.21	0.535	0.0082	221.61
6	20	5	0.67	1.6	0.618	96.4
7	20	5	13	1.4	0.0967	682.835
8	20	1	20.37	0.959	0.64	1069.68

5.3 Pelatihan Neural Network

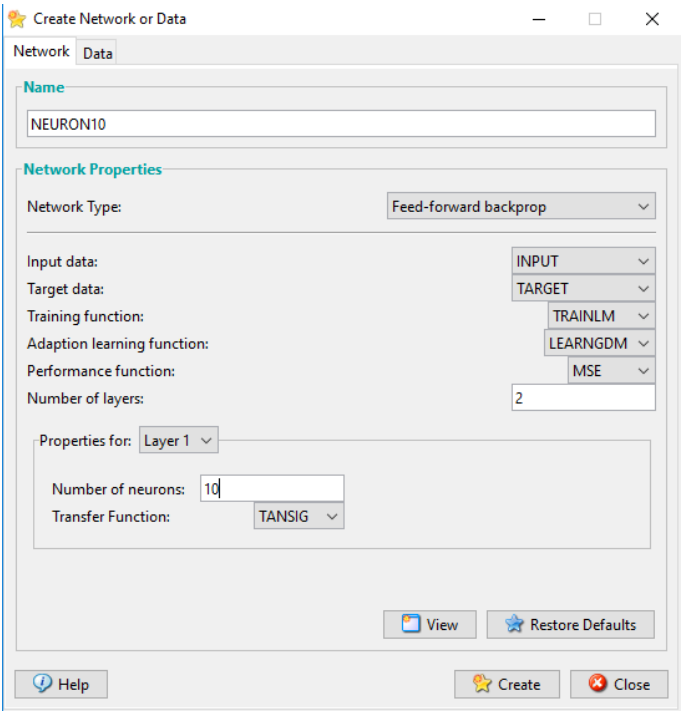
Pelatihan *neural network* dilakukan dengan *software* matlab. Pada *software* matlab terdapat *tool* berupa *data manager* untuk mengambil, membuat, menggunakan, dan memindahkan networks dan data. Dengan menggunakan syntax nntool, *data manager* dapat dipanggil seperti gambar dibawah ini:



Gambar 5.2 *Toolbox data manager*

Langkah selanjutnya yaitu memasukkan data untuk pelatihan *neural network*. Data pada tabel 5.1 terdapat nilai error dan nilai K_P dan K_I. Nilai error tersebut dimasukkan ke dalam input data, sedangkan nilai K_P dan K_I dimasukkan ke dalam target data.

Untuk membuat *neural network* dilakukan dengan memilih *new* pada *toolbox data manager*. Terdapat beberapa pilihan pada *toolbox create network or data* untuk menentukan model *neural network* yang diinginkan, seperti menentukan tipe dari *neural network*, jumlah *layer* dan *neuron*, dan lain-lain seperti pada gambar 5.3.



Gambar 5.3 *Toolbox create network or data*

5.4 Pelatihan *Neural Network* untuk Variasi Sudut *Roll*

Untuk sudut *roll* digunakan *neural network feed-forward backpropagation* dengan satu input *layer*, satu *hidden layer* dan satu output *layer*. Pada *hidden layer* dilakukan variasi jumlah *neuron* yang dimasukkan dengan rentang *neuron* sebanyak 3, 5 dan 10 *neuron* serta pada output *layer* terdapat 2 *neuron*. Pada penelitian ini dilakukan percobaan sebanyak 5 kali *training* data pada setiap variasi jumlah neuron. Kombinasi *layer* tersebut diperoleh dengan *trial* dan *error* pada pelatihan *neural network*. Hasil trial dan error adalah sebagai berikut:

Tabel 5.2 Hasil *trial and error* pada *training* data sudut *roll*

No	Input	Hidden Layer	Output Layer	Neuron Output Layer	Best MSE
1	1	3	1	2	2.99E-09
	1	3	1	2	5.00E-10
	1	3	1	2	1.29E-10
	1	3	1	2	2.78E-10
	1	3	1	2	1.33
2	1	5	1	2	2.55E-11
	1	5	1	2	1.80E-11
	1	5	1	2	3.33E-11
	1	5	1	2	1.33E-10
	1	5	1	2	9.30E-11
3	1	10	1	2	6.74E-16
	1	10	1	2	1.58E-15
	1	10	1	2	1.53E-15
	1	10	1	2	4.90E-09
	1	10	1	2	3.82E-09

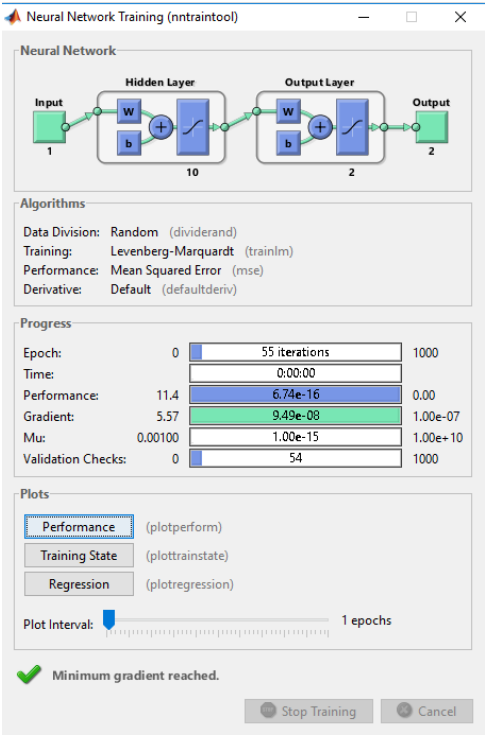
Tabel 5.2 adalah hasil *trial* dan *error* untuk sudut *roll* yang dilakukan sebanyak lima kali. Percobaan tersebut dilakukan dengan mengubah jumlah *hidden layer* dan jumlah *neuron* pada

hidden layer. Pada percobaan pertama, diatur *hidden layer* yang berjumlah satu dengan jumlah *neuron* 3. Diperoleh *mean square error* (MSE) terbaik yang diperoleh adalah 1,29 x 10^{-10}. Untuk percobaan kedua, jumlah *neuron* pada *hidden layer* ditambah menjadi 5 *neuron* sehingga dihasilkan MSE sebesar 1,8 x 10^{-11}. Kemudian untuk percobaan ketiga jumlah *neuron* pada *hidden layer* kembali ditambahkan menjadi 10, sehingga menghasilkan MSE sebesar 6,74× 10^{-16}.

Tabel 5.3 *Mean Square Error* yang dihasilkan oleh setiap percobaan variasi *neuron*

No	Input	Hidden Layer	Neuron Hidden Layer	Output Layer	Neuron Output Layer	Best MSE
1	1	1	3	1	2	1.29 E-10
2	1	1	5	1	2	1.80 E-11
3	1	1	10	1	2	6.74 E-16

Dengan kombinasi *layer* dan *neuron* diatas, diperoleh performa terbaik dari *neural network* yang diukur dengan *mean square error* (MSE) yaitu 6,74 × 10^{-16}. Adapun parameter-parameter yang membatasi pelatihan *neural network* adalah gradien maksimum yang diatur pada nilai 10^{-10}, jumlah iterasi dibatasi hingga mencapai nilai 1000, dan validasi diatur di angka 1000. Hasil dari performa *neural network* setelah pelatihan ditunjukkan pada gambar 5.4.



Gambar 5.4 Performa dari *neural network* untuk sudut *roll* setelah *training*

Pada gambar 5.4 terdapat hasil pengujian hasil dari *neural network*. Perfoma yang diukur adalah *mean square error*. Setelah menempuh 55 kali iterasi, diperoleh MSE sebesar $6{,}74 \times 10^{-16}$, gradien sebesar $9{,}49 \times 10^{-8}$, dan validasi *neural network* sebanyak 1000 kali.

Hasil dari pelatihan tersebut diperoleh persamaan *neural network*. Di dalam persamaan tersebut terdapat nilai *weight* dan *bias* yang berbentuk nilai matriks, sebagai berikut :

$$w^{(1)} = \begin{bmatrix} 14{,}0005 \\ 14 \\ 14 \\ 14 \\ 14{,}0045 \\ -13{,}9906 \\ 13{,}9976 \\ -14{,}0083 \\ -13{,}9849 \\ 13{,}9474 \end{bmatrix}$$

$$b^{(1)} = \begin{bmatrix} -13{,}995 \\ -10{,}8889 \\ -7{,}7778 \\ -4{,}6664 \\ -1{,}5246 \\ -1{,}6211 \\ 4{,}673 \\ -7{,}7653 \\ -10{,}9099 \\ 14{,}0525 \end{bmatrix}$$

Dari perhitungan nilai *weight* dan *bias* pada *hidden layer* dihasilkan matriks 10×1. Perhitungan tersebut diperoleh dari persamaan berikut:

$$u = \left(w^{(1)} \times input\right) + {w_0}^{(1)} \qquad 5.1$$

Nilai z akan melewati fungsi aktivasi persamaan 2.1. persamaan sebagai berikut:

$$f(u) = \frac{1}{1+\exp(u)} \qquad 5.2$$

Sedangkan pada output *layer*, *weight* dan *bias neural network* adalah sebagai berikut:

$$w^{(2)} = \begin{bmatrix} -0{,}45385 & 0{,}47865 & -0{,}10432 & 0{,}065764 & 0{,}48662 & -0{,}90641 & -0{,}10037 & -0{,}81076 & 0{,}65113 & 0{,}37804 \\ -0{,}77517 & -0{,}58525 & -0{,}33223 & -0{,}35966 & -0{,}82885 & -0{,}46613 & 0{,}39612 & 0{,}38734 & 0{,}28854 & -0{,}11847 \end{bmatrix}$$

$$b^{(2)} = [1{,}2395 \quad -1{,}3369]$$

Dari perhitungan nilai *weight* dan *bias* pada output *layer* dihasilkan matriks 2×1. Perhitungan tersebut diperoleh dari persamaan berikut:

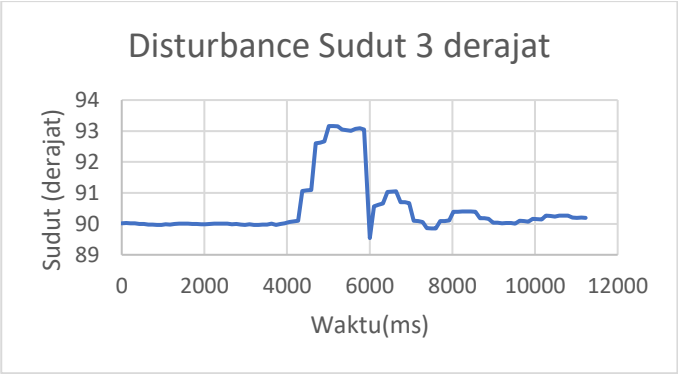
$$y = \left(w^{(2)} \times f(u)\right) + {w_0}^{(2)} \qquad 5.3$$

Sehingga persamaan *neural network controller* untuk sudut rotasi adalah:

$$\begin{bmatrix} k_p \\ k_i \end{bmatrix} = \left(w^{(2)} \times f(u)\right) + {w_0}^{(2)} \qquad 5.4$$

5.5 Pengujian Prototype PI Controller dengan PI-NN Controller

Pengujian program pada *prototype turret gun* dilakukan dengan memberikan input dari koding pada software arduino. Koding dari arduino kemudian dilakukan proses *uploading* yang lalu memunculkan data pada *serial monitor*. Pengambilan data dimulai dengan memberikan program pada arduino untuk memunculkan data pengulangan dari program. Data tersebut terdiri dari nilai error pembacaan MPU6050 dan waktu perubahan yang dialami. Setiap pengujian pengambilan data yang dilakukan akan diambil 200 hingga 300 data pengulangan yang terjadi.



Gambar 5.5 Grafik respon sistem untuk sudut roll dengan *disturbance* 3 derajat

Pada gambar 5.5 merupakan grafik yang ditunjukkan pada saat *prototype* sepeda diberikan *disturbance* sehingga mengalami kemiringan sudut sebesar 3º dari posisi seimbang. Dapat dilihat bahwa sistem telah menunjukkan respon mendekati niali aktual yang diinginkan. Saat diberikan gangguan sebesar 3º respon mengalami besaran *overshoot* sebesar 15,28%, waktu kenaikan *rise time* sebesar 0,55 *seconds* dan *steady state error* menunjukkan nilai 0,217.

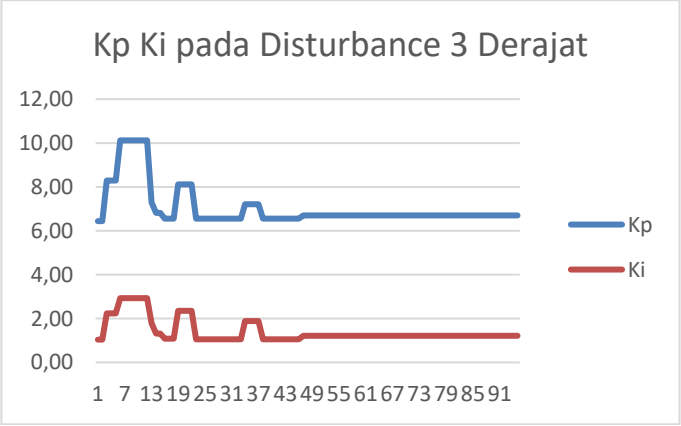
Tabel 5.4 Data respon pengaplikasian NN-PI *Controller* terhadap sistem

Error	OS	RT	SSE	F
1	2.53	0.4021	0.05	137.042
2	5.012	2.67	0.069	307.45
3	6.521	1	0.15	353.55
4	1.211	1.317	0.301	101.94
5	4.176	0.501	0.0066	219.15
6	0.5	1.55	0.501	81.05
7	12.67	1.38	0.0877	665.485
8	19.916	0.913	0.0911	1018.615
Rata-rata	6.567	1.216638	0.543	360.5353

Tabel 5.4 adalah hasil pengujian *prorotype turret gun* untuk mengetahui performa dari sistem pengendali NN-PI pada sudut roll kemiringan alat. Dengan pengujian pemberian *set point* yang telah ditentukan pada bab sebelumnya, didapatkan *%overshoot* (%OS) rata-rata adalah 6,657 %. Sedangkan untuk *rise time* (RT) rata-rata dari sistem adalah 1,2166 detik. Kemudian untuk *steady-state error* (SSE) rata-rata adalah 0,543 derajat dan *index performance* (F) sebesar 360,5353

5.6 Parameter Gain Proportional-Integral pada Sistem Pengendali NN

Parameter *proportional* gain (K_P) dan *integral* gain (K_I) pada *controller* yang diterapkan dengan metode *neural network* memiliki nilai kedua parameter yang dapat berubah. Perubahan ini terjadi disetiap waktu dengan mengikuti perubahan dari nilai error yang terjadi. Nilai Kp dan Ki dapat berubah menyesuaikan nilai error yang terjadi.



Gambar 5.6 Grafik Kp dan Ki pada *disturbance* dengan sudut 3 derajat

Gambar 5.6 terdapat grafik nilai Kp dan Ki pada *disturbance* dengan sudut 3 derajat. Grafik menunjukkan bahwa perubahan nilai K_P untuk sudut rotasi berkisar di angka 6 sampai 10. Perubahan nilai Ki untuk sudut rotasi berkisar di angka 1 sampai 3. Pada detik 27 Kp dan Ki mulai stabil menunjukkan nilai error juga sudah stabil.

BAB VI
KESIMPULAN DAN SARAN

4.2 Kesimpulan

Berdasarkan hasil analisa yang telah dijelaskan pada bab-bab sebelumnya, maka dapat disimpulkan bahwa:

1. Sistem *prototype* sepeda yang diberikan *controller* dengan menggunakan metode *neural network* menghasilkan mean square error terbaik $6{,}74 \times 10^{-16}$ dengan jumlah neuron sebanyak 10 buah.
2. Perancangan sistem kendali yang telah dibangun mampu mengendalikan *prototype* sepeda. Pada pengendalian PI-NN dapat mengurangi rata-rata *overshoot* sebesar 6,8178% menjadi 6,567% (turun 3,679%), menurunkan *rise time* sebesar 1,2555 detik menjadi 1,2166 detik (turun 3,069%) dan *steady state error* sebesar 0,246 derajat menjadi 0,543 derajat (15,15%).

4.3 Saran

Adapun saran yang dapat digunakan untuk pengembangan selanjutnya adalah sebagai berikut:

1) Data yang digunakan untuk pelatihan *neural network* dapat ditingkatkan sehingga hasil dari *neural network* lebih akurat.
2) Untuk pengendalian sistem self balancing bike dengan hanya mengatur kecepatan *spin* giroskop membutuhkan motor dengan spesifikasi rpm dengan range cukup besar pada kisaran 20000 hingga 30000

rpm untuk menahan *disturbance* yang menghasilkan variasi kemiringan sudut lebih besar.

3) Perlunya penggunaan mikrontroler setingkat di atas spesifikasi arduino, yaitu jenis mini-pc seperti Raspberry Pi yang memiliki RAM 32mb hingga 64mb agar proses pemrograman *neural network* bisa berjalan lebih cepat.

DAFTAR PUSTAKA

Aswam, A., Maret. 2015. **Konsep Perancangan Sistem Kontrol** ,<URL:http://www.digilib.itb.ac.id/files/disk1/611/jbptitbpp-gdl-agungaswam-30508-3-2008ta-2.pdf>

Azummar, M., 2012. **Pemodelan dan Simulasi Brushless DC Motor Kecil untuk Aplikasi Aktuator Sirip Roket**. Jakarta : Departemen Elektro Fakultas Teknik Universitas Indonesia.

Gurney, K., 1997. **An Introduction to Neural Networks**, University of Sheffield, London.

Ismeal, G.M. Kyslan, K. Fedak, V., 2012 **DC Motor Identification Based on Recurrent Neural Networks**. Košice: Technical University of Košice.

Kaplan, dan Fineman, A., 2008. **Design of An Active-Assistance Balancing Mechanism for A Bicyle**. Degree of Bachelor of Science in Robotics Engineering, Worcester Polytechnic Institute, United States.

Pitowarno, E., 2006. **Robotika Desain, Kontrol dan Kecerdasan Buatan**. Yogyakarta: Andi Yogyakarta .

Prayogo, R., 2012. **Pengaturan Pulse Width Modulation dengan PLC**. Malang : Fakultas Teknik Elektro Universitas Brawijaya.

Spry, C., Stephen., dan Girard, R., Anouck., 2008. **Gyroscopic Stabilitation of Unstable Vehicles : Configurations, Dynamics and Control**. Lam Research Corporation, Freemont and Department of Aerospace Engineering, University of Michigan.

Sujanarko, B., 2012. **Pengendalian Motor BLDC Menggunakan Jaring Saraf Tiruan**. Jember :

Departemen Elektro Fakultas Teknik Universitas Jember.

Syawal, M., Gaffar, M dan Pratama, Y., 2015. **Perancangan dan Simulasi Smart Door Berbasis Android Menggunakan Arduino Uno**. Jakarta : Universitas Bina Nusantara.

Tefay, B., Eizad, B., Croswhaite, P., Singh, S dan Postula, A., 2011. **Design of an Integrated Electronic Speed Controller for Compact Robotic Vehicles**. Robotic Aircraft Research Group, School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane 4072.

Umam, A., Cahyadi, A.I dan Herdjunanto, S., 2014. **Sistem Kendali PID dan Adaptif untuk Pengendalian Kecepatan Motor DC Berbasis PLC**. Yogyakarta : Fakultas Teknik Jurusan Teknik Elektro Universitas Gadjah Mada.

Wang, G.J., Fong, C.T dan Chang, K., 2001. **Neural Network Based Self Tuning PI Controller for Precise Motion Control of PMAC Motor**. Seoul : IEEE Trans.Ind. Electron.

Xingjia, Y., Lihai, G., Qinding, G., dan Xiaoyan, M., 2010. **RBF Neural Network Based Self-Tuning PID Pitch Control Strategy For Wind Power Generation Sistem**. International Conference on Computer, Mechatronics, Control and Electronic Engineering 1,1:482-485.

Zahra, S., dan Joyodiharjo, J., B., 2014. **Pengembangan Self-Balancing Bike**. Bandung : Jurnal Tingkat Sarjana FSRD Institut Teknologi Bandung.

LAMPIRAN

No	Sudut	Kp	Ki	OS (%)	RT (second)	SSE (derajat)	F
1	1	3	1	64.11	0.105	0.013	3208.25
	1	3	3	76.35	0.32	0.012263	3824.513
	1	3	5	74.75	0.221	0.000789	3741.959
	1	10	1	71.05	0.319	0.06789	3562.275
	1	10	3	61.24	0.746	0.011	3077.47
	1	10	5	5	0.426	0.144	265.72
	1	20	1	11.45	0.426	0.238	592.92
	1	20	3	12	0.427	0.056	611.34
	1	20	5	2.85	0.427	0.0567	153.875
2	2	3	1	74.73	0.32	0.108	3748.3
	2	3	3	65.05	0.961	0.00541	3271.991
	2	3	5	63.5	0.106	0.0155	3177.895
	2	10	1	53.06	0.64	0.015	2666.55
	2	10	3	16.85	0.747	0.00607	857.7435
	2	10	5	5.5	2.67	0.079	332.35
	2	20	1	12	1.28	0.073	629.25
	2	20	3	17.32	0.746	0.0946	885.65
	2	20	5	17.32	0.735	0.094	885.4
3	3	3	1	73.68	0.211	0.062	3691.32
	3	3	3	76.41	1.066	0.0792	3845.78
	3	3	5	82.96	0.426	0.0834	4160.69

	3	10	1	15.28	0.55	0.225	786.25
	3	10	3	19.33	0.96	1.241	1047.75
	3	10	5	13.53	0.64	0.623	720.45
	3	20	1	12.75	0.87	0.05	657.4
	3	20	3	20.3	0.426	0.575	1052.27
	3	20	5	6.68	1.067	0.17	363.84
4	4	3	1	62.81	0.32	0.146	3154.2
	4	3	3	64.25	0.42	0.051	3223.45
	4	3	5	64.23	0.427	0.0385	3221.965
	4	10	1	50.09	0.64	0.061	2520.35
	4	10	3	29	0.96	0.402	1489.3
	4	10	5	38.27	0.746	0.089	1932.87
	4	20	1	33.07	0.64	0.045	1668.55
	4	20	3	29.35	0.747	0.0435	1484.615
	4	20	5	1.25	1.386	0.301	105.27
5	5	3	1	65.32	0.319	0.8455	3314.655
	5	3	3	48.69	0.64	0.411	2467.85
	5	3	5	62.52	0.64	0.127	3145.15
	5	10	1	8	0.959	0.2	429.18
	5	10	3	31.91	0.96	0.1	1619.7
	5	10	5	17.28	1	0.194	893.7
	5	20	1	30	0.855	0.039	1519.05
	5	20	3	4.21	0.535	0.0082	221.61
	5	20	5	14.6	0.959	0.179	758.13
6	6	3	1	65.33	0.319	0.046	3275.18
	6	3	3	64.66	0.64	0.095	3250.55
	6	3	5	64.83	0.64	0.048	3256.7

	6	10	1	38.37	0.64	0.033	1932.95
	6	10	3	32.16	0.96	0.046	1629.5
	6	10	5	28.35	0.96	0.0796	1440.68
	6	20	1	31.33	0.746	0.03	1582.92
	6	20	3	16.67	1.06	0.526	881
	6	20	5	0.67	1.6	0.618	96.4
7	7	3	1	58.43	0.426	0.0117	2930.605
	7	3	3	56.57	0.427	0.218	2847.94
	7	3	5	50	0.639	0.056	2515.58
	7	10	1	55.57	0.747	0.044	2795.64
	7	10	3	40.28	0.722	0.082	2032.54
	7	10	5	32.57	0.722	0.1096	1648.42
	7	20	1	27.42	1.28	0.08	1400.6
	7	20	3	39.14	2.59	0.019	2009.75
	7	20	5	13	1.4	0.0967	682.835
8	8	3	1	49.25	0.64	0.1	2480.3
	8	3	3	50.25	0.64	0.0034	2525.47
	8	3	5	53.62	0.64	0.038	2695.7
	8	10	1	53.5	0.64	0.045	2690.05
	8	10	3	33.25	0.74	0.107	1682.65
	8	10	5	41.75	0.64	0.217	2111.15
	8	20	1	20.37	0.959	0.64	1069.38
	8	20	3	27.37	0.855	0.207	1389.95
	8	20	5	31	1.064	0.013	1571.93

```
#include <Servo.h>
Servo firstESC;
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#if I2CDEV_IMPLEMENTATION ==
I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif
MPU6050 mpu;
#define OUTPUT_READABLE_YAWPITCHROLL
#define LED_PIN 13
bool blinkState = false;
bool dmpReady = false;
uint8_t mpuIntStatus;
uint8_t devStatus;
uint16_t packetSize;
uint16_t fifoCount;
uint8_t fifoBuffer[64];
Quaternion q;           // [w, x, y, z]
VectorInt16 aa;         // [x, y, z]
VectorInt16 aaReal;     // [x, y, z]
VectorInt16 aaWorld;    // [x, y, z]
VectorFloat gravity;    // [x, y, z]
float euler[3];         // [psi, theta, phi]
float ypr[3];           // [yaw, pitch, roll]
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00,
'\r', '\n' };



//
============================================================
========
// ===               INTERRUPT DETECTION ROUTINE
===
//
============================================================
========

volatile bool mpuInterrupt = false;
void dmpDataReady() {
  mpuInterrupt = true;
}
```

```
        //
================================================================
========
        // ===                INITIAL SETUP                ===
        //
================================================================
========

        void setup() {
          firstESC.attach(9);
          int PWM = 1000;
          firstESC.writeMicroseconds(PWM);
          // join I2C bus (I2Cdev library doesn't do this automatically)
          #if I2CDEV_IMPLEMENTATION ==
I2CDEV_ARDUINO_WIRE
            Wire.begin();
            TWBR = 24;
          #elif I2CDEV_IMPLEMENTATION ==
I2CDEV_BUILTIN_FASTWIRE
            Fastwire::setup(400, true);
          #endif

          Serial.begin(9600);
          while (!Serial);


          devStatus = mpu.dmpInitialize();

          mpu.setXGyroOffset(220);
          mpu.setYGyroOffset(76);
          mpu.setZGyroOffset(-85);
          mpu.setZAccelOffset(1788); // 1688 factory default for my test
chip


          if (devStatus == 0) {
                mpu.setDMPEnabled(true);

            // enable Arduino interrupt detection
```

```
                //  Serial.println(F("Enabling interrupt detection (Arduino
external interrupt 0)..."));
                  attachInterrupt(0, dmpDataReady, RISING);
                  mpuIntStatus = mpu.getIntStatus();

                  // set our DMP Ready flag so the main loop() function knows
it's okay to use it
                 // Serial.println(F("DMP ready! Waiting for first
interrupt..."));
                  dmpReady = true;

                  // get expected DMP packet size for later comparison
                  packetSize = mpu.dmpGetFIFOPacketSize();
               } else {
                  // ERROR!
                  // 1 = initial memory load failed
                  // 2 = DMP configuration updates failed
                  // (if it's going to break, usually the code will be 1)
                 // Serial.print(F("DMP Initialization failed (code "));
                 // Serial.print(devStatus);
                  Serial.println(F(")"));
               }//

               // configure LED for output
               pinMode(LED_PIN, OUTPUT);
            }
void NN () {
   //
   M11 = 1/(1+exp(-((simpan_error*14,0005)-(13,9995))));
   M12 = 1/(1+exp(-((simpan_error*-14)+(-10,8889))));
   M13 = 1/(1+exp(-((simpan_error*-14)+(7,7778))));
   M14 = 1/(1+exp(-((simpan_error*14)+(-4,6664))));
   M15 = 1/(1+exp(-((simpan_error*14,0045)+(-1,5246))));
   M16 = 1/(1+exp(-((simpan_error*-13,9906)+(-1,6621))));
   M17 = 1/(1+exp(-((simpan_error*13,9976)+(4,673))));
   M18 = 1/(1+exp(-((simpan_error*-0,14083)-(7,7653))));
   M19 = 1/(1+exp(-((simpan_error*-13,968)+(-10,9099))));
   M110 = 1/(1+exp(-((simpan_error*13,9474)+(14,0525))));


  Mtotal1 = (M11*-0,45385)+(M12*0,4786)+(M13*-
0,10432)+(M14*0,065764)+(M15*-0,46882)+(M16*-0,10037)+(M17*-
0,81076)+(M18*0,3526)+(M19*0,65113)+(M110*0,37804)+1,2395;
```

```
  Mtotal2 = (M11*-0,77157)+(M12*-0,5852)+(M13*-0,33223)+(M14*-
0,35966)+(M15*-0,82855)+(M16*-
0,46613)+(M17*0,39612)+(M18*0,38734)+(M19*0,28854)+(M110*-
0,11847)-1,3369;


   }
          //
============================================================
========
          // ===              MAIN PROGRAM LOOP              ===
          //
============================================================
========

          void loop() {
            unsigned long currentMillis = millis();
            // if programming failed, don't try to do anything
            if (!dmpReady) return;

            // wait for MPU interrupt or extra packet(s) available
            while (!mpuInterrupt && fifoCount < packetSize) {
               // other program behavior stuff here
               // .
               // .
               // .
               // if you are really paranoid you can frequently test in
between other
               // stuff to see if mpuInterrupt is true, and if so, "break;" from
the
               // while() loop to immediately process the MPU data
               // .
               // .
               // .
            }

            // reset interrupt flag and get INT_STATUS byte
            mpuInterrupt = false;
            mpuIntStatus = mpu.getIntStatus();

            // get current FIFO count
            fifoCount = mpu.getFIFOCount();
```

```
            // check for overflow (this should never happen unless our code
is too inefficient)
            if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
               // reset so we can continue cleanly
               mpu.resetFIFO();
              // Serial.println(F("FIFO overflow!"));

            // otherwise, check for DMP data ready interrupt (this should
happen frequently)
            } else if (mpuIntStatus & 0x02) {
               // wait for correct available data length, should be a VERY
short wait
               while (fifoCount < packetSize) fifoCount =
mpu.getFIFOCount();

               // read a packet from FIFO
               mpu.getFIFOBytes(fifoBuffer, packetSize);

               // track FIFO count here in case there is > 1 packet available
               // (this lets us immediately read more without waiting for an
interrupt)
               fifoCount -= packetSize;
               #ifdef OUTPUT_READABLE_QUATERNION
                  // display quaternion values in easy matrix form: w x y z
                  mpu.dmpGetQuaternion(&q, fifoBuffer);
                  Serial.print("quat\t");
                  Serial.print(q.w);
                  Serial.print("\t");
                  Serial.print(q.x);
                  Serial.print("\t");
                  Serial.print(q.y);
                  Serial.print("\t");
                  Serial.println(q.z);
               #endif
               #ifdef OUTPUT_READABLE_EULER
                  // display Euler angles in degrees
                  mpu.dmpGetQuaternion(&q, fifoBuffer);
                  mpu.dmpGetEuler(euler, &q);
                  Serial.print("euler\t");
                  Serial.print(euler[0] * 180/M_PI);
                  Serial.print("\t");
                  Serial.print(euler[1] * 180/M_PI);
```

```
                Serial.print("\t");
                Serial.println(euler[2] * 180/M_PI);
              #endif
              // subrutin yang akan diproses
              #ifdef OUTPUT_READABLE_YAWPITCHROLL
                // display Euler angles in degrees
                mpu.dmpGetQuaternion(&q, fifoBuffer);
                mpu.dmpGetGravity(&gravity, &q);
                mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
                //Serial.print("ypr\t");
                //Serial.print(ypr[0] * 180/M_PI);
                //Serial.print("\t");
                //Serial.print(ypr[1] * 180/M_PI);
                //Serial.print("\t");
                //Serial.print("Sudut Roll,millis : ");
               int PWM=1000; // safety lock brushless
                //delay(100);
               // PWM=1100;
               // firstESC.writeMicroseconds(PWM);
                float sudutRoll=(abs((ypr[2] * 180/M_PI))) ;
                if (sudutRoll>=0 &&
sudutRoll<=1){PWM=1100;firstESC.writeMicroseconds(PWM);}
                if (sudutRoll>=2 &&
sudutRoll<=3){PWM=1300;firstESC.writeMicroseconds(PWM);}//
                if (sudutRoll>=3 &&
sudutRoll<=4){PWM=1500;firstESC.writeMicroseconds(PWM);}
                if (sudutRoll>=5 &&
sudutRoll<=6){PWM=1600;firstESC.writeMicroseconds(PWM);}//
                if (sudutRoll>=7 &&
sudutRoll<=8){PWM=1800;firstESC.writeMicroseconds(PWM);}
                if (sudutRoll>=9 &&
sudutRoll<=10){PWM=1800;firstESC.writeMicroseconds(PWM);}
                if
(sudutRoll>=10){PWM=1900;firstESC.writeMicroseconds(PWM);}// PID
ONLY
                Serial.print(sudutRoll);
                Serial.print("|");
                Serial.print(currentMillis);
                Serial.print("|");
                Serial.print((currentMillis)/1000);
                Serial.print("|");
                Serial.println(PWM);
```

```
                #endif
                #ifdef OUTPUT_READABLE_REALACCEL
                    // display real acceleration, adjusted to remove gravity
                    mpu.dmpGetQuaternion(&q, fifoBuffer);
                    mpu.dmpGetAccel(&aa, fifoBuffer);
                    mpu.dmpGetGravity(&gravity, &q);
                    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
                    Serial.print("areal\t");
                    Serial.print(aaReal.x);
                    Serial.print("\t");
                    Serial.print(aaReal.y);
                    Serial.print("\t");
                    Serial.println(aaReal.z);
                #endif
                #ifdef OUTPUT_READABLE_WORLDACCEL
                    // display initial world-frame acceleration, adjusted to
remove gravity
                    // and rotated based on known orientation from quaternion
                    mpu.dmpGetQuaternion(&q, fifoBuffer);
                    mpu.dmpGetAccel(&aa, fifoBuffer);
                    mpu.dmpGetGravity(&gravity, &q);
                    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
                    mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal,
&q);
                    Serial.print("aworld\t");
                    Serial.print(aaWorld.x);
                    Serial.print("\t");
                    Serial.print(aaWorld.y);
                    Serial.print("\t");
                    Serial.println(aaWorld.z);
                #endif
                    #ifdef OUTPUT_TEAPOT
                    // display quaternion values in InvenSense Teapot demo
format:
                    teapotPacket[2] = fifoBuffer[0];
                    teapotPacket[3] = fifoBuffer[1];
                    teapotPacket[4] = fifoBuffer[4];
                    teapotPacket[5] = fifoBuffer[5];
                    teapotPacket[6] = fifoBuffer[8];
                    teapotPacket[7] = fifoBuffer[9];
                    teapotPacket[8] = fifoBuffer[12];
                    teapotPacket[9] = fifoBuffer[13];
                    Serial.write(teapotPacket, 14);
```

```
                teapotPacket[11]++; // packetCount, loops at 0xFF on
purpose
                #endif

                // blink LED to indicate activity
                blinkState = !blinkState;
                digitalWrite(LED_PIN, blinkState);
            }
        }
```

[Halaman Ini Sengaja Dikosongkan

BIODATA PENULIS

Penulis dilahirkan Surakarta, 28 Maret 1994, merupakan anak pertama dari dua bersaudara pasangan Sri Baroto Susetyohadi dan Dewi PPS. Penulis menempuh dan mengikuti jenjang pendidikan formal di TK Puri Mandiri Surakarta, SD Negeri Mangkubumen Lor No.15 Surakarta, SMP Negeri 1 Surakarta dan SMA Negeri 1 Surakarta. Setelah lulus dari SMA Negeri 1 Surakarta dan mengikuti tes tulis SBMPTN, Penulis diterima di Jurusan Teknik Mesin-FTI ITS dan terdaftar dengan nomor regristasi NRP 02111240000122.

Di Jurusan Teknik Mesin ini Penulis mengambil bidang studi Manufaktur di Laboratorium Perancangan dan Pengembangan Produk. Penulis aktif dalam berbagai kegiatan kemahasiswaan pada tingkat fakultas sebagai Pemandu LKMM FTI dan tingkat institut yang tergabung dalam Badan Eksekutif Mahasiswa ITS. Untuk ranah keprofesian dan akademik, Penulis mengikuti beberapa pelatihan yang tersertifikasi seperti Lean Six Sigma, ISO 50001:2011 dan Matlab. Penulis bisa dihubungi melalui email mario.muhammadh@gmail.com.

[Halaman Ini Sengaja Dikosongkan]