SISTEM KENDALI JARAK TEMPUH *QUADCOPTER* MENGGUNAKAN METODE *PROPORTIONAL INTEGRAL DERIVATIVE*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Teknik

Disusun oleh:
Enno Roscitra Oktaria
NIM: 145150300111119

PROGRAM STUDI TEKNIK KOMPUTER
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

SISTEM KENDALI JARAK TEMPUH *QUADCOPTER* MENGGUNAKAN METODE *PROPORTIONAL INTEGRAL DERIVATIVE*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Teknik

Disusun Oleh :
Enno Roscitra Oktaria
NIM: 145150300111119

Skripsi ini telah diuji dan dinyatakan lulus pada
3 Agustus 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Gembong Edhi Setyawan, S.T, M.T	Wijaya Kurniawan, S.T, M.T
NIP. 201208 761201 1 001	NIP. 19820125 201504 1 002

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T., Ph.D.
NIP. 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 3 Agustus 2018

Enno Roscitra Oktaria

NIM: 145150300111119

KATA PENGANTAR

Assalamu'alaikum Warohmatullohi Wabarokatuh

Alhamdulillah, puji syukur kehadirat Allah SWT yang telah melimpahkan rahmat, taufik serta hidayah-Nya, sehingga penulis dapat menyelesaikan penelitian yang berjudul "Sistem Kendali Jarak Tempuh *Quadcopter* Menggunakan Metode *Proportional Integral Derivative*". Penulis bersyukur atas kontribusi dari berbagai pihak yang telah membantu penulis dalam menyelesaikan penelitian dan penyusunan laporan ini. Dalam kesempatan ini penulis ingin menyampaikan ucapan terima kasih kepada:

1. Allah Yang Maha Esa, dengan ridho dan rahmat-Nya yang selalu menyertai penulis.
2. Bapak Darno selaku ayah dan Ibu Martinik selaku ibu dari penulis yang selalu memberi semangat melalui kasih sayang yang tulus tanpa pernah menuntut apapun dan selalu mendukung setiap langkah penulis.
3. Nony Pretty Reslica dan Denia Mutiara Deslilia selaku kakak dan adik dari penulis yang selalu menjadi acuan penulis untuk menjadi lebih baik.
4. Bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D. selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya Malang.
5. Tri Astoto Kurniawan, S.T., M.T., Ph.D. selaku ketua jurusan Teknik Informatika.
6. Bapak Heru Nurwarsito, Ir., M.Kom. selaku Wakil Dekan I Bidang Akademik Fakultas Ilmu Komputer Universitas Brawijaya Malang.
7. Bapak Sabriansyah Rizqika Akbar, S.T, M.Eng. selaku Ketua Program Studi Teknik Komputer Universitas Brawijaya Malang.
8. Gembong Edhi Setyawan, S.T, M.T dan Wijaya Kurniawan, S.T, M.T selaku dosen pembimbing yang telah memberikan dukungan, saran dan ilmu dalam membimbing penulis hingga pengerjaan skripsi ini selesai.
9. Muliyahati Sutejo, Yohana Kristinawati, Bunga Boru Hasian, Olivia Rumiris, Riyyan Royhan selaku rekan dekat penulis yang selalu mendmpingi, mendoakan, membantu dan menjadi cermin untuk penulis.
10. Cindy Lilian, Haqqi Rizqi, Mesra Tamsar, Hendra selaku rekan persejuangan penulis di tim penelitian *Quadcopter*. Serta Ayang Setiyo Putri, Achmad Baecuni, Dimas Angger, Sabita Wildani, Yusril Dewantara, Andyan Bina, Sabitha Wildani, Amroy Casro, Fajar Miftakhul selaku senior yang telah memberikan ilmu terkait penelitian yang pernah dilakukan.
11. Bramantyo Ardi, Okke Rizki, Dimas Bagus, Gusti Arief selaku rekan penulis yang telah membantu penulis melakukan pengujian sistem.

12. Seluruh civitas akademika Informatika Universitas Brawijaya dan teman-teman Teknik Komputer Angkatan 2014 khususnya teman-teman Sedulur Tekkom E 2014 yang telah banyak memberi bantuan dan dukungan selama penulis menempuh studi di Teknik Komputer Universitas Brawijaya dan selama penyelesaian skripsi ini.
13. Dan kepada seluruh pihak yang tidak pernah bosan mengatakan pasti bisa kepada penulis serta yang selalu mendukung dalam doa untuk kelancaran pengerjaan tugas akhir ini. Sebuah kata ucapan penyemangat dan pertanyaan kapan lulus menjadi energi semangat untuk penulis. Mohon maaf penulis tidak dapat menyebutkan seluruh pihak satu persatu.

Penulis menyadari bahwa laporan ini masih jauh dari kata sempurna dan masih memiliki berbagai macam kekurangan. Oleh karena itu penulis mengharapkan kritik dan saran yang membangun, agar ke depannya penulis dapat menjadi lebih baik lagi. Semoga laporan skripsi ini dapat memberi manfaat bagi perkembangan ilmu pengetahuan di kemudian hari.

Malang, 3 Agustus 2018

Penulis

Roscitra12@gmail.com

ABSTRAK

Quadcopter merupakan salah satu robot *Unmanned Aerial Vehicle* (UAV) yang memiliki kemampuan manuver ke segala arah dan fleksibel untuk menjelajahi area yang sempit. Kemampuan tersebut dapat dioptimalkan manfaatnya dengan cara membangun sistem navigasi otomatis pada *quadcopter*. Salah satunya yaitu merancang sistem kendali jarak tempuh *quadcopter* menggunakan metode PID (*Proportional Integral Derivative*). *Tuning* parameter PID menggunakan metode osilasi *ziegler nichols*. Respon sistem berosilasi dengan stabil saat nilai $Ku = 0{,}07$ dan $Pu = 0{,}092$, sehingga pengendalian pergerakan dalam sudut *pitch* dan *roll* memperoleh nilai $Kp = 0{,}042$; $Ki = 0{,}046$; dan $Kd = 0{,}0115$. Dari pengamatan pengujian, menunjukan bahwa sistem kendali menggunakan PID menghasilkan tingkat kesalahan sebesar 4,99% dan waktu yang dibutuhkan sistem untuk memasuki $\pm 2\%$ dari keadaan *steady state* atau *settling time* sebesar 0,4145 detik. Tingkat kesalahan dan *settling time* diperoleh berdasarkan rata-rata hasil pada setiap pengujian yang telah dilakukan.

Kata kunci: Sistem Kendali, Sistem Navigasi, PID, *Quadcopter*, Jarak Tempuh, *Ziegler Nichols*

ABSTRACT

The Quadcopter is one of Unmanned Aerial Vehicle (UAV) robots that has the ability to maneuver in various directions and flexible to explore a narrow area. These capabilities can be optimized for the benefits by building automated navigation systems on the quadcopter. One of them is designed a mileage control system of quadcopter using a PID (Proportional Integral Derivative) method. Tuning of PID parameters using the Ziegler Nichols oscillation method. The response of the system oscillates stably when the value of Ku = 0.07 and Pu = 0.092, so that the movement control in the pitch and roll angle obtained Kp = 0.042; Ki = 0.046; and Kd = 0.0115. From the observation, testing showed that the control system using PID resulted in an error rate of 4.99% and the time required for the system to enter ± 2% of the state steady state or settling time of 0.4145 second. The error rate and settling time are obtained based on the average of results on each test that has been done.

Keywords: Controller, Navigation System, PID, Quadcopter, Distance, Ziegler Nichlos

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar belakang

Quadcopter merupakan salah satu jenis robot *Unmanned Aerial Vehicle* (UAV) yang memiliki struktur mekanis terdiri dari empat *rotor* pada empat ujung persimpangan tempat baterai dipasang (Piskorski, Brulez, Eline, & D'Haeyer, 2012). Dengan empat *rotor* tersebut *quadcopter* memiliki kemampuan manuver kesegala arah dengan fleksibel. Kemampuan *quadcopter* tersebut dimanfaatkan oleh manusia untuk memudahkan pekerjaan dalam lingkungan yang cukup berbahaya, seperti pengamatan terhadap kebakaran hutan yang dilakukan oleh Caballero, Martonez-de-Dios dan Maza (2010).

Quadcopter membutuhkan sistem navigasi otomatis untuk mengoptimalkan kemampuan manuvernya. Sistem navigasi otomatis pada *quadcopter* didukung oleh sensor *Global Positioning System* (GPS) dan *Inertial Measurement Unit* (IMU). Namun, sistem dengan menggunakan GPS memiliki masalah pada ketidak konsistenan konektivitas yang disebabkan oleh medan atau lingkungan yang dilalui oleh *quadcopter* (Kurniawan, Mutiara, & Hapsari, 2015). Sehingga sistem tersebut perlu diperbaiki dengan cara membangun sistem kendali jarak tempuh *quadcopter*. Pengendalian ini termasuk pengendalian otomatis berdasarkan jarak tempuh yang terdiri dari gerak maju, mundur, ke kiri, dan ke kanan.

Dalam menentukan metode untuk sistem kendali jarak tempuh *quadcopter* merupakan suatu permasalahan tersendiri. Penelitian tentang sistem kendali *quadcopter* pernah dilakukan oleh Setyawan, Setiawan dan Kurniawan (2015) menggunakan metode PID (*Proportional, Integral, Derivative*) dengan parameter ketinggian. Penelitian lain yang menggunakan metode PID juga pernah dilakukan oleh Lwin dan Tun (2014) dengan menambahkan metode *Kalman Filter* untuk simulasi *forward moving control* dan *vertical moving control* pada *quadcopter*. Hasil dari penelitian-penelitian tersebut menunjukan bahwa PID memiliki respon yang cepat sehingga dapat diterapkan dalam sistem *quadcopter*.

Berdasarkan permasalahan konektifitas GPS dan penelitian yang dilakukan oleh Setyawan, Setiawan dan Kurniawan (2015) maka pada penelitian ini akan membangun sistem kendali menggunakan metode PID dengan parameter jarak tempuh *quadcopter*. PID merupakan kontroler untuk menentukan presisi suatu sistem instrumentasi dengan karakteristik adanya umpan balik pada sistem. *Output* PID digunakan untuk mengatur nilai *command* kecepatan linear *x* dan kecepatan linear *y* pada *quadcopter* melalui ROS (*Robot Operating System*). Sedangkan *setpoint* pada pengendali ini yaitu nilai jarak tempuh yang diinginkan. Untuk *tuning* parameter PID pada penelitian ini menggunakan metode *ziegler nichols* dengan osilasi. Metode *ziegler nichols* adalah metode *tuning* parameter PID secara eksperimental. Metode ini didasarkan pada reaksi *plant* yang dikenai suatu perubahan. Sistem kendali ini akan diimplementasikan pada *quadcopter* jenis *Parrot AR Drone* 2.0.

1.2 Rumusan masalah

Berdasarkan latar belakang di atas, maka rumusan masalah dari penelitian ini adalah sebagai berikut.

1. Berapa nilai parameter PID yang tepat untuk mengontrol jarak tempuh *pitch* dan *roll* dari *quadcopter*?
2. Seberapa tepat pengendalian *quadcopter* dengan menggunakan metode PID?
3. Berapa waktu yang dibutuhkan untuk mencapai jarak tempuh yang diinginkan?

1.3 Tujuan

Sesuai dengan rumusan masalah yang telah disebutkan, tujuan dari penelitian ini adalah sebagai berikut.

1. Mengetahui nilai parameter PID yang tepat untuk mengontrol pergerakan *quadcopter* hingga mencapai jarak tempuh yang diinginkan.
2. Menguji tingkat akurasi sistem kendali jarak tempuh *quadcopter* yang dihasilkan oleh kontroler PID.
3. Mengetahui waktu yang dibutuhkan untuk mencapai jarak tempuh yang diinginkan.

1.4 Manfaat

Manfaat dari penelitian ini antara lain sebagai berikut.

1. Memberikan pengetahuan dan wawasan kepada pembaca terkait sistem *quadcopter*, kontroler PID, *tuning* parameter PID dan ROS yang diterapkan pada penelitian ini.
2. Memberikan solusi untuk sistem navigasi otomatis pada *quadcopter* dalam pengendalian jarak tempuh *quadcopter*.
3. Dapat digunakan sebagai referensi untuk perkembangan penelitian selanjutnya mengenai sistem kendali untuk *quadcopter*.

1.5 Batasan masalah

Batasan masalah ditentukan agar permasalahan yang dirumuskan dapat lebih terfokus dan tidak meluas. Pada penelitian ini, batasan masalah tersebut antara lain:

1. Sistem diimplementasikan pada *quadcopter* jenis *Parrot AR. Drone* 2.0.
2. Sistem kendali *quadcopter* menggunakan metode PID dengan *tuning* parameter *ziegler nichols*.
3. *Tuning* parameter PID dilakukan hanya satu kali dengan *setpoint* 10 pada gerak *pitch* untuk pengendali jarak tempuh gerak *pitch* dan *roll*.
4. Pengujian dilakukan dalam ruangan atau gedung.
5. Satuan yang digunakan untuk jarak tempuh adalah desimeter atau dm.
6. Pengujian dilakukan pada masing-masing gerakan *quadcopter* dengan *setpoint* sebesar 10 dm, 15 dm dan 20 dm.

7. Estimasi sistem dimulai ketika *quadcopter hover* dengan stabil.

1.6 Sistematika pembahasan

Sistematika penulisan dalam penelitian ini sebagai berikut.

BAB 1 PENDAHULUAN

Bab ini berisi tentang Latar Belakang, Rumusan Masalah, Batasan Masalah, Tujuan Penulisan, Manfaat Penulisan, dan Sistematika Penulisan.

BAB 2 TINJAUAN PUSTAKA DAN DASAR TEORI

Menguraikan tinjauan pustaka dan dasar teori yang mendasari tentang hasil penelitian yang disajikan dalam pustaka dan menghubungkannya dengan masalah penelitian yang sedang diteliti.

BAB 3 METODE PENELITIAN

Menguraikan tentang metode dan langkah kerja yang terdiri dari studi literatur, analisis kebutuhan sistem, perancangan sistem, implementasi dan analisis serta pengambilan kesimpulan.

BAB 4 ANALISIS KEBUTUHAN

Pada bab ini berisi penjelasan mengenai kebutuhan-kebutuhan yang terkait dalam penelitian, penjelasan pada analisis kebutuhan ini nantinya menjadi acuan untuk menjawab kebutuhan yang sesuai dengan kesepakatan penelitian yaitu kebutuhan fungsionalitas dan fungsi non fungsionalitas.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

Pada bab ini akan menguraikan proses implementasi dari dasar teori yang telah dipelajari sesuai analisis dan perancangan sistem.

BAB 6 PENGUJIAN DAN ANALISIS

Pada bab ini memuat hasil pengujian dan analisis terhadap sistem yang telah direalisasikan. Pada tahap ini juga disebut inti daripada penelitian, karena hasil penelitian bisa dilihat saat implementasi dari penelitian yang sudah dilaksanakan sebelumnya.

BAB 7 PENUTUP

Bab ini berisi kesimpulan atas penelitian yang telah dilakukan, dan memberikan saran untuk pengembangan sistem lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Tinjauan Pustaka

Penelitian yang dilakukan oleh Kurniawan, Mutiara dan Hapsari (2015) menyatakan semakin jauh *quadcopter* terbang akan semakin sulit untuk dikendalikan dan dipantau posisinya. Maka diperlukan bantuan GPS pada AR Drone 2.0 untuk pengendalian jarak jauh serta terbang otomatis. Pengiriman informasi GPS berupa *text* yang berisi data *latitude* dan *longtitude*. Hasil penelitian tersebut menyatakan bahwa sistem GPS sangat dipengaruhi oleh faktor cuaca dan keadaan lingkungan. Sistem GPS akan lebih cepat terkoneksi dengan satelit jika berada di luar ruangan bahkan dapat mencapai batas maksimal satelit yaitu 12 satelit, namun saat di dalam ruangan sistem GPS membutuhkan waktu yang lebih lama untuk terkoneksi dengan satelit bahkan tidak dapat terkoneksi dengan satelit.

Penelitian yang dilakukan oleh Setyawan, Setiawan dan Kurniawan (2015) merancang sistem navigasi otomatis pada *quadcopter,* yaitu sistem kendali *quadcopter* berdasarkan parameter ketinggian *quadcopter* menggunakan metode PID. Penelitian tersebut memilih menggunakan PID karena kelebihan dari masing-masing tipe kontroler dapat digabungkan dan respon PID yang lebih cepat baik untuk *quadcopter*. Metode yang dilakukan untuk melakukan *tuning* parameter PID dalam penelitian tersebut adalah menggunakan metode osilasi *ziegler nichols*. Dari penelitian tersebut diperoleh ketinggian yang stabil sekitar 3,05 meter dengan waktu (*settling time*) 14,5 detik dan penelitian tersebut membuktikan bahwa metode PID dapat bekerja dengan baik untuk sistem kendali otomatis pada *quadcopter*.

Penelitian yang dilakukan oleh Lwin dan Tun (2014) meggunakan dua buah metode yang digabungkan dalam satu sistem yaitu metode PID dan *Kalman Filter*. Kontroler PID dan *Kalma Filter* digunakan untuk merancang sistem kontrol penerbangan formasi *quadcopter* untuk kecepatan dan sudut *pitch*. PID digunakan untuk mewujudkan kontrol stabilitas penerbangan *quadcopter*. Hasil simulasi menunjukan bahwa pengendali PID dan *Kalman Filter* memiliki kinerja dinamis.

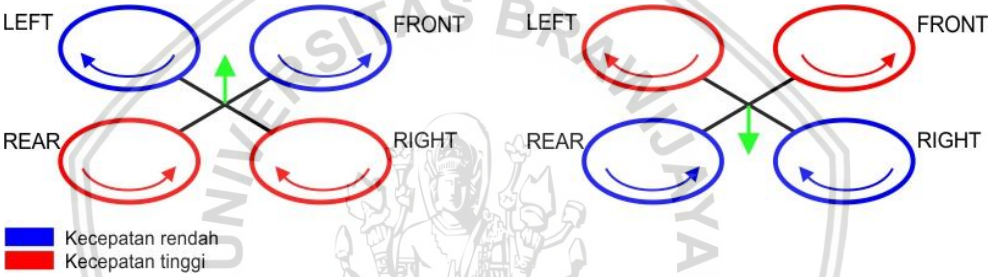
Berdasarkan penelitian-penelitian tersebut, diketahui bahwa metode PID dapat digunakan sebagai sistem kendali untuk *quadcopter*. Maka, pada penelitian ini akan menggunakan sistem kendali dengan metode PID. Namun, parameter yang dikendalikan adalah jarak tempuh *quadcopter*. Untuk *tuning* parameter PID digunakan metode *ziegler nichlos* seperti yang dilakukan oleh Setyawan, Setiawan dan Kurniawan (2015) yang menyatakan bahwa metode *tuning* tersebut didasarkan pada reaksi *plant* yang dikenai suatu perubahan.

2.2 Dasar Teori

2.2.1 *Quadcopter AR Drone* 2.0

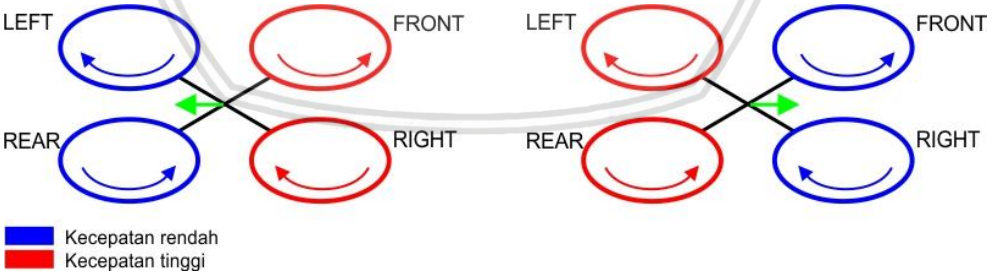
Quadcopter merupakan salah satu jenis robot UAV (*Unmanned Aerial Vehicle*) atau pesawat tanpa awak. Struktur mekanis *quadcopter* terdiri dari empat rotor yang melekat pada empat ujung persimpangan tempat baterai dan perangkat frekuensi radio dipasang (Piskorski, Brulez, Eline, & D'Haeyer, 2012). Setiap pasang rotor yang berlawanan berputar dengan cara yang sama. Satu pasang berputar searah jarum jam dan yang lainnya berlawanan arah jarum jam. Pergerakan *quadcopter* diperoleh dengan mengubah sudut *pitch*, *roll* dan *yaw* dari *quadcopter*. Berikut istilah dalam pergerakan *quadcopter*:

1. Pergerakan *quadcopter* ke arah depan dan belakang atau yang dinamakan gerakan *Pitch*. Gerakan ini diperoleh dengan mengubah kecepatan rotor pada bagian depan untuk gerakan maju dan mengubah kecepatan rotor bagian belakang untuk gerakan mundur seperti pada Gambar 2.1.



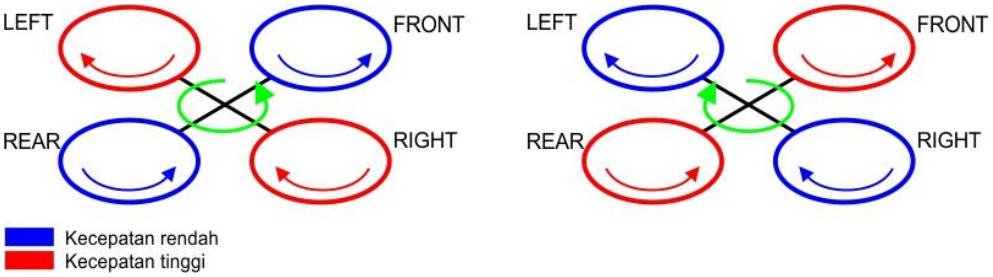
Gambar 2.1 Pergerakan *Pitch* pada *Quadcopter*

2. Pergerakan *quadcopter* ke arah kiri dan kanan atau yang dinamakan gerakan *Roll*. Gerakan ini diperoleh dengan mengubah kecepatan rotor pada bagian kiri untuk gerakan samping kiri dan mengubah kecepatan rotor bagian kanan untuk gerakan samping kanan seperti pada Gambar 2.2.



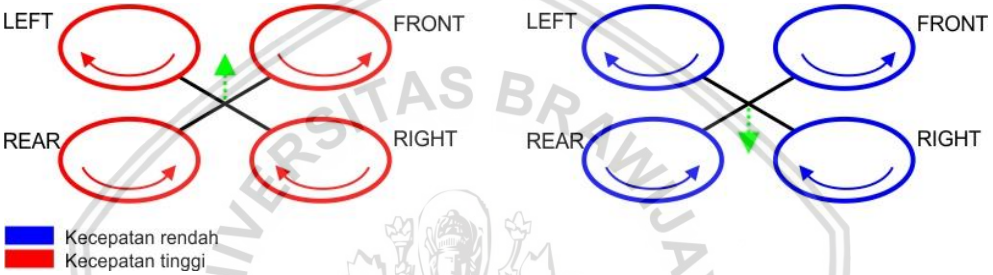
Gambar 2.2 Pergerakan *Roll* pada *Quadcopter*

3. Pergerakan *quadcopter* berputar ke kiri dan kanan atau yang dinamakan gerakan *Yaw*. Gerakan ini diperoleh dengan mengubah kecepatan rotor pada bagian kiri secara silang untuk gerakan berputar kiri dan mengubah kecepatan rotor bagian kanan secara silang untuk gerakan berputar kanan seperti pada Gambar 2.3.



Gambar 2.3 Pergerakan *Yaw* pada *Quadcopter*

4. Pergerakan *quadcopter* ke atas dan ke bawah atau yang dinamakan gerakan *Throttle*. Gerakan ini diperoleh dengan mengubah kecepatan rotor secara bersamaan seperti pada Gambar 2.4. Untuk bergerak ke atas maka rotor harus dipercepat secara bersamaan sedangkan untuk bergerak ke bawah maka rotor harus diperlambat secara bersamaan .

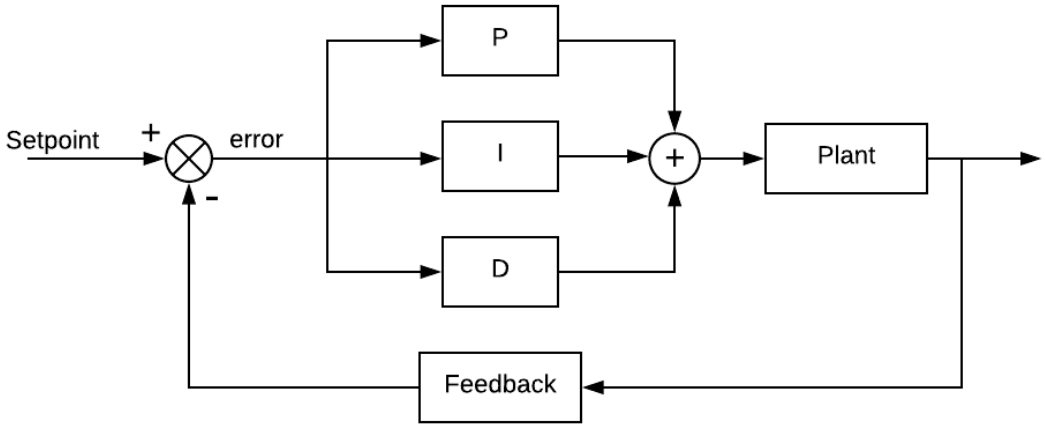


Gambar 2.4 Pergerakan *Throttle* pada *Quadcopter*

Quadcopter juga memiliki beberapa sensor yang berlokasi pada bagian tengah *quadcopter*, diantaranya adalah sensor *ultrasound* untuk mengukur ketinggian, dua kamera yang menghadap ke depan dan bawah, dan sensor *Inertial Measurements Unit* (IMU) yang digunakan untuk mengetahui posisi *pitch, roll* dan *yaw*.

2.2.2 Kontroler PID

PID (*Proportional Integral Derivative controller*) merupakan kontroler untuk menentukan presisi suatu sistem instrumentasi dengan karakteristik adanya umpan balik pada sistem tesebut. Kontroler PID terdiri dari tiga jenis komponen yaitu kontrol P (*Proportional*), I (*Integral*) dan D (*Derivative*), dengan masing-masing memiliki kelebihan dan kekurangan. Dalam implementasinya masing-masing komponen kontrol PID dapat bekerja sendiri-sendiri maupun bersamaan tergantung dari respon yang diinginkan terhadap suatu *plant.* Diagram blok dari kontroler PID dapat dilihat pada Gambar 2.5.



Gambar 2. 5 Diagram blok kontroler PID

Pada Gambar 2.5 terdapat *setpoint* yaitu masukkan *output* yang diinginkan, *feedback* yaitu umpan balik sistem, dan *error* yaitu selisih *setpoint* dengan *feedback* atau disebut dengan $e(t)$. Keluaran dari kontroler PID atau $mv(t)$ merupakan penjumlahan dari kontrol proporsional, kontrol integral dan kontrol diferensial. Adapun persamaan kontroler PID dapat dilihat pada persamaan 2.1.

$$mv(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \quad (2.1)$$

dengan :

$$K_i = K_p \times \frac{1}{T_i} \quad (2.2)$$

$$K_d = K_p \times T_d \quad (2.3)$$

Persamaan 2.2 dan 2.3 digunakan untuk menghitung nilai konstanta integral (K_i) dan konstanta diferensial (K_d) sebelum dimasukkan dalam Persamaan 2.1.

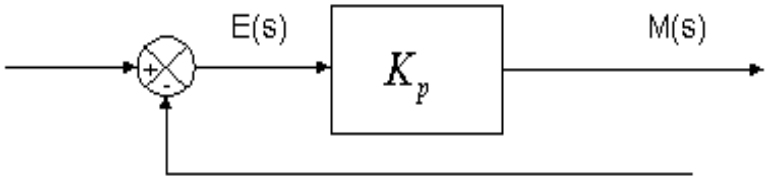
Setiap komponen kontrol PID memiliki fungsi tersendiri sebagai pengendali. Berikut fungsi dari masing-masing kontrol proporsional, kontrol integral dan kontrol diferensial.

1. Kontrol Proposional

Kontrol proposional atau K_p memiliki keluaran yang sebanding dengan besarnya sinyal kesalahan (Sharon, 1992, 19). K_p berlaku sebagai *Gain* (penguat) saja tanpa memberikan efek dinamik kepada kinerja kontroler. Penggunaan kontrol proposional memiliki berbagai keterbatasan karena sifat kontrol yang tidak dinamik ini. Walaupun demikian dalam aplikasi-aplikasi dasar yang sederhana kontrol proposional ini cukup mampu untuk memperbaiki respon transien khususnya *rise time* dan *settling time*.

Pada Gambar 2.6 menunjukkan diagram blok yang menggambarkan hubungan antara besaran *setting*, besaran aktual dengan besaran keluaran kontrol proporsional. Sinyal kesalahan (*error*) merupakan selisih antara besaran *setting* dengan besaran aktualnya. Selisih ini akan mempengaruhi kontroler, untuk mengeluarkan sinyal positif (mempercepat pencapaian harga *setting*) atau

negatif (memperlambat tercapainya harga yang diinginkan) (Chairuzzaini, Rusli, & Ariyanto, 1998).



Gambar 2.6 Diagram blok kontroler proposional

Sumber: Chairuzzaini, Rusli dan Ariyanto (1998)

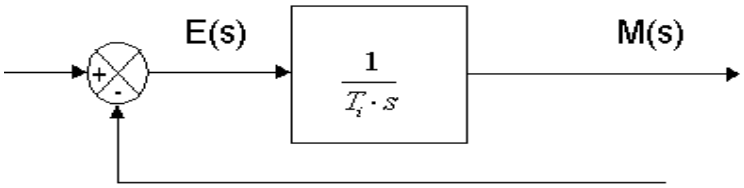
Ciri-ciri kontrol proposional harus diperhatikan ketika kontroler tersebut diterapkan pada suatu sistem. Secara eksperimen, pengguna kontroller proporsional harus memperhatikan ketentuan-ketentuan berikut ini:

a. Jika nilai K_p kecil, pengontrol proporsional hanya mampu melakukan koreksi kesalahan yang kecil, sehingga akan menghasilkan respon sistem yang lambat (menambah *rise time*).
b. Jika nilai K_p dinaikkan, respon sistem akan semakin cepat mencapai keadaan yang terbaik (mengurangi *rise time*).
c. Namun jika nilai K_p diperbesar sehingga mencapai harga yang berlebihan, akan mengakibatkan sistem bekerja tidak stabil, atau respon sistem akan berosilasi [Pakpahan, 1988, 193].
d. Nilai K_p dapat diatur sedemikian sehingga mengurangi *steady state error*, tetapi tidak menghilangkannya.

2. Kontrol Integral

Kontrol integral atau K_i berfungsi menghasilkan respon sistem yang memiliki *error steady state* sama dengan nol. Jika sebuah pengontrol tidak memiliki unsur integrator, pengontrol proporsional tidak mampu menjamin keluaran sistem dengan *error steady state* sama dengan nol. Kontrol integral dapat memperbaiki sekaligus menghilangkan respon *steady state*, namun pemilihan K_i yang tidak tepat dapat menyebabkan respon transien yang tinggi sehingga dapat menyebabkan ketidakstabilan sistem. Pemilihan K_i yang sangat tinggi justru dapat menyebabkan *output* berosilasi karena menambah orde sistem.

Keluaran pengontrol ini merupakan hasil penjumlahan yang terus menerus dari perubahan masukannya. Jika sinyal kesalahan tidak mengalami perubahan, maka keluaran akan menjaga keadaan seperti sebelum terjadinya perubahan masukan. Sinyal keluaran kontrol integral merupakan luas bidang yang dibentuk oleh kurva kesalahan penggerak. Sinyal keluaran akan berharga sama dengan harga sebelumnya ketika sinyal kesalahan berharga nol (Chairuzzaini, Rusli, & Ariyanto, 1998). Gambar 2.7 menunjukan diagram blok antara besaran kesalahan dengan keluaran suatu kontrol integral.



Gambar 2.7 Diagram blok kontrol integral

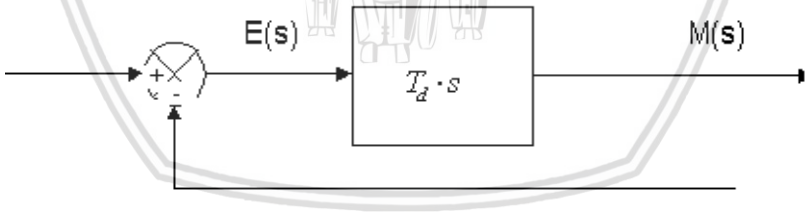
Sumber: Chairuzzaini, Rusli dan Ariyanto (1998)

Dalam penggunaanya kontrol integral memiliki beberapa karakteristik yang harus diperhatikan berikut ini:

a. Keluaran pengontrol integral membutuhkan selang waktu tertentu, sehingga pengontrol integral cenderung memperlambat respon.
b. Ketika sinyal kesalahan berharga nol, keluaran pengontrol akan bertahan pada nilai sebelumnya.
c. Jika sinyal kesalahan tidak berharga nol, keluaran akan menunjukkan kenaikan atau penurunan yang dipengaruhi oleh besarnya sinyal kesalahan dan nilai K_i.
d. Konstanta integral atau K_i yang berharga besar akan mempercepat hilangnya *offset*. Tetapi semakin besar nilai K_i akan mengakibatkan peningkatan osilasi dari sinyal keluaran pengontrol.

3. Kontrol Diferensial

Keluaran kontroler diferensial memiliki sifat seperti halnya suatu operasi derivatif. Perubahan yang mendadak pada masukan kontroler, akan mengakibatkan perubahan yang sangat besar dan cepat. Gambar 2.8 menunjukkan blok diagram yang menggambarkan hubungan antara sinyal kesalahan dengan keluaran kontroller.



Gambar 2.8 Diagram blok kontrol diferensial

Sumber: Chairuzzaini, Rusli dan Ariyanto (1998)

Gambar 2.8 menyatakan hubungan antara sinyal masukan dengan sinyal keluaran kontroler diferensial. Ketika masukannya tidak mengalami perubahan, keluaran kontroler juga tidak mengalami perubahan, sedangkan apabila sinyal masukan berubah mendadak dan menaik (berbentuk fungsi *step*), keluaran menghasilkan sinyal berbentuk impuls. Jika sinyal masukan berubah naik secara perlahan (fungsi *ramp*), keluarannya justru merupakan fungsi *step* yang besar magnitudnya sangat dipengaruhi oleh kecepatan naik dari fungsi *ramp* dan faktor konstanta diferensialnya T_d (Guterus, 1994, 8-4). Berikut karakteristik kontrol diferensial:

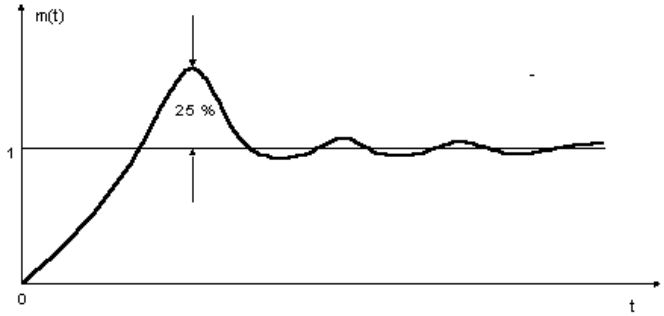
a. Kontroler ini tidak dapat menghasilkan keluaran bila tidak ada perubahan pada masukannya (berupa sinyal kesalahan).
b. Jika sinyal kesalahan berubah terhadap waktu, maka keluaran yang dihasilkan kontroler tergantung pada nilai T_d dan laju perubahan sinyal kesalahan. (Powel, 1994, 184).
c. Kontroler diferensial mempunyai suatu karakter untuk mendahului, sehingga kontroler ini dapat menghasilkan koreksi yang signifikan sebelum pembangkit kesalahan menjadi sangat besar. Jadi kontroler diferensial dapat mengantisipasi pembangkit kesalahan, memberikan aksi yang bersifat korektif, dan cenderung meningkatkan stabilitas sistem (Ogata, 1997, 240).

Berdasarkan karakteristik kontroler tersebut, kontroler diferensial umumnya dipakai untuk mempercepat respon awal suatu sistem, tetapi tidak memperkecil kesalahan pada keadaan tunaknya. Kerja kontrolller diferensial hanyalah efektif pada lingkup yang sempit, yaitu pada periode peralihan. Oleh sebab itu kontroler diferensial tidak pernah digunakan tanpa ada kontroler lain sebuah sistem (Sutrisno, 1990, 102).

2.2.3 *Tuning* Kontroler PID

Tuning kontroler PID selalu didasari atas tinjauan terhadap karakteristik yang diatur (*Plant*). Dengan demikian betapapun rumitnya suatu plant, perilaku plant tersebut harus diketahui terlebih dahulu sebelum *tuning* parameter PID itu dilakukan. Karena penyusunan model matematik plant tidak mudah, maka dikembangkan suatu metode eksperimental. Metode ini didasarkan pada reaksi plant yang dikenai suatu perubahan. Dengan menggunakan metode itu model matematik perilaku plant tidak diperlukan lagi, karena dengan menggunakan data yang berupa kurva krluaran, *tuning* kontroler PID telah dapat dilakukan. *Tuning* bertujuan untuk mendapatkan kinerja sistem sesuai spesifikasi perancangan. Ogata menyatakan hal itu sebagai alat control (*controller tuning*) (Ogata, 1997, 168, Jilid 2). Dua metode pendekatan eksperimen adalah *Ziegler-Nichols* dan metode *Quarter decay*.

Ziegler-Nichols pertama kali memperkenalkan metodenya pada tahun 1942. Metode ini memiliki dua cara, metode osilasi dan kurva reaksi. Kedua metode ditujukan untuk menghasilkan respon sistem dengan lonjakan maksimum sebesar 25%. Gambar 2.9 memperlihatkan kurva dengan lonjakan 25%.

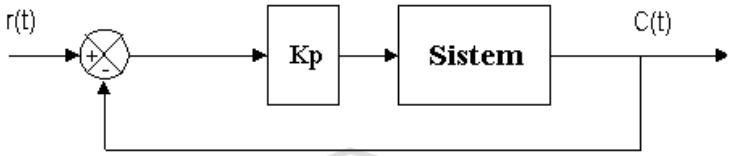


Gambar 2.9 Kurva respon tangga satuan yang memperlihatkan 25% lonjakan maksimum

Sumber: Chairuzzaini, Rusli dan Ariyanto (1998)

2.2.4 Metode Osilasi

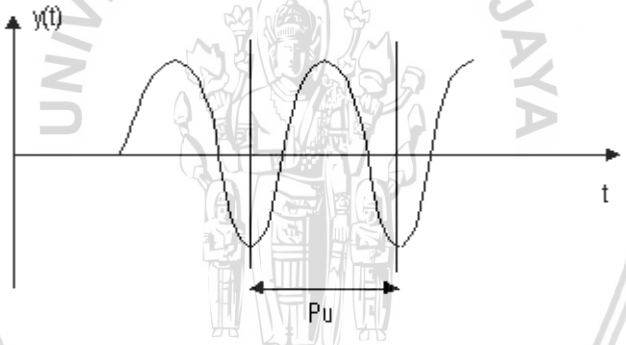
Metode ini didasarkan pada reaksi sistem untaian tertutup. Plant disusun serial dengan kontroller PID. Semula parameter parameter integrator disetel tak berhingga dan parameter diferensial disetel nol (T_i = ~ ; T_d = 0). Parameter proporsional kemudian dinaikkan bertahap. Mulai dari nol sampai mencapai harga yang mengakibatkan reaksi sistem berosilasi. Reaksi sistem harus berosilasi dengan magnitud tetap (*Sustain oscillation*) (Guterus, 1994, 9-9). Gambar 2.10 menunjukkan rangkaian untaian tertutup pada cara osilasi.



Gambar 2.10 Sistem untaian tertutup dengan alat kontrol proporsional

Sumber: Chairuzzaini, Rusli dan Ariyanto (1998)

Nilai penguatan proporsional pada saat sistem mencapai kondisi *sustain oscillation* disebut *ultimate gain* K_u. Periode dari *sustained oscillation* disebut *ultimate periode*; T_u (Perdikaris, 1991, 433).



Gambar 2.11 Kurva respon sustain *oscillation*

Sumber: Chairuzzaini, Rusli dan Ariyanto (1998)

Tuning parameter PID didasarkan terhadap kedua konstanta hasil eksperimen, K_u dan P_u. *Ziegler Nichols* menyarankan penyetelan nilai parameter K_p , T_i , dan T_d berdasarkan rumus yang diperlihatkan pada Tabel 2.1.

Tabel 2.1 *Tuning* parameter PID dengan metode osilasi

Tipe Kontroler	K_p	T_i	T_d
P	$0{,}5 \times K_u$		
PI	$0{,}45 \times K_u$	$^1/_{1{,}2}\, P_u$	
PID	$0{,}6 \times K_u$	$0{,}5 \times P_u$	$0{,}125 \times P_u$

Sumber: Chairuzzaini, Rusli dan Ariyanto (1998)

2.2.5 Kinematika Dalam Satu Dimensi

Kinematika merupakan ilmu yang mempelajari bagaimana sebuah benda dapat bergerak dan berpindah. Salah satu yang dibahas dalam kinematika adalah jarak tempuh yang disebabkan perubahan kecepatan dan perubahan waktu. Jarak tempuh sendiri adalah perubahan kedudukan suatu benda setelah bergerak selama selang waktu tertentu yang memiliki arah. Jarak tempuh hanya mempersoalkan jarak antar kedudukan awal dan akhir suatu objek. Sedangkan kecepatan adalah besaran vektor yang menunjukkan seberapa cepat benda berpindah. Kecepatan juga bisa berarti kelajuan yang mempunyai arah. Misal sebuah mobil bergerak ke timur dengan kecepatan 60 km/jam.

Bentuk matematis dari jarak tempuh, waktu dan kecepatan dinyatakan dalam persamaan sebagai berikut.

$$v = \frac{s}{t} \tag{2.4}$$

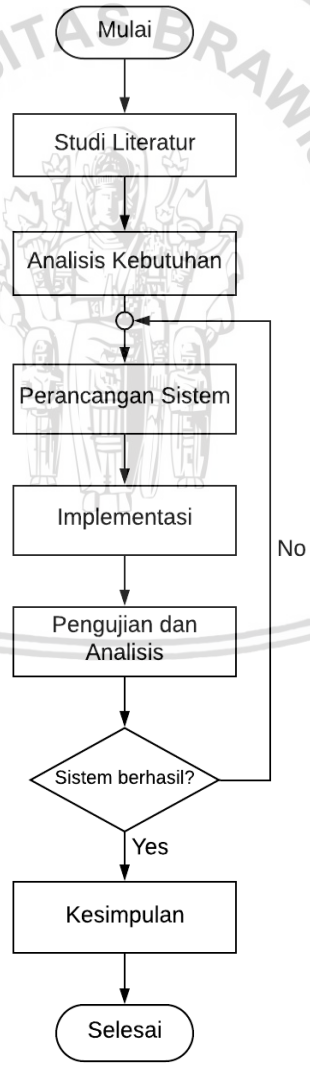
Dengan:

v: kecepatan (m/s)
s : jarak tempuh (m)
t : waktu (s)

BAB 3 METODOLOGI

3.1 Metode Penelitian

Penelitian ini menggunakan tipe implementatif yaitu perancangan sistem dengan mengacu pada topik permasalahan penelitian dan perealisasian sistem. Metode yang digunakan untuk algoritme pemecahan masalah disesuaikan dengan tujuan yang ingin dicapai dalam penelitian ini. Tahap awal dilakukan pengumpulan studi literatur sebagai dasar penelitian. Tahap selanjutnya mengumpulkan apa saja yang menjadi kebutuhan sistem. Kemudian menentukan bagaimana sistem dirancang serta diimplementasi dan tahap terakhir dilakukan pengujian terhadap sistem dengan kondisi jika sistem berhasil maka ditarik kesimpulan namun jika sistem tidak berhasil maka akan kembali ke tahap perancangan sistem. Alur metodologi penelitian ini lebih jelasnya dapat dilihat pada Gambar 3.1.



Gambar 3.1 Alur metodologi penelitian

3.2 Studi Literatur

Tahap ini akan menjelaskan tentang kajian pustaka dari penelitian terdahulu serta dasar teori yang terkait yang menjadi acuan hingga selesainya penelitian Sistem Kendali Jarak Tempuh *Quadcopter* Menggunakan Metode *Proportional Integral Derivative*. Dasar teori yang dijelaskan pada tahap ini adalah sebagai berikut.

1. *Quadcopter AR Drone* 2.0
2. Kontroler PID
3. *Tuning* parameter kontroler PID
4. Metode osilasi
5. Kinematika dalam satu dimensi

3.3 Analisis Kebutuhan

Tahap kedua dalam penelitian ini akan menentukan dan menjelaskan kebutuhan yang diperlukan dalam merancang sistem. Kebutuhan tersebut terdiri dari kebutuhan pengguna yang menjelaskan tentang apa saja yang diperlukan agar dapat menghubungkan antara pengguna dengan sistem, kebutuhan sistem membahas perangkat yang diperlukan untuk merancang sistem yang terdiri dari perangkat keras dan perangkat lunak, serta kebutuhan fungsional yaitu *quadcopter* dapat bergerak sesuai jarak tempuh ketika diberi masukkan berupa jarak dalam satuan meter dan non-fungsional yang membahas karakteristik pengguna, lingkungan operasi, asumsi dan ketergantungan, serta batasan perancangan dan implementasi.

3.4 Perancangan dan Implementasi

Pada tahap yang menentukan tipe implementatif ini bertujuan untuk merancang sistem sesuai kebutuhan fungsional. Perancangan dibagi menjadi beberapa bagian yaitu perancangan komunikasi sistem dan perancangan sistem kendali.

Tahap selanjutnya dilakukan realisasi sistem dalam bentuk implementasi. Implementasi ini didasarkan pada perancangan sistem yang telah dibuat yaitu tahap awal implementasi komunikasi untuk koneksi perangkat lunak dengan perangkat keras. Program yang dibuat berfungsi memenuhi kebutuhan fungsional. Program tersebut juga dijalankan secara langsung pada *quadcopter*.

3.5 Pengujian dan Analisis

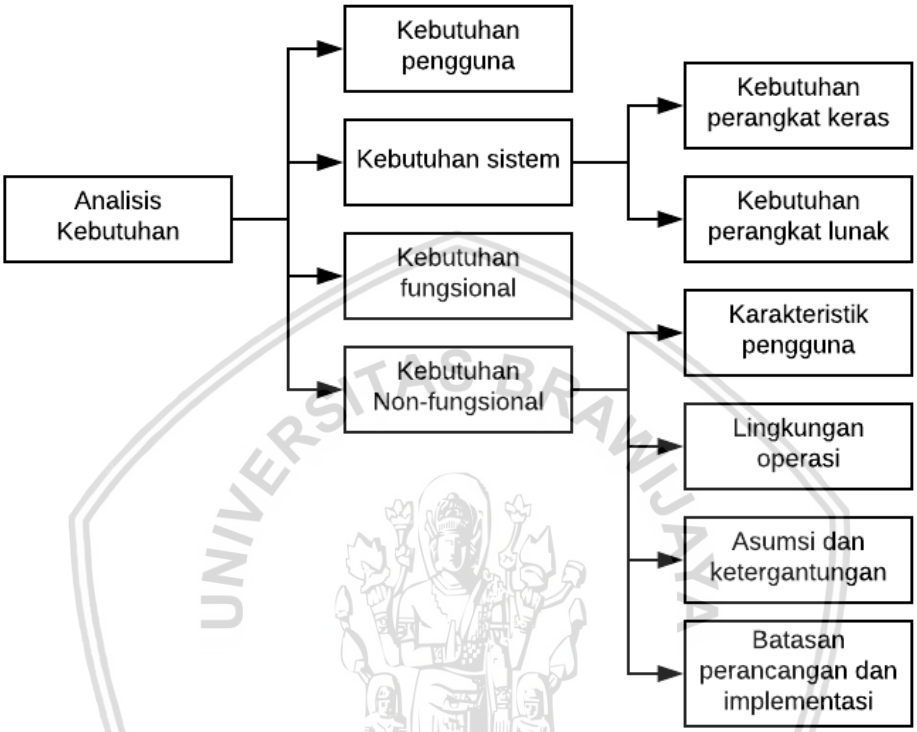
Tahap ini untuk mengetahui apakah kinerja sistem telah sesuai dengan cara melakukan pengujian sistem secara keseluruhan dan menganalis hasil uji tersebut. Hal yang dilakukan untuk skenario *tuning* serta pengujian kontroler PID dan pengujian ketepatan posisi *quadcopter*. Setelah pengujian dan analisis sistem selesai dan sistem telah sesuai maka dapat melanjutkan ke tahap penelitian berikutnya, namun jika belum sesuai maka kembali ke tahap penelitian perancangan sistem.

3.6 Kesimpulan dan Saran

Tahapan terakhir ini dilakukan setelah semua tahapan selesai dengan syarat sistem yang dirancang sudah sesuai. Kesimpulan berisi jawaban dari rumusan masalah penelitian yang berasal dari hasil pengujian dan analisis sistem. Dan terakhir penulisan saran bukan karena sebuah penelitian salah atau tidak berhasil namun bertujuan untuk pengembangan sistem yang lebih lanjut pada penelitian ini.

BAB 4 ANALISIS KEBUTUHAN

Pada tahap ini pembahasan akan dibagi dalam beberapa bagian kebutuhan yaitu kebutuhan pengguna, kebutuhan sistem, kebutuhan fungsional dan kebutuhan non-fungsional. Untuk lebih jelasnya dapat dilihat pada gambar diagram analisis kebutuhan di Gambar 4.1.



Gambar 4.1 Diagram Analisis Kebutuhan

4.1 Kebutuhan Pengguna

Untuk memudahkan pengguna dalam mengawasi sistem maka dalam kebutuhan pengguna ini dibutuhkan antarmuka pengguna dengan sistem yang menggunakan *window* yang ada pada *linux*. Informasi yang akan ditampilkan berupa nilai *output* PID, *floating point quadcopter*, *setpoint*, *feedback* sistem, *error* sistem dan kecepatan *quadcopter*, waktu dalam detik.

4.2 Kebutuhan Sistem

Kebutuhan sistem dibagi menjadi dua bagian analisis kebutuhan yaitu kebutuhan perangkat keras dan kebutuhan perangkat lunak.

4.2.1 Kebutuhan Perangkat Keras

Dalam kebutuhan perangkat keras yang dibutuhkan pada sistem ini adalah.

1. *Parrot AR Drone* 2.0

AR Drone 2.0 adalah sebuah *quadrotor* yang dikendalikan melalui sebuah *remote control* dan dikembangkan secara khusus oleh *French Company Parrot*. *AR.Drone 2.0* memiliki spesifikasi kamera yang cukup baik yaitu 720 *pixel* yang dapat mengambil gambar atau video dengan kualitas *High Definition*.

Untuk mendapatkan kinerja yang optimal *Parrot AR.Drone 2.0* telah didukung oleh sensor yang akurat dan memiliki sistem yang dapat melindungi dari getaran pada mesin *quadcopter.* Selain itu, *AR.Drone 2.0* juga didukung dengan kendali secara otomatis melalui dan bersifat *user-friendly* yaitu melalui *smarthphone* yang dapat memfasilitasi *quadcopter* untuk melakukan terbang, navigasi dan mendarat (SA, 2016). Spesifikasi yang terdapat pada *Parrot AR.Drone 2.0* ditunjukkan pada Tabel 4.1.

Tabel 4.1 Spesifikasi *Parrot Ar.Drone 2.0*

HD Video	Struktur	Home	Support	Dimensi dan berat
Live Streaming pada *tablet* dan *smartphone*	Desain dengan *figure acrobatic* paling banyak	Baterai LiPo 1000 mA/H yang dapat diisi ulang	1 GHz 32-bit *processor*	Badan *indoor* dengan dimensi : 52 x 52 x 11 cm, 420 gr.
Camera 720P *HD*, 30 *fps*	Pusat inersia yang aman dari getaran mesin	Dapat melakuka maneuver dengan performansi tinggi	1 *GB DDR*2 *RAM* pada 200 *MHz*	Badan *indoor* dengan dimensi : 32 x 25 x 11 cm, 380 gr.
Lebar sudut lensa sebesar 92˚	*Polypropylene* yang di perluas	8 *MIPS AVR Processor* tiap motor	*Wi-Fi*	
Encoding Profile H264	*Nylon plastic* dengan kualitas tinggi sebesar 30 %	Pelumasan *bearing ball* otomatis	Sensor *gyroscope* 3 sumbu dengan akurasi 2000˚/ detik	
Streaming video dengan latensi yang rendah	Bungkus *nano-hydrophobic* pada sensor *ultasonik*	Gigi pada baling-baling besi yang diperkuat	Sensor tekanan dengan akurasi 10 Pascal (800 cm diatas permukaan laut)	
Pengambilan gambar yang disimpan dengan format JPEG	Secara penuh bagian mesin dapat diperbaiki	Perdm *Gear Nylatron* pada baling- baling	*Vertical QVGA* 60 *fps camera*, Sensor *magnetometer* akurasi 6˚	
		OS *Linux* 2.6.32, *USB* 2.0 *high speed*	Perangkat elektronik tahan air	

Berdasarkan spesifikasi pada Tabel 4.1 maka peneliti menggunakan *Parrot AR Drone* 2.0 dalam penelitian ini dikarenakan mendukung sifat *open source*.

4.2.2 Kebutuhan Perangkat Lunak

1. Sistem operasi *Linux Ubuntu* versi 14.04

Ubuntu merupakan distribusi *Linux* berbasis *Debian* yang berfungsi sebagai sistem operasi bersifat *open source* dan tidak berbayar. Sasaran *Ubuntu* adalah menciptakan sistem operasi dekstop *Linux* yang mudah dipakai dan pengguna memiliki kebebasan untuk mendapatkan, mengubah dan mendistribusikan perangkat lunak sesuai dengan apa yang dibutuhkan tanpa halangan apapun. Biasanya sistem operasi ini digunakan berbagai kepentingan seperti penggunaan pribadi, namun ubuntu juga sangat populer untuk digunakan dalam *network servers*. Pada penelitian ini akan menggunakan *Ubuntu* versi 14.04 karena ROS indigo hanya mendukung *Debian Packages* versi *Saucy* (13.01) dan *Trusty* (14.04).

2. ROS

Robot Operating System adalah sebuah *framework* fleksibel serta bersifat *opensource* yang berfungsi untuk merancang perangkat lunak robot. *Framework* ini sebagai penghubung antara pengguna dengan robot *quadcopter* dengan keuntungan mudah dioperasikan. Di dalam ROS terdapat berbagai jenis *library*, *driver*, *tool* yang memiliki tujuan untuk menyelesaikan tugas dalam membangun robot yang sistemnya kompleks dalam *platform* robot. Sistem ROS terdiri dari sejumlah *node* independen, yang masing-masing berkomunikasi dengan *node* lain menggunakan model *publisher* atau *subscriber*.

3. Python

Python merupakan bahasa pemrograman interpretatif multiguna. Salah satu fitur yang tersedia pada *python* adalah sebagai bahasa pemrograman dinamis yang dilengkapi dengan manajemen memori otomatis. *Python* dapat digunakan untuk berbagai keperluan pengembangan perangkat lunak dan dapat berjalan di berbagai *platform* sistem operasi, salah satunya pada *Linux*.

4. Spyder

Spyder adalah perangkat lunak pengembangan interaktif yang kuat untuk bahasa pemrograman *python* dengan pengeditan lanjutan, pengujian interaktif, *debugging*, dan fitur introspeksi. Fitur lainnya dapat melakukan komputasi numerik berkat dukungan dari *IPython* (*interpreter Python interaktif* yang disempurnakan) dan pustaka *python* populer seperti *Numpy* (aljabar linier), *SciPy* (pemrosesan sinyal dan gambar) atau *matplotlib* (interaktif 2D / 3D *plotting*).

4.3 Kebutuhan Fungsional

Dalam kebutuhan ini akan menjelaskan bagaimana sistem yang akan dirancang nantinya hingga menghasilkan *output* sesuai yang diinginkan. Adapun kebutuhan fungsional dari sistem yang harus dipenuhi ini adalah sebagai berikut.

1. Sistem harus bisa menghasilkan nilai PID dengan melakukan *tuning* parameter PID.
2. Sistem harus bisa bergerak menuju posisi yang telah ditentukan.
3. Sistem harus bisa menampilkan nilai *output* PID, *floating point quadcopter*, *setpoint*, *feedback* sistem, *error* sistem dan kecepatan *quadcopter*, waktu dalam detik.

4.4 Kebutuhan Non-Fungsional

Kebutuhan non-fungsinal merupakan kebutuhan yang menjelaskan tentang batasan terhadao kebutuhan perancangan sistem. Berikut beberapa kebutuhan non-fungsional dari sistem ini.

4.4.1 Karakteristik Pengguna

Karakteristik pengguna digunakan untuk menjalankan sistem ini. Dengan adanya sistem ini diharapkan dapat mempermudah dalam melakukan pengendalian posisi *quadcopter* sesuai yang diinginkan.

4.4.2 Lingkungan Operasi

Persyaratan lingkungan operasi untuk menjalankan sistem ini adalah sebagai berikut.

1. S*etpoint* pada sistem ini adalah jarak tempuh *quadcopter* yang diinginkan.
2. Membutuhkan area di dalam ruangan dengan luas minimal 4 meter agar *quadcopter* dapat bergerak dengan optimal.

4.4.3 Asumsi dan Ketergantungan

1. Kontroler PID dapat berfungsi sebagai sistem kendali *quadcopter* untuk jarak tempuh yang diinginkan.
2. Sistem akan berjalan ketika komputer sudah terhubung dengan *Wi-Fi* yang dipancarkan AR Drone 2.0.
3. Sistem hanya dapat dijalankan pada komputer dengan sistem operasi Ubuntu 14.04 dan telah terpasang *ROS* di dalamnya.

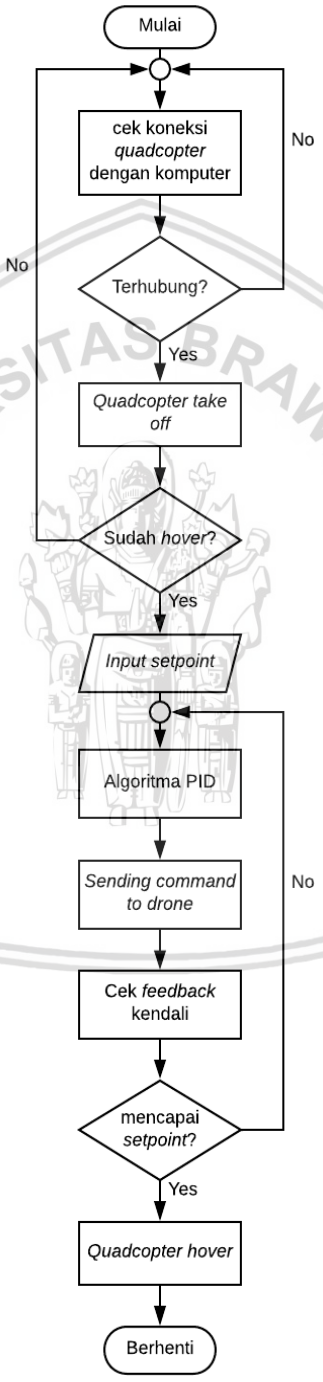
4.4.4 Batasan Perancangan dan Implementasi

Bagian ini digunakan sebagai penentu batasan sistem agar penelitian ini menjadi lebih terarah dan berjalan sesuai harapan. Adapun batasan sistem yang dimaksud adalah sebagai berikut.

1. Metode *tuning* parameter PID yang digunakan adalah *ziegler nichols* dengan cara osilasi untuk mencari nilai K_u dan P_u.
2. Gerakan *quadcopter* terdiri dari gerak maju, mundur, ke kiri dan ke kanan sesuai jarak tempuh yang telah ditentukan.
3. Sistem kendali dapat mengontrol *quadcopter* setelah perintah *takeoff* dijalankan hingga mencapai *hover*.
4. Pengujian dilakukan di dalam ruangan.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

Tahap perancangan dan implementasi sistem akan diuraikan pada bab ini. Tahapan tersebut terdiri dari komunikasi sistem, algoritme PID dan *tuning* kontroler PID. Untuk mengetahui lebih jelas tentang tahap perancangan dan implementasi sistem kendali jarak tempuh *quadcopter* ini dapat dilihat pada Gambar 5.1.



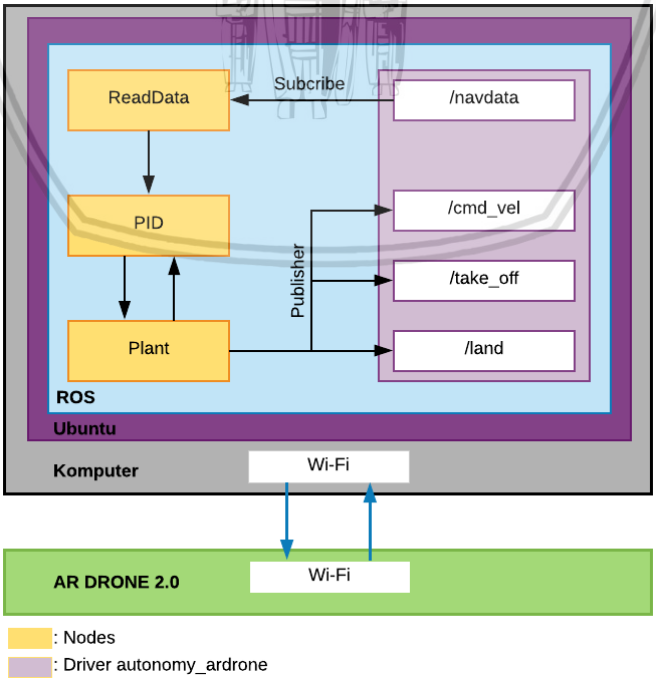
Gambar 5.1 Diagram alir perancangan sistem

Pada Gambar 5.1 diagram alir dimulai dari tahap cek koneksi antara komputer dengan *quadcopter*, tahapan tersebut akan dijelaskan pada sub bab komunikasi sistem. Setelah itu *quadcopter* diberi perintah *takeoff*, *quadcopter* harus dalam keadaan *hover* sebelum menjalankan algoritme PID. Perancangan dan Implementasi algoritme PID akan dijelaskan secara urut pada sub bab kontroler PID. Untuk *tuning* kontroler PID dalam algoritme PID dijelaskan pada sub bab *tuning* PID.

5.1 Komunikasi Sistem

5.1.1 Perancangan Komunikasi Sistem

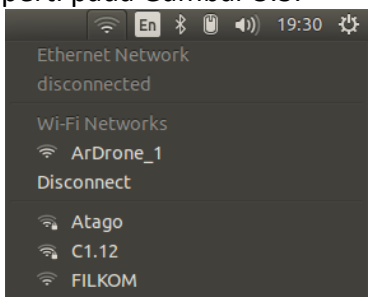
Tahap awal perancangan komunikasi sistem adalah merancang komunikasi antar *node* pada ROS. *Node* ini digunakan untuk mengurangi kesalahan sistem dan membuat sistem lebih sederhana dengan memisahkan kode program. Sebuah *node* harus memiliki nama unik yang akan digunakan untuk berkomunikasi dengan *node* lain. Pada sistem ini menggunakan tiga buah *node* yaitu *ReadData* untuk mengambil data yang terdapat pada *driver autonomy_drone*, PID untuk pemodelan matematika kontrol PID dan *plant* yaitu *node* yang berisi sistem kendali ketepatan posisi *quadcopter*. Masing-masing *node* tersebut menggunakan *library* Rospy yang terdapat dalam ROS agar dapat menjadi *node* dalam bahasa pemrograman *python*. Pada *node* PID dan *plant* juga menggunakan *library* Time yang terdapat pada *python* untuk mengambil nilai waktu dalam satuan detik. Setelah komunikasi *node* dirancang tahap selanjutnya membuat komunikasi antar komputer dengan *quadcopter* melalui *Wireless Fidelity* (*Wi-Fi*). Untuk lebih jelasnya dapat dilihat pada Gambar 5.2.



Gambar 5.2 Skema komunikasi sistem

5.1.2 Implementasi Komunikasi Sistem

Bagian ini membahas langkah untuk implementasi komunikasi sistem pada penelitian ini. Langkah awal adalah menghubungkan komputer dengan *quadcopter* melalui *Wi-Fi* seperti pada Gambar 5.3.



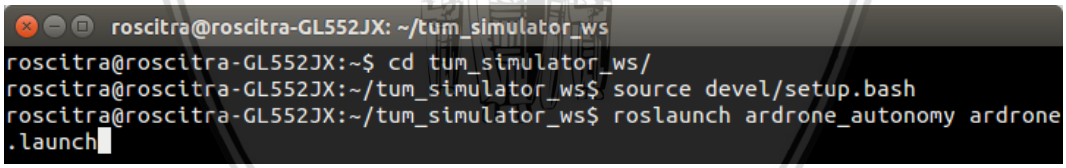
Gambar 5.3 Menghubungkan komputer dengan *quadcopter*

Pada Gambar 5.3 terlihat *Wi-Fi quadcopter* yang bernama ArDrone_1 telah terhubung dengan komputer. Untuk menjalankan program keseluruhan, diperlukan *library* rospy yang diperlukan agar node dapat dijalankan. Pengimplementasian *library* rospy pada sistem ini ditunjukkan dengan baris program dibawah ini.

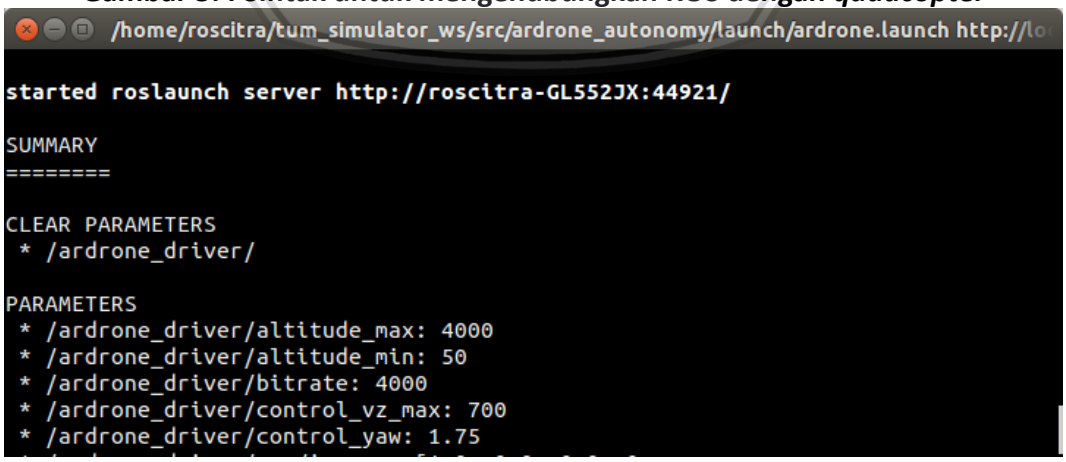
```
import rospy
```

Kemudian membuka terminal dan masuk pada *packages tum_simulator_ws*. Selanjutnya menghubungkan ROS dengan sistem *quadcopter* menggunakan *driver autonomy_ardrone* melalui terminal pada *Ubuntu* dengan sintak dibawah ini.

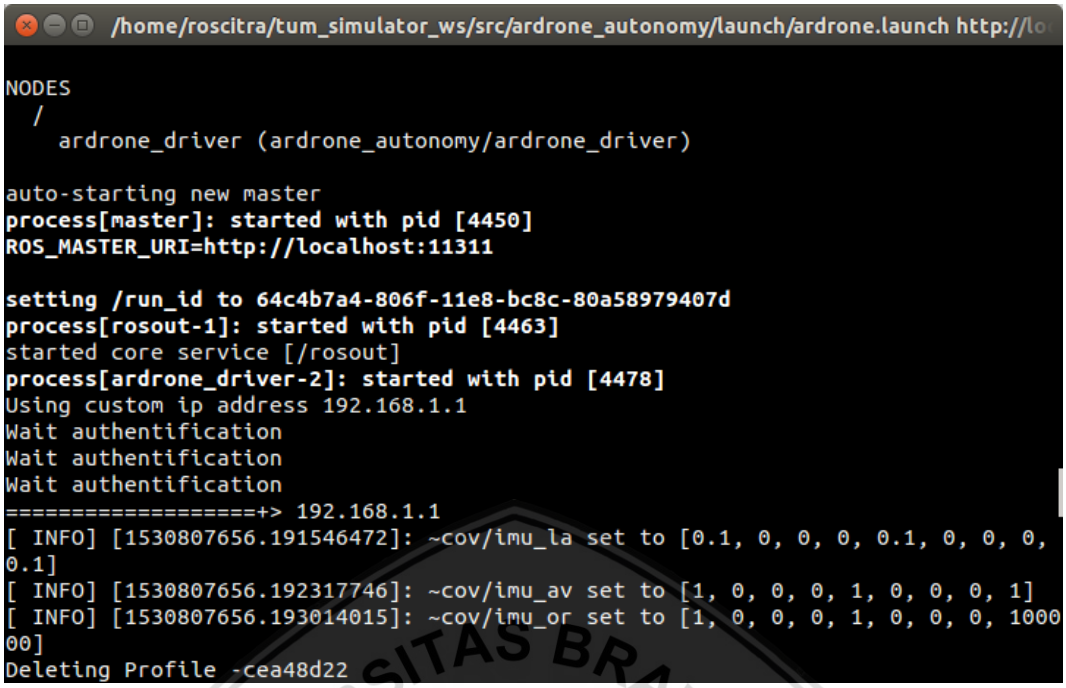
```
$ roslaunch ardrone_autonomy ardrone.launch
```



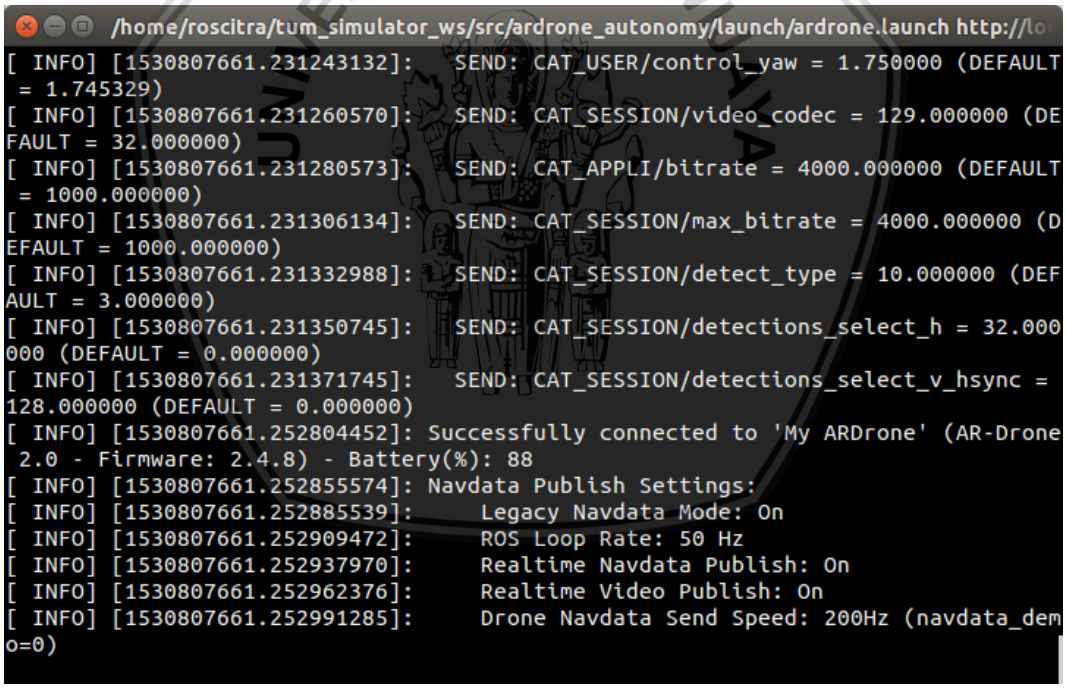
Gambar 5.4 Sintak untuk mengehubungkan ROS dengan *quadcopter*



Gambar 5.5 Proses memulai menghubungkan ROS dengan *quadcopter*

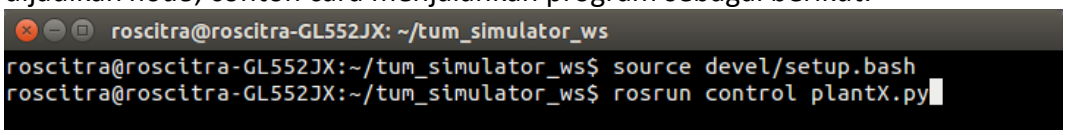


Gambar 5.6 Proses menghubungkan dengan IP *quadcopter*



Gambar 5.7 ROS sudah terhubung dengan sistem *quadcopter*

Gambar 5.4 menampilkan bahwa koneksi sudah berhasil dan menampilkan informasi tentang *quadcopter* seperti daya baterai serta *Navdata* yang siap diakses. Selanjutnya dapat menjalankan program yang telah dibangun dan telah dijadikan *node*, contoh cara menjalankan program sebagai berikut.



Gambar 5.8 Cara menjalankan program yang telah dijadikan *node*

Langkah untuk menjadikan program sebagai *node* dengan membuka terminal lalu masuk ke dalam direktori tempat program berada, setelah itu mengetikkan sintak seperti pada Gambar 5.9 dibawah ini.

```
roscitra@roscitra-GL552JX:~/tum_simulator_ws/src/control$ chmod +x plantX.py
roscitra@roscitra-GL552JX:~/tum_simulator_ws/src/control$
```

Gambar 5.9 Sintak untuk membuat *node* baru

Untuk mengecek apakah program sudah menjadi *node* adalah dengan cara mengetikkan perintah ls untuk membuka isi direktori tersebut. Jika nama file dari program sudah menjadi warna hijau seperti pada Gambar 5.10 maka program telah berhasil menjadi *node*.

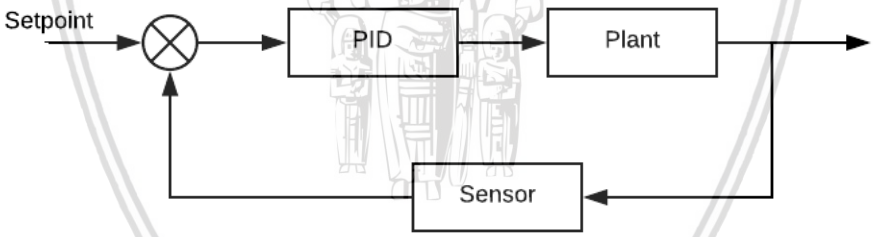
```
roscitra@roscitra-GL552JX: ~/tum_simulator_ws/src/control
roscitra@roscitra-GL552JX:~$ cd tum_simulator_ws/
roscitra@roscitra-GL552JX:~/tum_simulator_ws$ cd src
roscitra@roscitra-GL552JX:~/tum_simulator_ws/src$ cd control
roscitra@roscitra-GL552JX:~/tum_simulator_ws/src/control$ ls
CMakeLists.txt  package.xml  PID.pyc     ReadData.py  src
include         PID.py       plantX.py   ReadData.pyc
roscitra@roscitra-GL552JX:~/tum_simulator_ws/src/control$
```

Gambar 5.10 Cek status program

5.2 Kontroler PID

5.2.1 Diagram Blok Sistem

Pada tahap ini diagram blok dirancang sebelum mengimplementasikan sistem kendali ke *quadcopter*. Diagram blok sistem dapat dilihat pada Gambar 5.5.



Gambar 5.11 Diagram blok sistem kendali

Setpoint adalah jarak jarak tempuh yang diinginkan oleh pengguna, dalam sistem ini menggunakan satuan meter untuk jarak. Setelah *setpoint* sudah dimasukkan pengguna maka sistem akan mengecek *error* lalu mengirim nilai *error* tersebut ke kontroler PID. Hasil pengolahan kontroler PID dikirim ke *plant* yaitu *command* yang terdapat pada *quadcopter* untuk memberikan perintah bergerak. Selanjutnya sensor akan mengecek posisi saat ini, sensor yang digunakan adalah *Inertial Measurement Unit* (IMU) untuk mengambil nilai kecepatan linear *quadcopter* yang kemudian diolah menggunakan konsep turunan dan integral pada posisi serta kecepatan untuk mencari posisi saat ini.

5.2.2 Perancangan Algoritme PID

Kontroler PID bermaksud untuk melakukan penjumlahan dari proses-proses penguatan, pengintegralan dan penurunan nilai *error* dan mengeluarkan hasil perhitungan sebagai sinyal kontrol. Persamaan dari penjumlahan tersebut dapat

dilihat pada Persamaan 2.1. Untuk dapat diterapkan pada sistem, maka Persamaan 2.1 harus diubah ke dalam persamaan diskrit karena komputer digital bekerja secara diskrit dengan mendefinisikan persamaan terhadap waktu, sehingga diperoleh pendekatan integral dan deferensial untuk mendapat bentuk diskrit, menggunakan:

$$\int_0^t e\, dt \approx T \sum_{k=0}^{n} e(k) \quad (5.1)$$

$$\frac{de(t)}{dt} \approx \frac{e(n)-e(n-1)}{T} \quad (5.2)$$

Dimana n adalah waktu diskrit pada waktu t. Sehingga persamaan dari kontroler:

$$u(n) = K_p e(n) + K_i \sum_{k=0}^{n} e(k) + K_d(e(n) - e(n-1) \quad (5.3)$$

Dimana:

$U(t)$ = sinyal kendali
$e(t)$ = *error*
K_p = *gain* proporsional
K_i = *gain* integral
T = konstanta waktu
K_d = *gain* derivatif

Sebagai contoh, jika diketahui sebuah sistem dengan *setpoint* sebesar 10 dengan *tuning* parameter $K_p = 4{,}8$, $K_i = 1$ dan $K_d = 0{,}25$ didapatkan respon sistem sebagai berikut:

$e(0) = 10$
$e(1) = 8$

Maka,

$u(n) = 4{,}8 \times 8 + 1 \times (10 + 8) + 0{,}25 \times (8 - 10)$
$u(n) = 38{,}4 + 18 - 0{,}5$
$u(n) = 55{,}9$

5.2.3 Implementasi Algoritme PID

Algoritme PID diimplementasikan dengan cara diturunkan menjadi kode program, pada penelitian ini menggunakan bahasa pemrograman *python node* yang diprogram pada ROS. Adapun cuplikan kode program sesuai langkah perhitungan kontroler PID dapat dilihat pada Tabel 5.1 dan penjelasannya dapat dilihat pada Tabel 5.2. Untuk melihat kode program lengkap dapat dilihat di lampiran. Pada kode program ditambahkan batasan untuk nilai *Iterm* atau yang disebut *windup guard*. Fungsi *windup guard* ini untuk mengatasi masalah *overshoot* pada sistem kendali. Pada sistem kendali ini *windup guard* diatur sebesar 1 untuk batas atas dan -1 untuk batas bawah karena nilai parameter *command* di ROS diantara -1 dan 1. Nilai keluaran PID akan digunakan untuk mengontrol nilai *command linear.x* dan *linear.y* pada ROS yang berfungsi untuk mengatur pergerakan *quadcopter* agar mencapai ketepatan posisi sesuai dengan jarak tempuh yang diinginkan.

Tabel 5.1 Cuplikan kode program Algoritme PID

No	Kode Program
1	`error = self.SetPoint - feedback_value`
2	`self.current_time = time.time()`
3	`self.delta_time = self.current_time - self.last_time`
4	`delta_error = error - self.last_error`
5	`if (self.delta_time >= self.sample_time):`
6	`    self.PTerm = self.Kp * error`
7	`    self.ITerm += error * self.delta_time`
8	`    if (self.ITerm < -self.windup_guard):`
9	`        self.ITerm = -self.windup_guard`
10	`    elif (self.ITerm > self.windup_guard):`
11	`        self.ITerm = self.windup_guard`
12	`    self.DTerm = 0.0`
13	`    if self.delta_time > 0:`
14	`        self.DTerm = delta_error / self.delta_time`
15	`        self.last_time = self.current_time`
16	`        self.last_error = error`
17	`        self.output = self.PTerm + (self.Ki * self.ITerm) + (self.Kd * self.DTerm)`

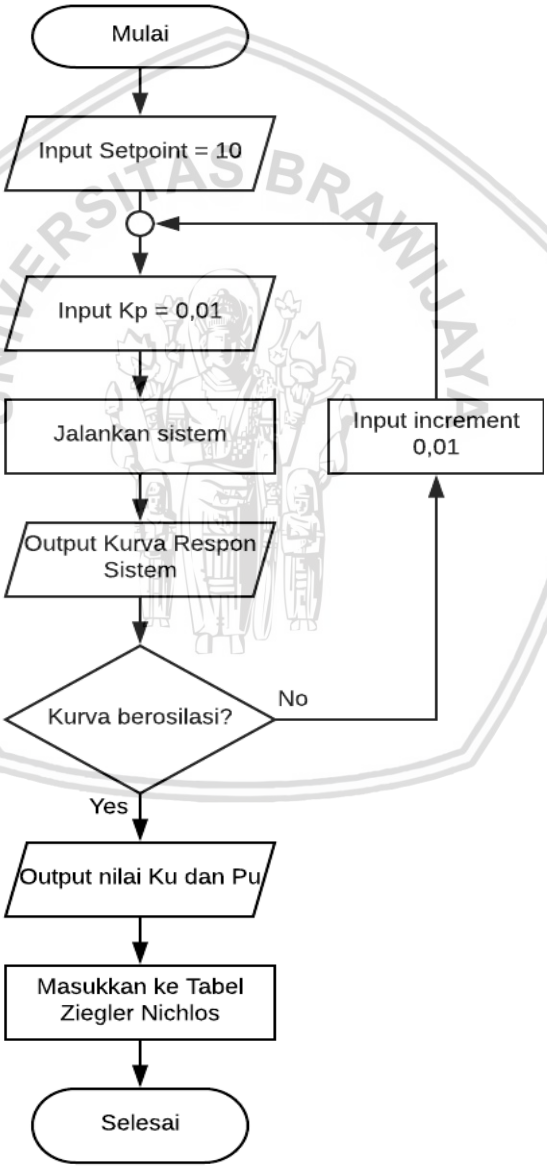
Tabel 5.2 Penjelasan Cuplikan kode program Algoritme PID

No	Penjelasan Kode Program
1	Menghitung nilai *error* pada sistem
2	Menyimpan waktu saat ini dalam variabel *current_time*
3	Menyimpan nilai *delta time*
4	Menghitung selisih *error* saat ini dengan *error* sebelumnya
5	Kondisi jika *delta time* harus lebih dari sama dengan *sample time*
6	Menghitung nilai P sesuai Persamaan 5.3
7	Menghitung *error* untuk konstanta I seperti Persamaan 5.1
8	Kondisi dalam kondisi jika *Iterm* kurang dari batas bawah *windup*
9	*Statement* untuk mengganti nilai *ITerm* menjadi nilai batas bawah *windup*
10	Jika *Iterm* lebih dari batas atas *windup*
11	*Statement* untuk mengganti nilai *ITerm* menjadi nilai batas atas *windup*
12	Mengatur nilai D sebesar 0 (nol)
13	Kondisi dalam kondisi jika *delta time* lebih dari 0
14	Menghitung nilai error untuk D seperti Persamaan 5.2
15	Menyimpan waktu sekarang di variabel *last_error*
16	Menyimpan error terakhir di variabel *last_error*
17	Menghitung keluaran PID sesuai Persamaan 5.3

Selanjutnya membuat program baru sebagai *plant* yaitu sistem yang harus dikontrol. Program *plant* ini berfungsi untuk mengatur hubungan antara algoritme PID dengan pengendalian yang ingin dilakukan, dalam penelitian ini dilakukan pengendalian jarak tempuh *quadcopter*. Program *plant* dijalankan jika posisi dari *quadcopter* sudah *hover*. Jarak tempuh *quadcopter* didapatkan dari perhitungan matematis dengan menggunakan Persamaan 2.4 yang kemudian di integralkan.

5.3 *Tuning* Parameter PID

5.3.1 Perancangan *Tuning* Parameter PID



Gambar 5.12 Diagram alir mencari nilai K_u dan P_u

Untuk menentukan nilai parameter PID menggunakan metode *ziegler nichlos* dengan respon sistem osilasi diperlukan nilai K_u terlebih dahulu. Dalam sistem

nilai K_u sama dengan Kp atau konstanta proporsional. Nilai K_u ini didapatkan seperti yang ditunjukkan oleh Gambar 5.12. *Setpoint* diatur sebesar 10 dekameter karena diambil dari nilai *setpoint* terkecil dalam pengujian. Nilai K_u dinaikkan 0,01 setiap gagal mendapatkan respon sistem osilasi agar dapat dilihat perubahan respon sistem dengan teliti. Nilai P_u didapatkan setelah nilai K_u dengan cara mengamati waktu yang dibutuhkan dalam satu gelombang. Setelah didapatkan nilai K_u dan P_u maka dihitung nilai parameter PID sesuai dengan perhitungan metode *ziegler nichols* pada Tabel 2.1.

5.3.2 Implementasi Penalaan Parameter PID

Hasil perancangan penalaan parameter PID diimplementasikan dalam kode program *plant* yang dapat dilihat pada cuplikan kode program di Tabel 5.3. Untuk mengubah nilai K_u atau konstanta proporsional dapat dilakukan dengan mengubah nilai P pada baris 2 di Tabel 5.3. untuk mangatur *setpoint* dapat dilakukan dengan mengubah nilai variabel *setpoint* pada baris 14 pada Tabel 5.3. Kode program lebih lengkapnya dapat dilihat pada lampiran.

Tabel 5.3 Cuplikan kode program plant

No	Kode Program
1	`def posisi(self):`
2	`P= 0.01 //untuk mengatur nilai P`
3	`I= 0 //untuk mengatur nilai I`
4	`D= 0 //untuk mengatur nilai D`
5	`pid = PID.PID(P, I, D)`
6	`pid.SetPoint=0.0`
7	`pid.setSampleTime(0.025)`
8	`pid.last_error=0.0`
9	`feedback = 0.0`
10	`totalX=0.0`
11	`i=0`
12	`end_time=0.0`
13	`totaltime=0.0`
14	`pid.SetPoint = 10 //untuk mengatur nilai setpoint`
15	`while pid.SetPoint is not None:`
16	`pid.update(feedback)`
17	`self.output = pid.output`
18	`outputawal=self.output`
19	`if self.output > 1:`
20	`self.output = 1`
21	`elif self.output < -1:`
22	`self.output = -1`
23	`print (self.output)`
24	`start_time = time.clock()`
25	`bless.SetCommand(self.output,0,0,0,0,0)`
26	`end_time = time.clock()`
27	`timer = end_time - start_time`
28	`totaltime = totaltime + timer`
29	`posisiX = uav.vx * timer`
30	`totalX += posisiX`
31	`feedback = totalX`
32	`i+=1`

Hasil keluaran dari implementasi ini berupa data yang akan dijadikan kurva respon sistem untuk mengamati respon sistem yang berosilasi. Karakteristik respon sistem yang berosilasi seperti pada Gambar 2.11 dimana jarak antar tiap gelombang dan tinggi antar tiap hampir sama.

BAB 6 PENGUJIAN DAN ANALISIS

Tahap ini akan menguji serta menganalisis sistem yang sudah dibangun, mengetahui apakah sistem sudah sesuai dengan tujuan dari penelitian ini, dan menjawab rumusan masalah yang telah dijabarkan sebelumnya. Pengujian dilakukan dalam dua tahap yaitu pengujian sistem kendali dan pengujian jarak tempuh. Pada pengujian sistem kendali dilakukan pengujian untuk *tuning* paramater PID menggunakan metode *ziegler nichols* dan pengujian pada masing-masing pengendali PID. Sedangkan pada pengujian jarak tempuh dilakukan dengan beberapa *setpoint* yaitu 10 dm, 15 dm dan 20 dm. Pengujian dilakukan pada masing-masing gerakan *pitch* dan *roll*.

6.1 Pengujian Sistem Kendali

6.1.1 Tujuan Pengujian

Pengujian ini bertujuan untuk menentukan parameter pengendali *proporsional*, *integral* dan *differensial* pada metode *ziegler nichols* dan mendapatkan respon sistem yang bagus dimana *rise time* dan *error steady state* semakin kecil atau mendekati *setpoint* yang diinginkan.

6.1.2 Pelaksanaan Pengujian

Pengujian dilakukan dengan menerbangkan *quadcopter* hingga mencapai *hover* lalu baru menjalankan program dengan menguji beberapa nilai konstanta proporsional yang berbeda nantinya akan didapatkan nilai parameter PID yang sesuai untuk digunakan dalam pengendalian pergerakan *roll* dan *pitch*.

6.1.3 Prosedur Pengujian

Pengujian dilakukan dengan langkah-langkah sesuai dengan prosedur berikut ini.

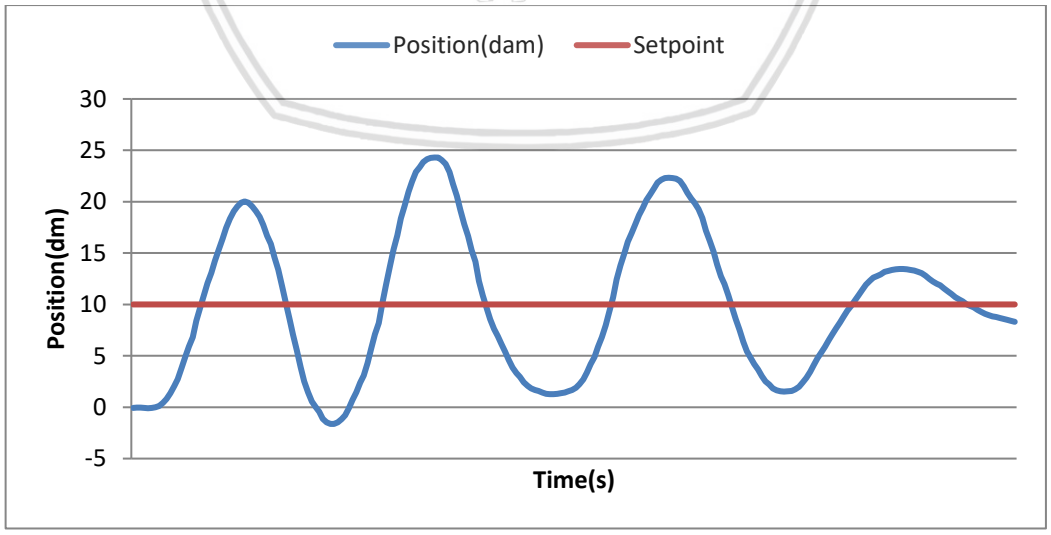
1. Menyiapkan *quadcopter*, memasang baterai *quadcopter* dan menunggu hingga *quadcopter* terkalibrasi. Setelah terdengar suara *beep* sebanyak empat kali maka *quadcopter* siap digunakan.
2. Menghubungkan komputer dengan *quadcopter* melalui *Wi-Fi.*
3. Membuka *terminal* pada *ubuntu* dan masuk ke direktori dengan menggetik “cd tum_simulator_ws”, kemudian mengetik “source devel/setup.bash”. setelah itu mengetik “roslaunch ardrone_autonomy ardrone.launch” untuk menghubungkan *ROS* dengan *quadcopter*.
4. Mengecek sisa daya baterai pada *quadcopter*. Daya baterai minimal 30% karena kinerja *quadcopter* berjalan baik dengan daya baterai diatas 30%.
5. Membuka *terminal* baru lagi pada *ubuntu* dan masuk ke direktori dengan menggetik “cd tum_simulator_ws”. kemudian mengetik “source

devel/setup.bash". Setelah itu mengetik "rostopic pub ardrone/takeoff std_msgs/Empty" untuk menerbangkan *quadcopter*.

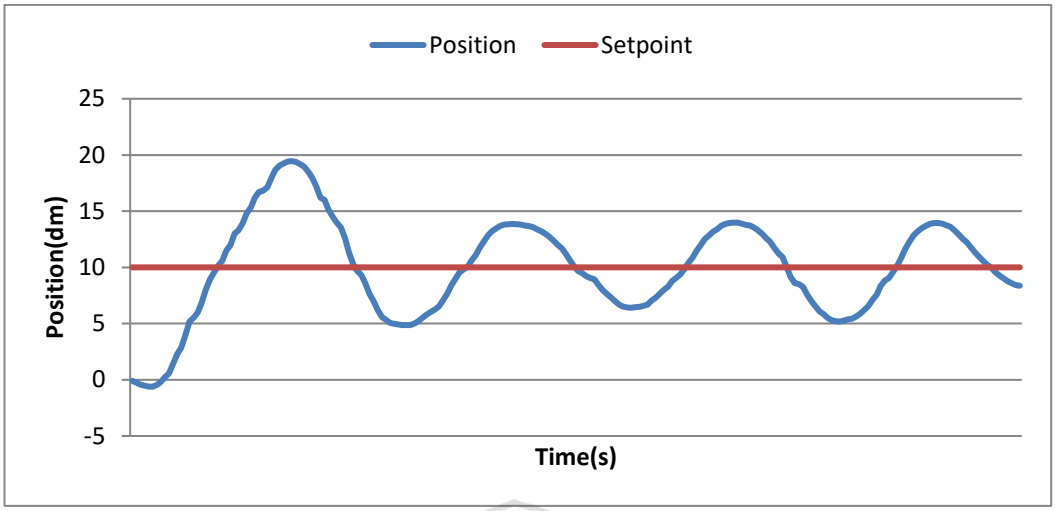
6. Menunggu hingga *quadcopter* melakukan *hover* secara otomatis.
7. Memberikan nilai konstanta proporsional dan menyetel $T_i = \infty$ dan $T_d = 0$. Nilai konstanta proporsional diubah secara bertahap hingga tanggapan sistem berosilasi secara kontiyu.
8. Mengatur nilai *setpoint* sebesar 10.
9. Membuka *terminal* baru lagi pada *ubuntu* dan masuk ke direktori dengan menggetik "cd tum_simulator_ws", kemudian mengetik "source devel/setup.bash". setelah itu mengetik "rosrun control plant" untuk menjalankan sistem kendali *quadcopter*.
10. Mengamati grafik posisi terhadap waktu dan mencatat nilai *ultimate gain* K_u dan *ultimate period* T_u.
11. Ulangi pengujian hingga mendapatkan hasil konstanta proporsional dengan grafik osilasi yang baik.
12. Setelah didapatkan hasil K_u dan T_u selanjutnya mengulang prosedur 9 namun terlebih dahulu mengatur nilai P, I dan D sesuai tabel hasil *tuning* parameter PID. Pengujian dilakukan pada tiga pengendali yaitu P, PI, dan PID.
13. Mengamati gerak *quadcopter* dan mencatat pengendali apa yang hasil respon sistemnya memuaskan.

6.1.4 Hasil Pengujian

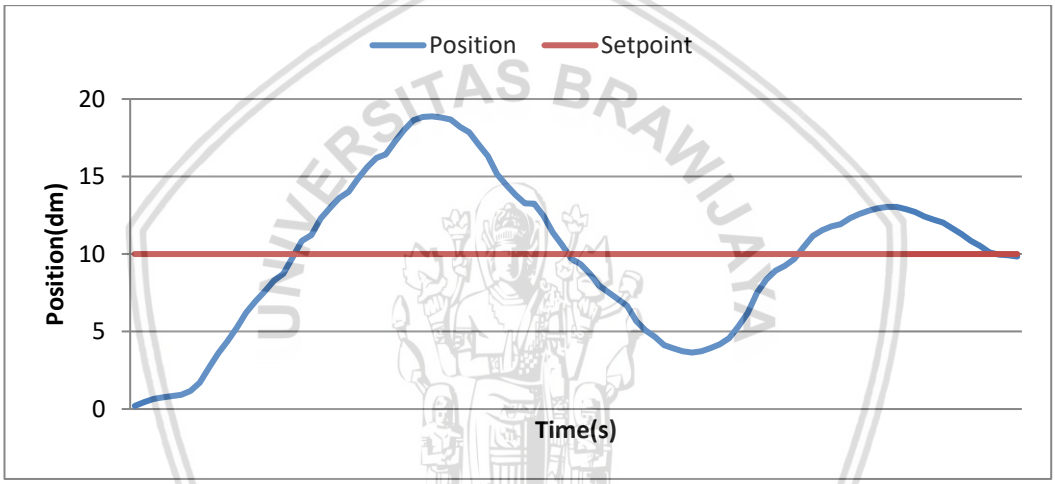
Tuning parameter PID dilakukan pada pergerakan *pitch* dengan *setpoint* 10 dm. Berikut hasil pengujian untuk mencari nilai $K_u$sebesar 0,01 hingga 0,07.



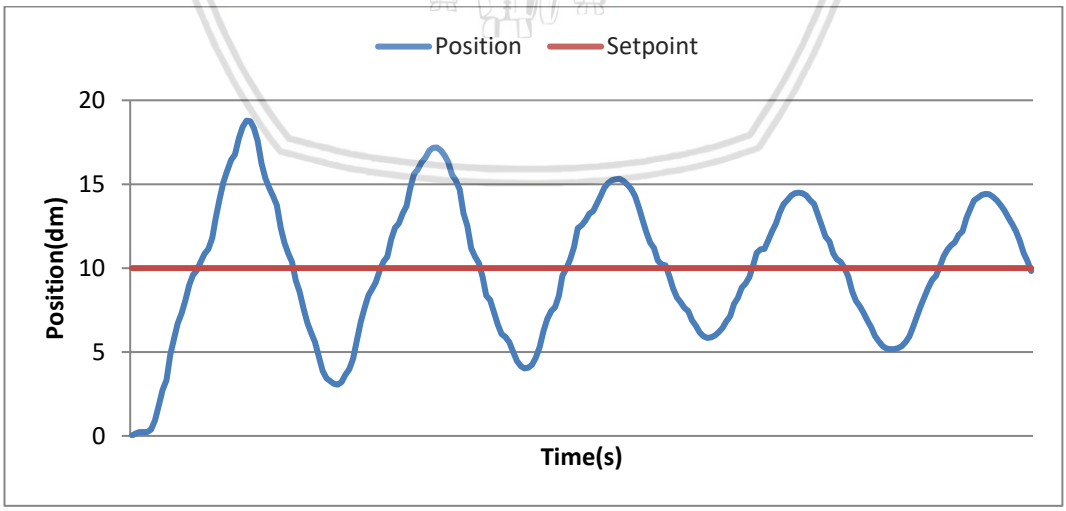
Gambar 6.1 Respon sistem dengan nilai K_u sebesar 0,01



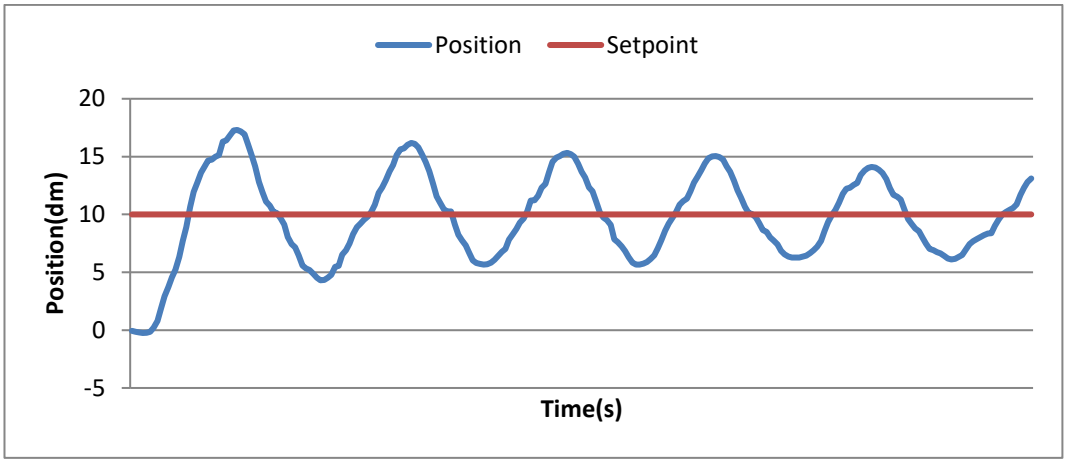
Gambar 6.2 Respon sistem dengan nilai K_u sebesar 0,02



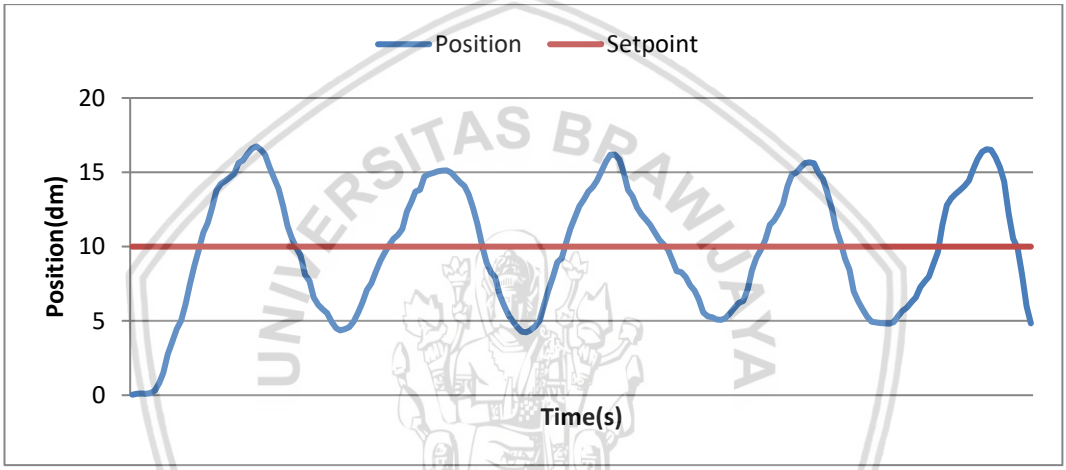
Gambar 6.3 Respon sistem dengan nilai K_u sebesar 0,03



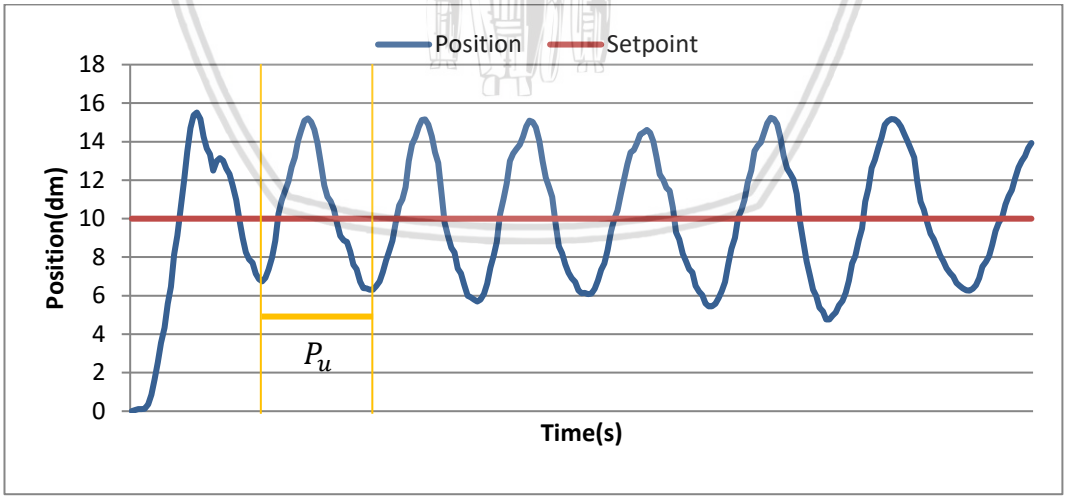
Gambar 6.4 Respon sistem dengan nilai K_u sebesar 0,04



Gambar 6.5 Respon sistem dengan nilai K_u sebesar 0,05



Gambar 6.6 Respon sistem dengan nilai K_u sebesar 0,06



Gambar 6.7 Respon sistem dengan nilai K_u sebesar 0,07

Dari hasil pengamatan terhadap respon sistem pada pengujian untuk mencari nilai K_u didapatkan respon sistem yang berosilasi sesuai dengan Gambar 2.11 dengan K_u= 0,07 dan P_u = 0,091981. Grafik dapat dilihat pada Gambar 6.7. Perhitungan *tuning* parameter PID setiap kontroler dapat dilihat pada Persamaan 6.1 hingga Persamaan 6.6 sebagai berikut :

Kontroler P

$$K_p = 0{,}5 \times K_u \quad (6.1)$$

Kontroler PI

$$K_p = 0{,}45 \times K_u \quad (6.2)$$

$$T_i = \frac{1}{12} \times P_u \quad (6.3)$$

Kontroler PID

$$K_p = 0{,}6 \times K_u \quad (6.4)$$

$$T_i = 0{,}5 \times P_u \quad (6.5)$$

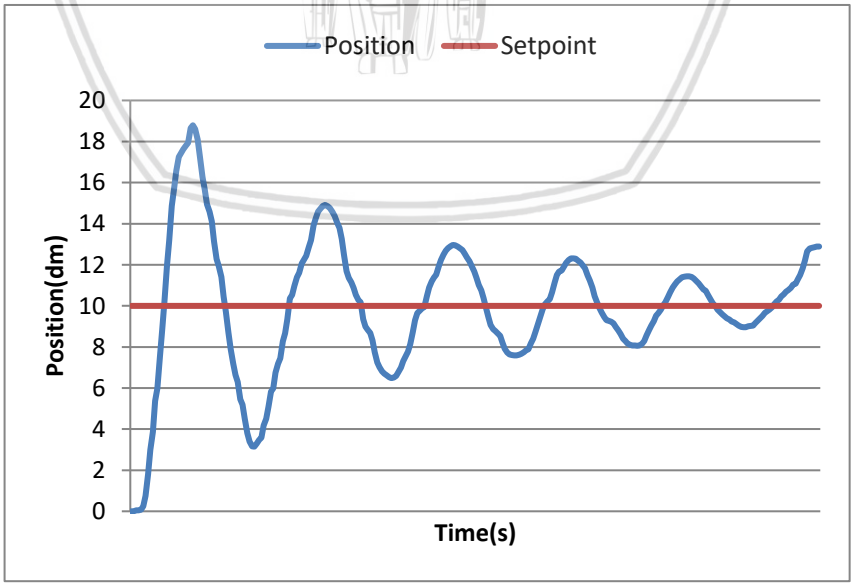
$$T_d = 0{,}125 \times P_u \quad (6.6)$$

Sehingga didapatkan hasil perhitungan *tuning* parameter PID berdasarkan metode *ziegler nichols* pada Tabel 6.1.

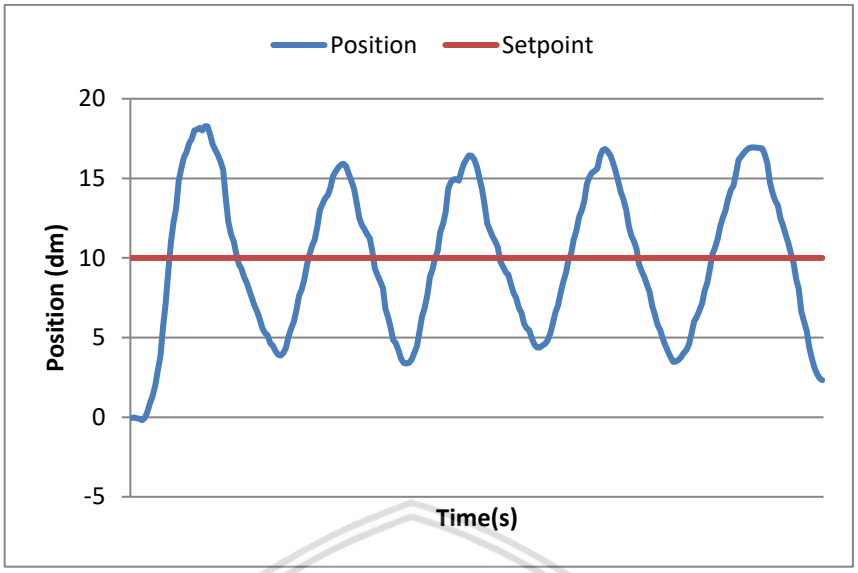
Tabel 6.1 Hasil *tuning* parameter PID gerak *pitch* dengan metode osilasi

Tipe Kontroler	K_p	T_i	T_d
P	0,035		
PI	0,0315	0,076	
PID	0,042	0,046	0,0115

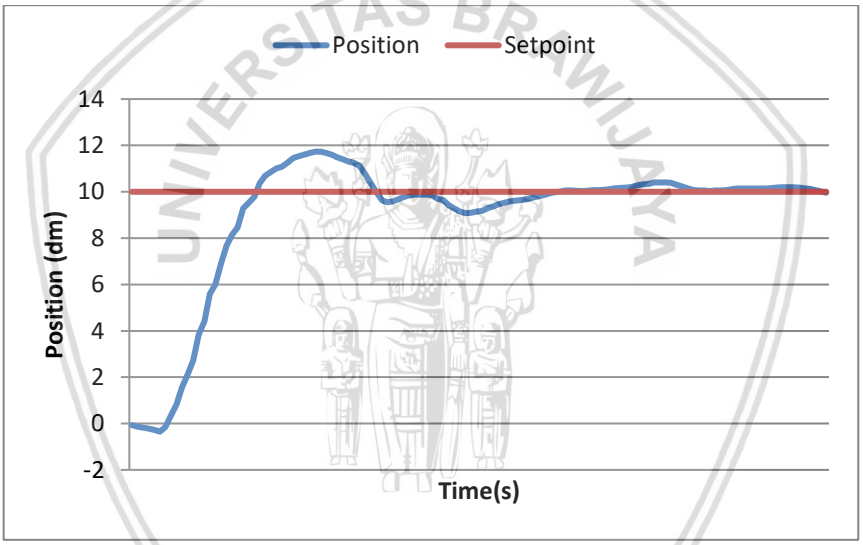
Berdasarkan Tabel 6.1 berikut hasil pengujian pada masing-masing pengendali P, PI dan PID.



Gambar 6.8 Respon sistem pengendali P pada *pitch*



Gambar 6.9 Respon sistem pengendali PI pada *pitch*



Gambar 6.10 Respon sistem pengendali PID pada *pitch*

6.1.5 Analisis Hasil Pengujian

Analisis pada pengujian ini menggunakan tanggapan sistem atau respon sistem. Respon sistem adalah perubahan perilaku *output* terhadap perubahan sinyal *input*. Respon sistem berupa kurva ini akan menjadi dasar untuk menganalisa karakteristik sistem selain menggunakan persamaan/model matematika. Bentuk kurva respon sistem dapat dilihat setelah mendapatkan sinyal *input.* Berdasarkan karakteristik respon waktu (*time response*) pada sistem berikut diuraikan spesifikasi performansi respon waktu pada Tabel 6.2.

Tabel 6.2 Analisis respon sistem berdasarkan waktu

Kontrol	Parameter	Hasil Analisa
P	*settling time* (t_s)	Ukuran waktu yang menyatakan respon telah masuk $\pm 2\%$ dari keadaan *steady* state pada pengendali P tidak ada. Karena kurva respon sistem tidak pernah mencapai kestabilan pada *steady state*.
	peak time (t_p)	Ukuran waktu puncak pada pengendali P sebesar 0,0853 s
	Overshoot max (M_p)	Hasil kesalahan puncak pertama sebesar 18,79 dm, sehingga didapatkan nilai *overshoot* sebesar 8,79 dm
PI	*settling time* (t_s)	Ukuran waktu yang menyatakan respon telah masuk $\pm 2\%$ dari keadaan *steady* state pada pengendali PI tidak ada. Karena kurva respon sistem tidak pernah mencapai kestabilan pada *steady state*.
	peak time (t_p)	Ukuran waktu puncak pada pengendali PI sebesar 0,092 s
	Overshoot max (M_p)	Hasil kesalahan puncak pertama sebesar 18,27 dm, sehingga didapatkan nilai *overshoot* sebesar 8,27 dm
PID	*settling time* (t_s)	Ukuran waktu yang menyatakan respon telah masuk $\pm 2\%$ dari keadaan *steady* state pada pengendali PID sebesar 0,271 s.
	peak time (t_p)	Ukuran waktu puncak pada pengendali PID sebesar 0,09 s
	Overshoot max (M_p)	Hasil kesalahan puncak pertama sebesar 11,73 dm, sehingga didapatkan nilai *overshoot* sebesar 1,73 dm

Dari pengamatan terhadap Tabel 6.2 hanya pengendali PID dapat digunakan sebagai sistem kendali jarak tempuh pada *quadcopter* dengan *settling time* sebesar 0,271 s, *peak time* sebesar 0,09 s dan *overshoot* sebesar 1,73 dm. Hal ini karena respon sistem sesuai yang diinginkan *setpoint* dimana memiliki *rise time* dari pada pengendali lain dan *overshoot* respon sistem lebih rendah dibanding *overshoot* respon sistem pengendali P dan pengendali PID. Maka, pada pengujian sistem selanjutnya akan menggunakan pengendali PID dengan nilai K_p sebesar 0,042, K_i sebesar 0,046 dan K_d sebesar 0,0115.

6.2 Pengujian Jarak tempuh

6.2.1 Tujuan Pengujian

Pengujian ini dilakukan setelah pengujian sistem kendali selesai. Tujuan pengujian ini untuk menguji ketepatan posisi dengan *setpoint* yang berbeda dan mengetahui respon sistem berdasarkan respon waktu yaitu respon yang spesifikasi perfomarsinya didasarkan pada pengamatan bentuk respon *output* sistem terhadap berubahnya waktu.

6.2.2 Pelaksanaan Pengujian

Pengujian dilakukan dengan menerbangkan *quadcopter* hingga *hover* lalu menjalankan sistem kendali dan mengatur nilai *setpoint* sebesar 10 dm, 15 dm dan 20 dm pada setiap pergerakan maju, mundur, kiri dan kanan. Pengujian dilakukan dalam ruangan.

6.2.3 Prosedur Pengujian

Pengujian dilakukan dengan langkah-langkah sesuai dengan prosedur berikut ini.

1. Menyiapkan *quadcopter*, memasang baterai *quadcopter* dan menunggu hingga *quadcopter* terkalibrasi. Setelah terdengar suara *beep* sebanyak empat kali maka *quadcopter* siap digunakan.
2. Menghubungkan komputer dengan *quadcopter* melalui *Wi-Fi.*
3. Membuka *terminal* pada *ubuntu* dan masuk ke direktori dengan menggetik "cd tum_simulator_ws", kemudian mengetik "source devel/setup.bash". setelah itu mengetik "roslaunch ardrone_autonomy ardrone.launch" untuk menghubungkan *ROS* dengan *quadcopter*.
4. Mengecek sisa daya baterai pada *quadcopter*. Daya baterai minimal 30% agar *quadcopter* tetap stabil.
5. Membuka *terminal* baru lagi pada *ubuntu* dan masuk ke direktori dengan menggetik "cd tum_simulator_ws". kemudian mengetik "source devel/setup.bash". Setelah itu mengetik mengetik "rostopic pub ardrone/takeoff std_msgs/Empty" untuk menerbangkan *quadcopter*.
6. Menunggu hingga *quadcopter* melakukan *hover* secara otomatis.
7. Mengatur nilai *setpoint* pada program.
8. Membuka *terminal* baru lagi pada *ubuntu* dan masuk ke direktori dengan menggetik "cd tum_simulator_ws". kemudian mengetik "source devel/setup.bash". setelah itu mengetik "rosrun control plant" untuk menjalankan sistem kendali *quadcopter*.
9. Mengamati pergerakan *quadcopter* hingga mencapai posisi yang diinginkan, setelah itu mengukur dan mencatat panjang lintasan jarak yang telah ditempuh *quadcopter*.
10. Analisis hasil pengamatan. Jika ditemukan masalah maka kembali ke tahap perancangan sistem, namun jika berhasil maka lanjutkan pengujian pada jarak lainnya.
11. Ulangi pengujian dengan *setpoint* yang berbeda sesuai pada pelaksaan pengujian.

6.2.4 Hasil Pengujian

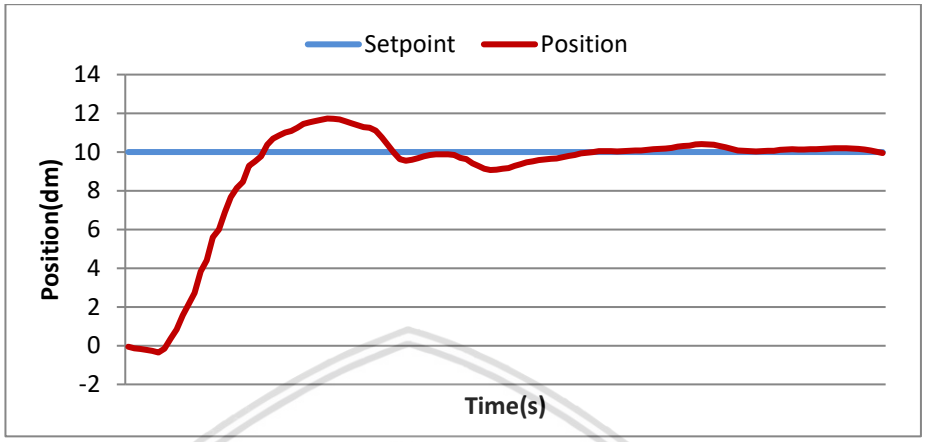
Berikut hasil pengujian yang telah dilakukan pada pergerakan sudut *roll* dan pergerakan sudut *pitch* dari *quadcopter*.

1. Pergerakan sudut *pitch* dari *quadcopter*

   Gerak maju dari *quadcopter* disebabkan oleh perubahan sudut *pitch* yang bernilai positif atau lebih dari nol. Sebaliknya, gerak mundur dari *quadcopter* disebabkan oleh perubahan sudut *pitch* yang bernilai negatif atau kurang dari

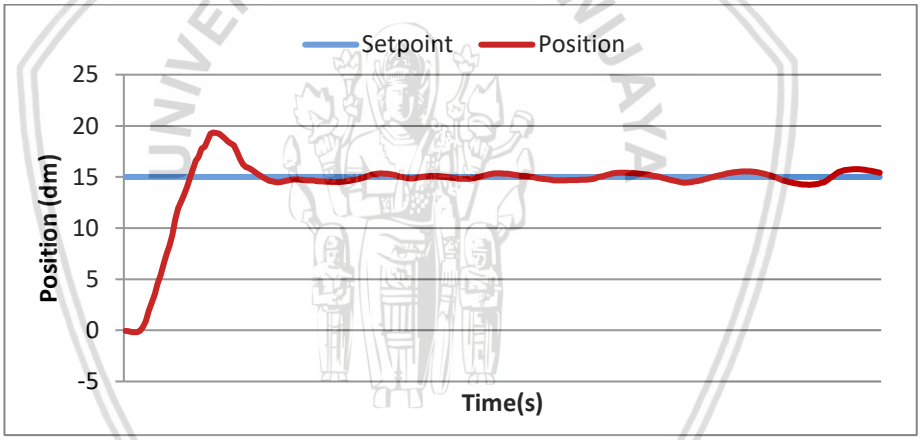
nol. Besar dan kecilnya perubahan sudut ini tergantung dari perintah *command linear.x* pada ROS. Berikut hasil pengujian gerak maju dan mundur dari *quadcopter* dengan *setpoint* 10 dm, 15 dm dan 20 dm.

a. Gerak maju dengan *setpoint* 10 dm



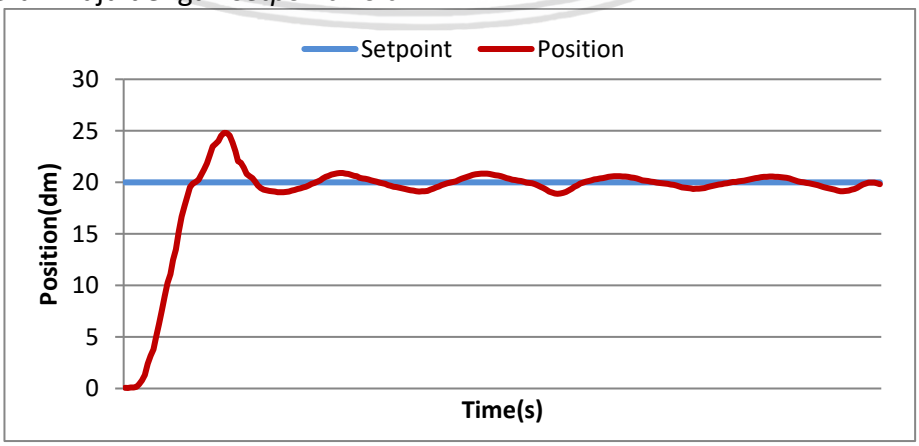
Gambar 6.11 Respon sistem dari gerak maju dengan *setpoint* 10

b. Gerak maju dengan *setpoint* 15 dm



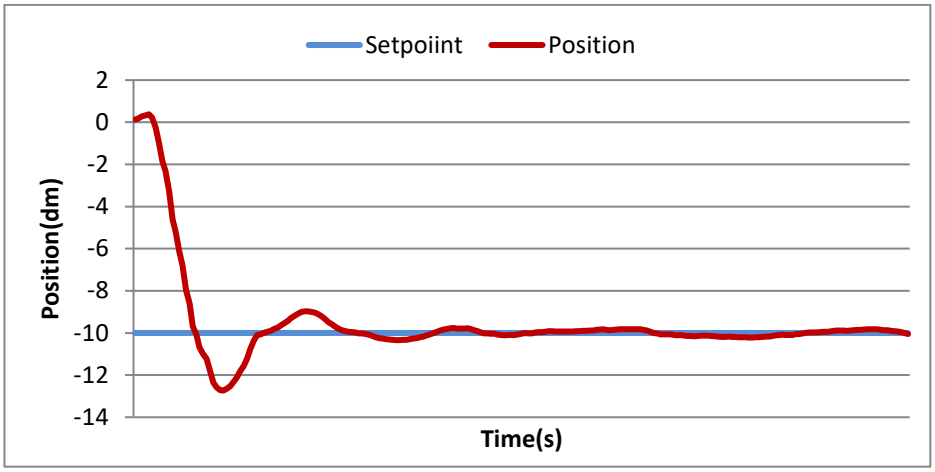
Gambar 6.12 Respon sistem dari gerak maju dengan *setpoint* 15

c. Gerak maju dengan *setpoint* 20 dm



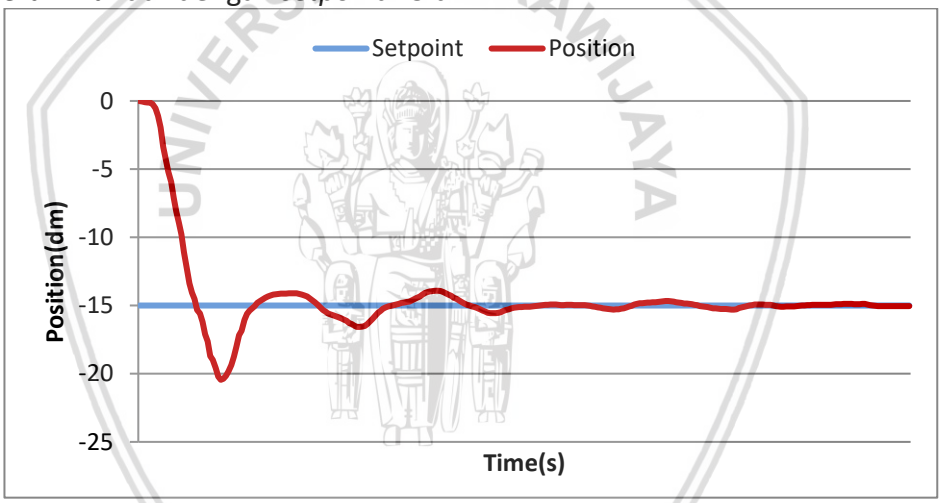
Gambar 6.13 Respon sistem dari gerak maju dengan *setpoint* 20

d. Gerak mundur dengan *setpoint* 10 dm



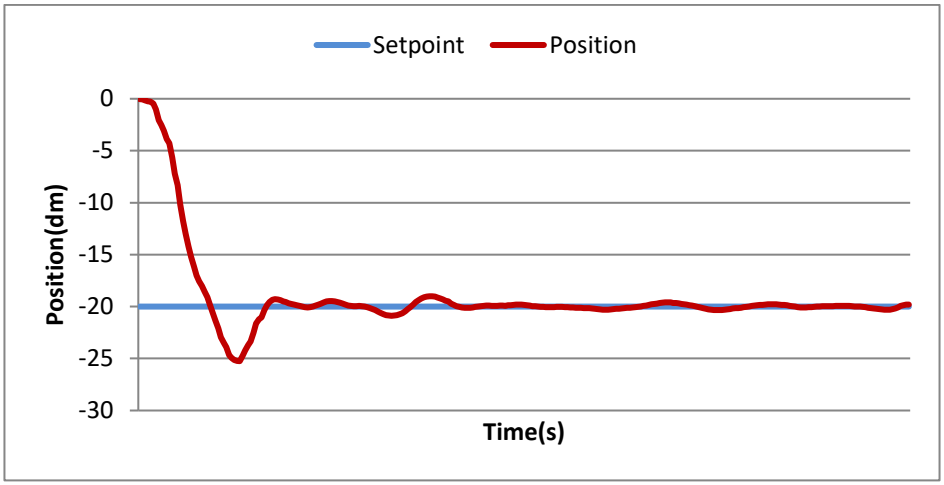
Gambar 6.14 Respon sistem dari gerak mundur dengan *setpoint* 10

e. Gerak mundur dengan *setpoint* 15 dm



Gambar 6.15 Respon sistem dari gerak mundur dengan *setpoint* 15

f. Gerak mundur dengan *setpoint* 20 dm

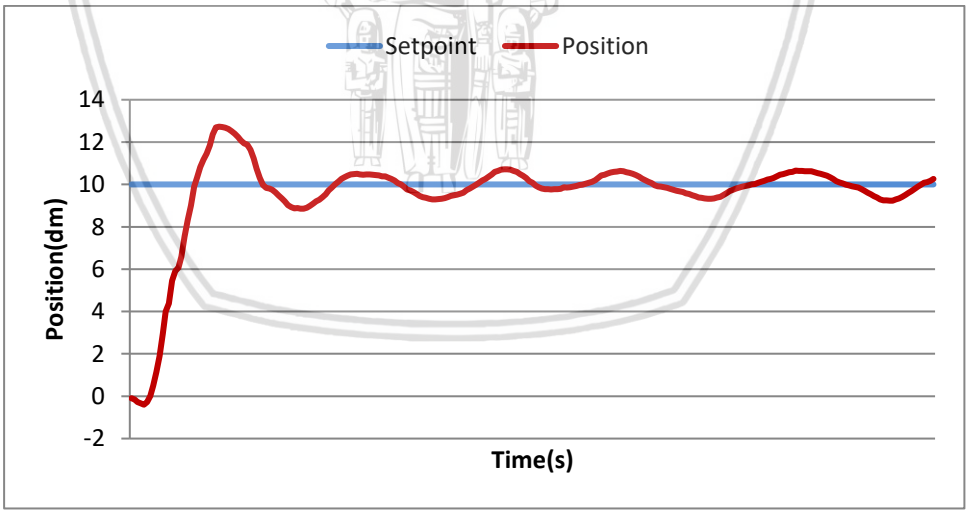


Gambar 6.16 Respon sistem dari gerak mundur dengan *setpoint* 20

2. Pergerakan sudut *roll* dari *quadcopter*

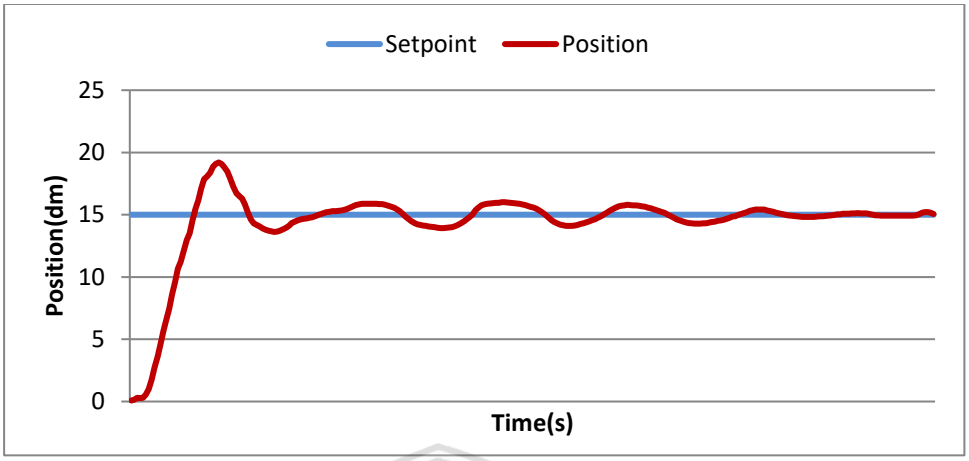
   Perubahan pada sudut *roll* akan menghasilkan pergerakan ke kiri dan ke kanan. Apabila nilai sudut *roll* bernilai positif atau lebih dari 0 maka *quadcopter* akan bergerak ke kiri, sebaliknya jika nilai sudut *roll* bernilai negatif atau kurang dari 0 maka *quadcopter* akan bergerak ke kanan. Perubahan ini disebabkan oleh perintah *command linear.y* pada ROS. Berikut hasil pengujian gerak ke kiri dan ke kanan dari *quadcopter* dengan *setpoint* 10 dm, 15 dm dan 20 dm.

   a. Gerak ke kiri dengan *setpoint* 10 dm



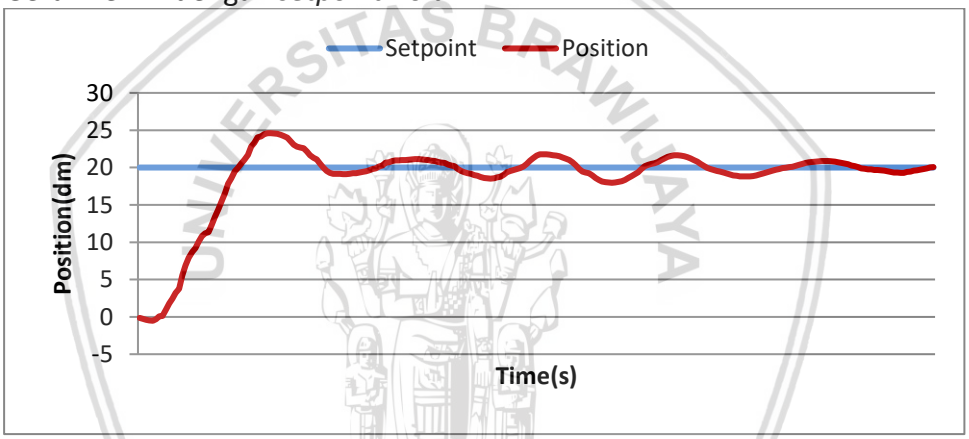
Gambar 6.17 Respon sistem dari gerak ke kiri dengan *setpoint* 10

b. Gerak ke kiri dengan *setpoint* 15 dm



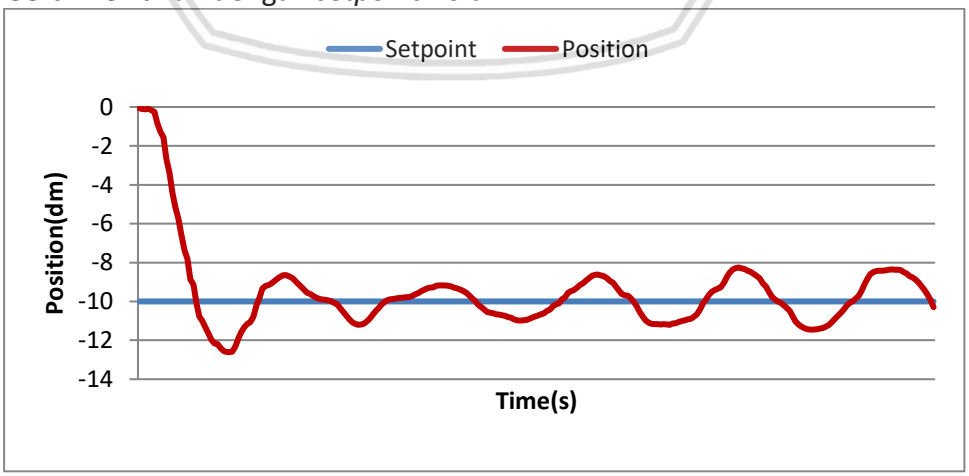
Gambar 6.18 Respon sistem dari gerak ke kiri dengan *setpoint* 15

c. Gerak ke kiri dengan *setpoint* 20 dm



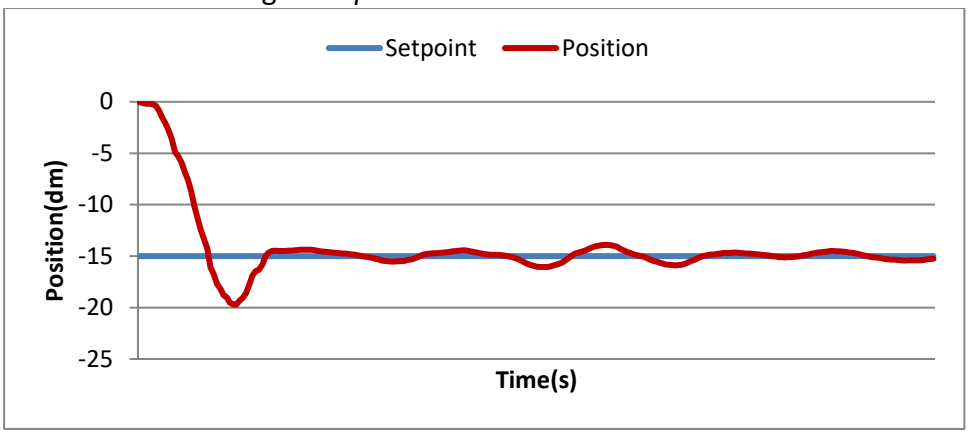
Gambar 6.19 Respon sistem dari gerak ke kiri dengan *setpoint* 20

d. Gerak ke kanan dengan *setpoint* 10 dm



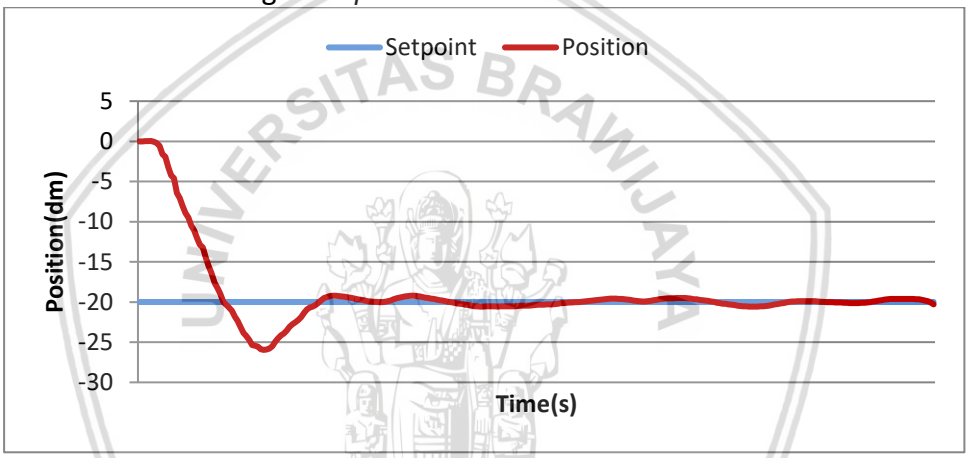
Gambar 6.20 Respon sistem dari gerak ke kanan dengan *setpoint* 10

e. Gerak ke kanan dengan *setpoint* 15 dm



Gambar 6.21 Respon sistem dari gerak ke kanan dengan *setpoint* 15

f. Gerak ke kanan dengan *setpoint* 20 dm



Gambar 6.22 Respon sistem dari gerak ke kanan dengan *setpoint* 20

6.2.5 Analisis Hasil Pengujian

Berdasarkan pengujian yang dilakukan terkait jarak tempuh *quadcopter* dengan *setpoint* yang berbeda yaitu 10 dm, 15 dm dan 20 dm didapatkan presentase kesalahan sistem. Nilai kesalahan dari setiap gerakan *quadcopter* dirata-rata kemudian dijumlahkan dengan kesalahan pada setiap *setpoint* yang sama lalu dirata-rata kembali hingga didapat nilai kesalahan total. Perhitungan kesalahan dilakukan dengan menggunakan Persamaan 6.7, 6.8 dan 6.9.

$$error = setpoint - feedback \tag{6.7}$$

$$Presentase\ error\ gerak\ = \frac{|error|}{setpoint} \times 100\% \tag{6.8}$$

$$\overline{error} = \frac{\sum_0^i presentase\ error}{i} \tag{6.9}$$

Dengan:

setpoint : nilai jarak tempuh *quadcopter* yang diinginkan

feedback : nilai jarak tempuh saat ini

i : Jumlah data

Berdasarkan Persamaan 6.7 didapatkan hasil rata-rata kesalahan pada setiap gerak berdasarkan *setpoint* yang dapat dilihat pada Tabel 6.3.

Tabel 6.3 Rata-rata kesalahan pada gerak *quadcopter*

Gerakan	Rata-rata kesalahan tiap gerak (dm)		
	Setpoint (dm)		
	10	15	20
Maju	0,986	0,645	1,011
Mundur	0,456	0,499	0,903
Ke kiri	0,442	0,597	1,214
Ke kanan	0,458	0,71	0,908

Dari rata-rata kesalahan didapatkan presentase kesalahan setiap gerak berdasarkan Persamaan 6.8 pada Tabel 6.4.

Tabel 6.4 Presentase kesalahan gerak *quadcopter*

setpoint	**Maju**	**Mundur**
10	$\frac{0,986}{10} \times 100\% = 9,86\%$	$\frac{0,456}{10} \times 100\% = 4,56\%$
15	$\frac{0,645}{15} \times 100\% = 4,3\%$	$\frac{0,499}{15} \times 100\% = 3,326\%$
20	$\frac{1,011}{20} \times 100\% = 5,05\%$	$\frac{0,903}{20} \times 100\% = 4,515\%$
setpoint	**Ke kiri**	**Ke kanan**
10	$\frac{0,442}{10} \times 100\% = 4,42\%$	$\frac{0,458}{10} \times 100\% = 4,58\%$
15	$\frac{0,597}{15} \times 100\% = 3,98\%$	$\frac{0,71}{15} \times 100\% = 4,73\%$
20	$\frac{1,214}{20} \times 100\% = 6,07\%$	$\frac{0,908}{20} \times 100\% = 4,54\%$

Dari hasil presentase kesalahan dirata-ratakan berdasarkan *setpoint*. dengan menggunakan Persamaan 6.9 didapatkan hasil sebagai berikut.

$$setpoint\ 10 = \frac{9,86 + 4,56 + 4,42 + 4,58}{4} = 5,85\%$$

$$setpoint\ 15 = \frac{4{,}3 + 3{,}326 + 3{,}98 + 4{,}73}{4} = 4{,}084\%$$

$$setpoint\ 20 = \frac{5{,}05 + 4{,}515 + 6{,}07 + 4{,}54}{4} = 5{,}044\%$$

Maka rata-rata kesalahan sistem adalah sebagai berikut.

$$\overline{error} = \frac{5{,}85 + 4{,}084 + 5{,}044}{3} = 4{,}99\%$$

Sehingga dari pengujian yang telah dilakukan didapatkan hasil kesalahan sebesar 4,99%.

Hasil ukuran waktu yang menyatakan respon telah masuk $\pm 2\%$ dari keadaan *steady state* atau *settling time* pada setiap pengujian dapat dilihat pada Tabel 6.5.

Tabel 6.5 Waktu mencapai kestabilan (s).

Settling time (t_s)				
Gerakan	*Setpoint*			Rata-rata
	10	15	20	
Maju	0,271	0,284	0,686	0,413
Mundur	0,30	0,414	0,354	0,356
Ke kiri	0,397	0,587	0,48	0,488
Kekanan	0,35	0,475	0,378	0,401
Total rata-rata				0,4145

Berdasarkan Tabel 6.5, rata-rata waktu yang dibutuhkan untuk memasuki $\pm 2\%$ dari keadaan *steady state* adalah 0,4145 detik.

BAB 7
PENUTUP

7.1 Kesimpulan

Berdasarkan hasil analisis yang diperoleh melalui pengujian yang dilakukan, maka didapatkan beberapa kesimpulan dari rumusan masalah yang telah ditentukan sebelumnya sebagai berikut.

1. Berdasarkan metode *ziegler nichols* untuk *tuning* parameter PID didapatkan hasil K_p sebesar 0,042, K_i sebesar 0,046 dan K_d sebesar 0,0115.
2. Hasil pengujian jarak tempuh *quadcopter* dengan *setpoint* berbeda-beda menunjukan bahwa sistem kendali dengan menggunakan metode PID memiliki tingkat kesalahan yang rendah yaitu 4,99 %.
3. Waktu yang dibutuhkan sistem untuk memasuki $\pm 2\%$ dari keadaan *steady state* adalah 0,4145 detik.

7.2 Saran

Agar sistem dapat dikembangkan lebih lanjut dengan performa kinerja yang lebih baik, maka diperlukan pengembangan sistem sebagai berikut.

1. Mengembangkan sistem kendali menggunakan kontroler PID untuk pengendalian jarak jauh.
2. Diharapkan pengembangkan pada sistem ini ke depannya dapat mengendalikan ketepatan gerak diantara sudut *pitch* dan *roll* sesuai dengan jarak tempuh yang diinginkan.
3. Sistem dapat dioptimalkan dengan memanfaatkan fungsi kamera pada *quadcopter* untuk mendeteksi objek di lingkungan sekitar *quadcopter* agar dapat bernavigasi dengan lebih baik.

DAFTAR PUSTAKA

Bansal, R., A.Patra, & Bhuria, V. (2012). Design of PID Controller for Plant Control and Comparison with Z-N PID Controller. *International Journal of Emerging Technology and Advanced Engineering , 2*(4).

Caballero, F., Martinez-de-Dios, J., & Maza, I. (2010). Automatic Forest Fire Monitoring and Measurement Using Unmanned Aerial Vehicle, International Conference on Forest Fire Research.

Chairuzzaini, Rusli, M., & Ariyanto, R. (1998). *Pengenalan Metode Ziegler-Nichols pada Perancangan Kontroler pada PID*. Dipetik Juni 1, 2018, dari www.elektroindonesia.com

Hendriawan, A., Utomo, G. P., & Oktavianto, H. (2012). Sistem Kontrol Altitude Pada UAV Model Quadcopter Dengan Metode PID.

Kurniawan, A. P., Mutiara, G. A., & Hapsari, G. I. (2015). Pengiriman Informasi GPS (Global Positioning System) Berupa Teks Melalui Wireless pada AR Drone 2.0. *e-Proceeding of Applied Science, 1*(2).

Lwin, N., & Hla Myo, T. (2014). Implementation Of Flight Control System Based On Kalman And PID Controller For UAV. *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH, 3*(4).

Piskorski, S., Brulez, N., Eline, P., & D'Haeyer, F. (2012). AR.Drone Developer Guide. Dalam *AR.Drone Developer Guide.*

SA, P. (2016). *Parrot Official Website*. Dipetik September 06, 2017, dari Parrot AR Drone 2.0 Elite Edition: https://www.parrot.com/us/drones/parrot-ardrone-20-elite-edition#parrot-ardrone-20-elite-edition

Setyawan, G. E., Setiawan, E., & Kurniawan, W. (2015). SISTEM KENDALI KETINGGIAN QUADCOPTER MENGGUNAKAN PID . *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) , 2*.