TUGAS AKHIR – EE 184801

PERANCANGAN SISTEM PENDARATAN PRESISI *DRONE* PADA *PLATFORM* DARAT BERGERAK BERBASIS VISI KOMPUTER

Dionisius Yose Deofili Abadi
NRP 07111640000109

Dosen Pembimbing
Astria Nur Irfansyah, S.T., M.Eng., Ph.D.
Ronny Mardiyanto, S.T., M.T., Ph.D.

DEPATERMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

TUGAS AKHIR – EE 184801

PERANCANGAN SISTEM PENDARATAN PRESISI *DRONE* PADA *PLATFORM* DARAT BERGERAK BERBASIS VISI KOMPUTER

Dionisius Yose Deofili Abadi
NRP 07111640000109

Dosen Pembimbing
Astria Nur Irfansyah, S.T., M.Eng., Ph.D.
Ronny Mardiyanto, S.T., M.T., Ph.D.

DEPATERMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

Halaman ini sengaja dikosongkan

Institut
Teknologi
Sepuluh Nopember

FINAL PROJECT – EE 184801

DESIGN OF PRECISION LANDING DRONE SYSTEM ON GROUND MOVING PLATFORM BASED ON COMPUTER VISION

Dionisius Yose Deofili Abadi
NRP 07111640000109

Supervisor
Astria Nur Irfansyah, S.T., M.Eng., Ph.D.
Ronny Mardiyanto, S.T., M.T., Ph.D.

ELECTRICAL ENGINEERING DEPARTMENT
Faculty of Electrical and Intelligent Information Technology
Sepuluh Nopember Institute of Technology
Surabaya 2020

Halaman ini sengaja dikosongkan

PERNYATAAN KEASLIAN
TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "**Perancangan Sistem Pendaratan Presisi *Drone* pada Objek Darat Bergerak berbasis Visi Komputer**" adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka. Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, Juli 2020

Dionisius Yose Deofili Abadi
NRP. 0711 16 40000 109

Halaman ini sengaja dikosongkan

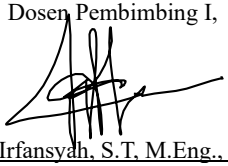
PERANCANGAN SISTEM PENDARATAN PRESISI *DRONE* PADA OBJEK DARAT BERGERAK BERBASIS VISI KOMPUTER

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik
Pada
Bidang Studi Elektronika
Departemen Teknik Elektro
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Menyetujui:

Dosen Pembimbing I,



A. Nur Irfansyah, S.T, M.Eng., Ph.D.
NIP. 198103252010121002

SURABAYA
JULI, 2020

PERANCANGAN SISTEM PENDARATAN PRESISI *DRONE* PADA OBJEK DARAT BERGERAK BERBASIS VISI KOMPUTER

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik
Pada
Bidang Studi Elektronika
Departemen Teknik Elektro
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Menyetujui:

Dosen Pembimbing II,

Ronny Mardiyanto, S.T., M.T., Ph.D.
NIP. 198101182003121003

SURABAYA
JULI, 2020

Halaman ini sengaja dikosongkan

PERANCANGAN SISTEM PENDARATAN PRESISI *DRONE* PADA *PLATFORM* DARAT BERGERAK BERBASIS VISI KOMPUTER

Nama : Dionisius Yose Deofili Abadi
Pembimbing I : Astria Nur Irfansyah, S.T., M.Eng., Ph.D.
Pembimbing II : Ronny Mardiyanto, S.T., M.T., Ph.D.

ABSTRAK

Pendaratan presisi merupakan salah satu manuver dari *drone* yang didefinisikan sebagai sistem pendaratan dimana *drone* mampu mendarat tepat di atas area pendaratan dengan variasi deviasi terkecil yang tidak memiliki nilai kuantitatif. Sistem ini dapat digunakan pada area perairan, konflik ataupun geografis yang rapuh dimana *drone* memiliki potensi hancur ketika mendarat di area tersebut. Sistem ini juga mampu menggantikan peran GPS pada area dimana sinyal satelit GPS lemah.

Pada penelitian yang telah dilakukan, berbagai metode dengan mengkombinasikan GPS dengan sensor lain telah dilakukan tetapi belum dilakukan uji coba pada area pendaratan bergerak. Pada penelitian ini, dikembangkan metode koreksi posisi dengan mengkombinasikan komputer visi dengan sensor jarak untuk melakukan pendaratan presisi pada *platform* darat bergerak. Dengan mengkombinasikan x dan y yang didapat dari pengolahan citra dengan memanfaatkan penghitungan titik berat dan koordinat z dari sensor jarak, maka akan didapatkan sinyal kontrol untuk mengontrol *drone*.

Dari pengujian, didapatkan hasil bahwa *drone* mampu melakukan koreksi posisi dengan baik dengan deviasi minimal hingga 10 cm tetapi akan mendapatkan deviasi besar apabila *drone* kehilangan area pendaratan saat menurunkan ketinggian sehingga area pendaratan menjadi tidak terdeteksi oleh kamera.

Dapat disimpulkan bahwa *drone* mampu melakukan pendaratan presisi pada *platform* darat bergerak dengan kecepatan area pendaratan sekitar 10 cm/detik dengan deviasi rata – rata sebesar 35.25 cm dengan deviasi minimal 10 cm dan deviasi maksimal 103.5 cm.

Kata kunci: *drone*, *platform* darat bergerak, visi komputer

Halaman ini sengaja dikosongkan

DESIGN OF PRECISION LANDING DRONE ON GROUND MOVING PLATFORM BASED ON COMPUTER VISION

Name : Dionisius Yose Deofili Abadi
Supervisor : Astria Nur Irfansyah, S.T., M.Eng., Ph.D.
Co-Supervisor : Ronny Mardiyanto, S.T., M.T., Ph.D.

ABSTRACT

Precision landing is one of drone maneuvers that is determined by a landing system which drone is able to land directly above the landing area with the smallest deviation that has no quantitative value. This system can be used in water areas, conflict area, or fragile geographic areas which dronehave the potential to be destroyed if it is land on such places. This system is also able to replace the role of GPS in areas where GPS satellite signals are weak.

In the recent research, various methods by combining GPS with other sensors have been done but have not been tested in the mobile landing area. In this research, a position correction method was developed with combining computer vision and proximity sensor to perform precision landing on moving platform. By combining x and y form image processing by utilizing the calculation of center of gravity an z coordinate from proximity sensor, a control signal will be obtained to control the drone.

From the test, the result are obtained that the drone is able to make a good position correction with minimum deviation up to 10 cm but will get a large large deviation if the drone lost the sight of landing area so that the landing area cannot be tracked by the camera.

It can be concluded that the drone is able to make a precision landing on a moving platform with a landing area velocity about 10 cm/sec with an average deviation of 35.25 cm with a minimum deviation of 10 cm and a maximum deviation of 103.5 cm.

Keywords: drone, ground moving platform, computer vision

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji dan syukur kepada Tuhan YME karena dengan berkat dan limpahan rahmat yang diberikan, penulis dapat menyelesaikan laporan tugas akhir dengan judul "**PERANCANGAN SISTEM PENDARATAN PRESISI *DRONE* PADA *PLATFORM* DARAT BERGERAK BERBASIS KOMPUTER VISI"**, sebagai salah satu persyaratan akademik untuk menyelesaikan pendidikan Strata-Satu di Departemen Teknik Elektro, Fakultas Teknologi Elektro dan Informatika Cerdas, Intitut Teknologi Sepuluh Nopember.

Dalam penyusunan dan penyelesaian laporan Tugas Akhir ini, penulis mendapatkan banyak sekali doa, bantuan dan dukungan dari berbagai pihak. Untuk itu penulis mengucapkan terima kasih sebesar – besarnya kepada:

1. Tuhan YME yang telah memberikan rahmat keselamatan, kesehatan dan rejeki sehingga mampu menyelesaikan Tugas Akhir ini di tengah pandemi yang melanda.
2. Orang tua, Bapak Yohanes Totok Herqutanto dan Wiwik Septiningsih yang telah mendoakan dan selalu memberi semangat dan kasih saying luar biasa kepada penulis
3. Seluruh keluarga besar penulis yang selalu memberikan semangat dan doa untuk menyelesaikan Tugas Akhir ini.
4. Bapak Astria Nur Irfansyah, S.T., M.Eng., Ph.D. selaku dosen pembimbing 1 atas gagasan topik tugas akhir serta bimbingan dan arahan dalam menyelesaikan tugas akhir ini.
5. Bapak Ronny Mardiyanto, S.T., M.T., Ph.D. selaku dosen pembimbing 2 atas gagasan topik tugas akhir serta bimbingan dan arahan dalam menyelesaik tugas akhir ini.
6. Seluruh Dosen dan Tendik di Departemen Teknik Elektro Institut Teknologi Sepuluh Nopember yang telah membantu penulis di masa kehidupan kampusnya.
7. Teman – teman bidang studi Elektronika: Hamid, Salik, Mar'ie, Syarif, Revo, Dzulfikar, Herline, Addin, Satria yang telah membantu dan memberikan semangat kepada penulis selama masa perkuliahan di Departemen Teknik Elektro ITS
8. Teman Robotika ITS: Rasyid yang telah memberikan ide dan masukan serta menemani penulis di dalam pengambilan data dan

pengerjaan tugas akhir ini.

9. Teman – teman tim Bayucaraka ITS, khususnya tim Soeromiber yang telah membantu ide dan komponen untuk melengkapi kebutuhan tugas akhir ini.
10. Teman di kost penulis: Rum yang bersedia menemani dan membantu penulis selama mengerjakan tugas akhir di masa pandemi ini.

Penulis menyadari bahwa masih banyak yang dapat dikembangkan dan diperbaiki di dalam tugas akhir ini. Oleh karena itu, penulis menerima setiap masukan dan kritik yang diberikab. Semoga tugas akhir ini dapat memberikan manfaat bagi banyak pihak.

Surabaya, 1 Juli 2020

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Halaman ini sengaja dikosongkan

BAB 1
PENDAHULUAN

Tugas akhir ini merupakan suatu penelitian yang dilakukan sebagai syarat akademik untuk mendapatkan gelar sarjana teknik di Institut Teknologi Sepuluh Nopember (ITS) Surabaya. Topik yang dibahas pada tugas akhir ini adalah pendaratan presisi *drone* pada objek darat bergerak berbasis komputer visi.

Bab ini akan menjelaskan mengenai hal – hal yang berkaitan tentang permasalahan yang diambil untuk topik ini. Hal – hal tersebut meliputi latar belakang, perumusan masalah, tujuan penulisan, batasan masalah, metodologi penelitian, sistematika penulisan, dan relevansi.

1.1. Latar Belakang

Di masa sekarang ini, *Unmanned Aerial Vehicle* (UAV) atau pesawat tanpa awak merupakan salah satu teknologi yang banyak dikembangkan oleh para peneliti. *Drone*, sebutan lain dari UAV, sendiri pada umumnya memiliki ukuran yang kecil dan mampu melakukan navigasi di berbagai kondisi geografis. Hal inilah yang membuat *drone* menjadi solusi alternatif yang digunakan untuk mencapai tempat – tempat yang sulit dijangkau oleh manusia.

Pada area bencana seperti area rawan tanah longsor ataupun area geografis setelah terkena gempa bumi, biasanya susunan permukaan tanahnya memiliki struktur yang rapuh. Dalam hal seperti ini diperlukan suatu wahana yang efisien, kecil dan memiliki teknologi yang mampu membantu di dalam pencarian korban atau target lain yang perlu dicari. Untuk itu, *drone* dijadikan salah satu alternatif wahana yang digunakan untuk melakukan navigasi di dalam misi *search and rescue* (SAR)[1]. Tetapi, di dalam melakukan navigasinya, *drone* harus terus terkoneksi dengan *Ground Control Station* (GCS) untuk terus melakukan pemantauan terhadap *drone* sehingga *drone* peru berada pada jangkauan GCS. Dalam hal ini, apabila GCS maupun area pendaratan berada di satu tempat dalam jangka waktu yang lama, maka area di sekitar GCS dan area pendaratan di dekat GCS memiliki kemungkinan untuk longsor karena memiliki struktur tanah yang rapuh. Untuk itu, GCS dan area pendaratan perlu terus bergerak sehingga dapat mengurangi potensi longsornya tempat GCS dan *drone* dapat terus dipantau. Kejadian lain dimana GCS

dan area pendaratan akan terus bergerak adalah pada area perairan khususnya laut, dimana area GCS dan area pendaratan pasti akan terus menerus diguncang oleh gelombang laut.

Pada area yang terdapat jumlah massa yang banyak, diperlukan pula suatu sistem *crowd monitor* atau pemantauan keramaian yang memiliki fungsi untuk memantau keamanan publik, menjaga keamanan dan hal lain terkait massa dengan jumlah yang banyak[2]. Untuk itu, diperlukan suatu area pendaratan bagi *drone* untuk mengganti baterai ataupun hal lain yang membutuhkan pendaratan yang terus menerus bergerak. Hal ini dilakukan untuk menghindari kerusakan area pendaratan karena massa yang secara sengaja maupun tidak sengaja menghancurkan area pendaratan.

Dalam melakukan navigasinya, *drone* dibedakan menjadi dua macam yaitu *manual control drone* dan *autonomous drone.* Pada *manual control drone, drone* dikendalikan secara manual oleh pilot melalui *remote control* yang tersambung pada *receiver* yang terpasang pada *drone*. Sedangkan pada *autonomous drone*, *drone* dikendalikan oleh berbagai macam sensor untuk melakukan navigasi secara otomatis atau *autonomous*. Pada umumnya, *autonomous drone* menggunakan sistem GPS (*Global Positioning System*) dalam melakukan navigasi. Dalam hal ini, *drone* akan melakukan misi sesuai dengan *waypoint* yang telah diberikan berdasarkan GPS[3]. Selain itu, GPS digunakan untuk menentukan posisi *drone* berdasarkan satelit yang terdeteksi[4].

Salah satu kelemahan dari navigasi dengan menggunakan GPS adalah, GCS harus memiliki data satelit yang cukup untuk menghindari *loss signal* dan penangkapan sinyal satelit yang buruk. Permasalahan dimana *GPS* menangkap sinyal satelit yang buruk adalah ketika cuaca mendung, *drone* berada di dalam ruangan dan *drone* berada pada hutan yang sangat lebat sehingga sinyal satelit tidak dapat ditangkap dengan baik. Pada umumnya, apabila hal ini terjadi terdapat suatu sistem *fail-safe* seperti pendaratan langsung. Selain itu, juga dikembangkan metode lain dengan mengkombinasikan antara sensor magnetik, komputer visi dan GPS sehingga *drone* mampu melakukan navigasi ketika GPS sedang mengalami masalah[5].

Saat ini, dikembangkan pula metode navigasi *drone* tanpa menggunakan bantuan GPS (*Global Positioning System)*. Pada sistem ini, perangkat utama navigasi adalah kamera yang

digunakan untuk melakukan deteksi objek dan dilakukan pengolahan sehingga menghasilkan sinyal kontrol untuk melakukan navigasi pada *drone*[6]. Sistem deteksi objek kemudian dikembangkan menjadi sistem pendaratan presisi dengan mengandalkan visi komputer dengan menggunakan sensor kamera dengan ketelitian 0.0137 m[7]. Sensor inframerah ditambahkan pada area pendaratan dan *drone* mendeteksi area pendaratan dengan kamera dan melakukan pendaratan di area tersebut[8]. Di dalam membantu proses navigasi *drone* dengan menggunakan komputer visi, *drone* ditambahkan dengan berbagai macam sensor untuk melakukan koreksi ketinggian. Salah satu contoh sensor ketinggian adalah sensor *rangefinder* yang memiliki keunggulan apabila dibandingkan dengan sensor barometer karena dapat digunakan dalan *indoor*[9].

Pada sistem yang dikembangkan oleh Situmorang, dkk[10], dikembangkan suatu metode pendaratan presisi dengan mengkombinasikan GPS dan visi komputer. Sistem ini memiliki kelemahan karena kecepatan pergerakan yang konstan ketika melakukan koreksi posisi sehingga tidak mampu melakukan koreksi posisi pada area pendaratan yang bergerak. Untuk itu, dikembangkan suatu sistem pendaratan presisi yang memiliki koreksi posisi untuk dapat melakukan pendaratan presisi pada *platform* darat bergerak. Sistem pendaratan presisi dapat dirancang menggunakan komponen – komponen visi komputer untuk menentukan posisi relatif dari *drone* terhadap area pendaratan dan sensor ketinggian untuk melakukan koreksi ketinggian. Pendaratan presisi sendiri tidak memiliki ukuran yang ditentukan berdasarkan nilai kuantitatif. Tetapi dapat didefinisikan sebagai suatu sistem pendaratan dimana *drone* mampu mendarat dengan tepat diatas area pendaratan dan variasi deviasi terkecil[8], [10]. Kelebihan dari sistem ini adalah kamera yang dapat digunakan baik *indoor* maupun *outdoor* dan sensor ketinggian yang mampu melakukan pendeteksian yang lebih akurat. Selain itu, dengan melakukan penambahan kontrol PID di dalam melakukan koreksi posisi akan membuat *drone* melakukan koreksi posisi secara presisi pada area pendaratan. Diharapkan, dengan pengembangan sistem integrasi visi komputer, *drone* mampu melakukan pendaratan pada area pendaratan bergerak dengan geografis yang sulit dilalui tanpa mengandalkan sistem navigasi GPS yang baik.

1.2. Perumusan Masalah

Permasalahan yang dibahas dalam tugas akhir ini adalah:

1. Sistem navigasi pendaratan pada *drone* yang mampu menggantikan peran *Global Positioning System* dalam melakukan koreksi posisi.
2. Metode integrasi antara sensor jarak dan kamera untuk menentukan titik pendaratan secara presisi.
3. Metode pendaratan *drone* secara presisi pada *platform* darat bergerak.

1.3. Tujuan Penelitian

Penelitian pada tugas akhir ini bertujuan sebagai berikut:

1. Implementasi visi komputer sebagai pengganti metode koreksi posisi di dalam navigasi pendaratan pada *drone*.
2. Implementasi pengolahan citra dan pendeteksian ketinggian dengan sensor jarak untuk menentukan titik tengah area pendaratan.
3. Implementasi kontrol untuk melakukan pendaratan pada *platform* darat bergerak.

1.4. Batasan Masalah

Batasan masalah dari tugas akhir ini adalah sebagai berikut:

1. Area pendaratan dan *drone* tidak saling komunikasi.
2. Sistem bekerja pada kondisi pencahayaan optimal yang dilakukan di luar ruangan *(outdoor*) pada rentang waktu pagi hingga sore hari.
3. *Drone* diarahkan ke daerah landasan pendaratan secara manual.
4. Area pendaratan dapat digerakkan secara manual menggunakan *remote control* maupun dengan tangan.
5. Desain area pendaratan yang disesuaikan untuk *drone* mendarat untuk mempermudah pendaratan presisi.

1.5. Metodologi Penelitian

Langkah-langkah yang dikerjakan pada tugas akhir ini adalah sebagai berikut:

1. Studi literatur

Studi literatur berfungsi untuk pengumpulan sumber informasi, dasar, teori, dan data – data yang dapat mendukung keabsahan tugas akhir ini. Literatur yang dicari berupa buku, jurnal, artikel maupun *paper* yang memiliki keabsahan sehingga dapat dijadikan referensi. Tujuan dari studi literatur adalah untuk memperkuat permasalahan dan sebagai landasan teori untuk melakukan pemecahan – pemecahan permasalahan yang ada.

2. Perakitan dan Pemrograman *Drone*

Perakitan dan pemrograman *drone* merupakan perancangan *drone* berupa *quadcopter* baik dari segi mekanik, segi elektronik maupun dari segi pemrograman misi. Dari sisi mekanik, *drone* harus memiliki desain dan ukuran yang mampu menunjang misi. Sedangkan dari sisi elektronik, *drone* dirancang dengan menggunakan komponen – komponen berupa sensor untuk menerbangkan *drone* dengan stabil. Dalam tugas akhir ini, sensor ketinggian akan diintegrasikan dengan *raspberry pi* yang berfungsi untuk mengolah gambar dan mengirimkan perintah ke *flight controller*. Setelah itu, melakukan pemrograman algoritma untuk menentukan bagaimana *drone* akan bergerak dan mengirimkan datanya pada *flight controller*. Pemograman *drone* sendiri akan terus dilakukan hingga, *drone* mampu melakukan pendaratan secara optimal.

3. Pengujian dan Pengambilan Data

Pada tahap pengujian, *drone* akan diuji agar mampu terbang dengan stabil menggunakan metode yang ada dan mampu mempertahankan posisi dengan baik. Selanjutnya, *drone* akan diuji untuk dapat melakukan pendaratan presisi dengan menggunakan citra dan menguji kecepatan pendeteksian dan pendaratan dan dibandingkan hasilnya. Dari hasil pengujian, akan didapatkan data – data yang dapat dianalisa untuk mengetahui karakteristik dari *drone* sehingga sistem yang diuji dapat dievaluasi.

4. Penulisan Laporan Tugas Akhir

Penulisan laporan dilakukan beriringan sesuai dengan tahap – tahap tugas akhir yang lain. Laporan ini berfungsi untuk mendapatkan kesimpulan dari seluruh data – data yang telah diambil untuk mendapatkan jawaban dari topik permasalahan yang dianalisa.

1.6. Sistematika Penulisan

Dalam buku tugas akhir ini, pembahasan mengenai sistem yang

dibuat terbagi menjadi lima bab dengan sistematika penulisan sebagai berikut:

- **BAB I : Pendahuluan**
  Bab ini meliputi penjelasan latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi, sistematika penulisan, dan relevansi.
- **BAB II : Tinjauan Pustaka**
  Bab ini berisi tentang teori-teori yang mampu menunjang penelitian berdasarkan penelitian – penelitian yang telah dikerjakan. Selain itu juga menjelaskan setiap komponen perangkat keras maupun perangkat lunak yang digunakan dalam tugas akhir ini.
- **BAB III : Perancangan Sistem**
  Bab perancangan sistem akan membahas sistem yang digunakan pada realisasi tugas akhir. Dari segi perangkat lunak maupun perangkat keras serta algoritma yang berjalan.
- **BAB IV : Pengujian dan Analisis**
  Pada bab ini menguraikan tentang pengujian sistem pada *drone.*
- **BAB V : Penutup**
  Bab ini berisi tentang kesimpulan yang diperoleh dari sistem yang telah dibuat serta saran untuk pengembangan sistem lebih lanjut.

1.7. Relevansi

Hasil dari tugas akhir ini diharapkan mampu membantu pekerjaan manusia di bidang transportasi khususnya dalam pengantaran paket ke area bencana[1], [11]. Selain itu, *drone* mampu melakukan *monitoring* pada area – area yang sulit dijangkau manusia maupun area – area konflik yang menyulitkan untuk dilakukan pengawasan[2].

Halaman ini sengaja dikosongkan

BAB 2
TINJAUAN PUSTAKA DAN DASAR TEORI

Pada bab memaparkan tinjauan pustaka dan dasar teori pendukung. Teori-teori ini menjadi dasar dalam penyusunan laporan tugas akhir ini. Tinjauan pustaka memuat teori dan penelitian yang telah dilakukan sebelumnya. Teori tersebut digunakan untuk menemukan solusi dalam permasalahan pada tugas akhir ini.

2.1. Tinjauan Pustaka

Unmanned Aerial Vehicle (UAV) atau *drone* saat ini memainkan peran krusial di dalam kehidupan sehari – hari dimana robot dapat mengoperasikan dan membantu di dalam pelaksanaan tugas. Salah satu jenis *drone* yang digunakan adalah *quadcopter* yang merupakan jenis wahana yang mampu melakukan *vertical take-off and landing* (VTOL), yang memiliki empat motor, diposisikan dengan jarak yang sama satu sama lain dan parallel terhadap tanah. Secara umum, *quadcopter* dengan tipe *frame* X dipilih karena memiliki kekuatan yang cukup untuk menahan perubahan karena beban yang diangkat[12]. Aplikasi dari wahana ini antara lain adalah pada pertanian, industri, dan militer. Pada umumnya, *drone* melakukan navigasi *autonomous* menggunakan *Global Positioning System* (GPS), akan tetapi tidak seluruh daerah yang akan dilewati *drone* memiliki cakupan GPS yang baik sehingga diperlukan suatu sistem yang mampu melakukan navigasi dan meningkatkan akurasi ketika cakupan GPS buruk[5]. Untuk itu, dikembangkan pula sistem untuk membantu *drone* melakukan pendaratan dengan menggunakan pengolahan citra komputer.

2.1.1.Pengolahan Citra pada Drone

Pengolahan citra pada *drone* merupakan suatu topik yang sudah banyak diaplikasikan. Pengolahan citra pada *drone* umumnya digunakan sebagai komponen untuk membantu navigasi *drone*[6],[12],[13], sebagai komponen untuk membantu pendaratan[7], dan membantu *drone* untuk melakukan menghindari rintangan atau *obstacle avoidance*[15]. Berikut ini merupakan penelitian – penelitian yang pernah dilakukan berkaitan dengan pengolahan citra pada *drone*.

1. *Visual-based Tracking and Control Algorithm Design for Quadcopter UAV*[6]

Pada penelitian yang dilakukan oleh Qin dan Wang, dikembangkan suatu metode untuk melakukan navigasi dengan mengkombinasikan antara metode Kalman Filter dan algoritma *Camshift* untuk mencari titik tengah dari suatu objek yang di *track*. Algoritma *camshaft* menggunakan histogram warna dari target untuk diubah menjadi map distribusi probabilitas warna, mengukur ukuran dan posisi dari *frame* dan mengestimasi posisi dari target pada *frame*.

2. *A Simple Visual Navigation System for an UAV*[13]

   Metode yang dikembangkan oleh Krajnik, dkk menggunakan *monocular camera* atau kamera berlensa satu untuk melakukan navigasi pada *drone*. Metode ini dilakukan dengan teknik *"record and replay"* dimana fase *mapping* (*"record"*) dilakukan dengan cara menuntun *drone* mengikuti suatu jalur yang akan diikuti secara *autonomous*. Ketika fase ini, algoritma navigasi mencari dan melakukan *track* fitur yang menonjol pada gambar yang ditangkap oleh kamera pada *drone*. Ketika navigasi secara *autonomous* diinisiasi, *drone* akan mengikuti jalur sesuai dengan cara membandingkan antara fitur yang tertangkap oleh kamera dengan fitur yang telah dipelajari. Kelemahan dari metode ini, adalah hanya mampu melakukan navigasi pada area yang telah ditentukan sebelumnya.

3. *An Autonomous UAV with an Optical Flow Sensor for Positioning and Navigation*[14]

   Metode yang dikembangkan oleh Gageik, Sthrohmeier dan Montenegro menggunakan sensor *optical flow* untuk melakukan navigasi pada *drone*. Sensor *optical flow* memiliki fungsi untuk melakukan pemosisian 2D dari *drone* sehingga *drone* dapat melakukan navigasi. Metode *optical flow* yang digunakan adalah metode Lucas-Kanade yang membandingkan posisi antara *frame* saat ini dengan *frame* sebelumnya. Selain itu dengan menggunakan algoritma Srinivasan yang digunakan untuk menyederhanakan intensitas dari piksel dan membandingkannya dengan piksel sebelumnya dan diskala antara piksel tetangga. Dengan menggunakan metode ini didapatkan error posisi sebesar 10 cm dan error posisi 30 cm setelah mendarat.

4. *Automatic Quadcopter Control Avoiding Obstacle Using Camera with Integrated Ultrasonic Sensor*[15]

   Pada metode deteksi halangan yang diajukan oleh Anis, dkk menggunakan kamera yang diintegrasikan dengan sensor ultrasonik.

Kamera digunakan untuk melakukan pengolahan citra dengan berbagai tingkatan yaitu *filtering*, deteksi fitur, dan *optical flow*. Di dalam penelitian ini, metode deteksi Shi and Tomaso digunakan karena memiliki jangkauan deteksi dan rasio dari *frame* yang mengandung target yang dideteksi lebih baik. Metode Lukas-Kanade digunakan untuk proses *optical flow* untuk mendeteksi pergerakan dari area yang dideteksi. Dalam *filtering* atau pra pemrosesan gambar, peneliti dalam penelitian ini menggunakan metode pembagian kuadran untuk membagi fitur yang terdeteksi pada tiap kuadran. Dari hasil pembagian kuadran ini, maka *drone* dapat menentukan perintah yang cocok untuk fitur yang terdeteksi di tiap kuadran.

2.1.2.Penelitian terkait Pendaratan Presisi

Drone yang mampu melakukan navigasi secara *autonomous* perlu dilengkapi dengan suatu sistem pendaratan presisi. Pendaratan presisi sendiri memiliki definisi yang bermacam – macam. Nguyen, dkk[10] menjelaskan bahwa UAV yang mampu melakukan pendaratan presisi ketika posisinya berada diatas area pendaratan pada *threshold* tertentu, maka *drone* akan menghentikan motor sehingga disebut dengan manuver pendaratan. Janousek dan Marcon[8] menjelaskan bahwa suatu sistem wahana tanpa awak harus selalu dituntun untuk mendarat pada titik yang sama pada area pendaratan dengan variasi deviasi sekecil mungkin. Sesuai dari pernyataan tersebut maka tidak adanya nilai kuatitatif dari pendaratan presisi sendiri. Sehingga dapat disimpulkan bahwa berbagai macam metode dilakukan untuk meningkatkan akurasi dari pendaratan presisi. Pendaratan presisi sendiri memiliki fungsi dalam berbagai bidang seperti pendaratan paket, misi pengintaian ataupun misi penyerangan. Berikut ini beberapa penelitian yang berkaitan dengan pendaratan presisi pada *drone*.

1. *A Precise and GNSS-Free Landing System on Moving Platforms for Rotary-Wing UAVs*[16]

   Di dalam penelitian yang diajukan oleh Alarcón, dkk, *drone* akan berusaha melakukan navigasi tanpa GNSS dengan cara mengimplementasikan konsep posisi relatif dan kecepatan antara kendaraan udara dan area pendaratan dapat dihitung berdasarkan sudut dari kabel yang terkoneksi secara fisik ke *drone* dan melakukan pendaratan. Dengan menggunakan metode ini, juga data memfasilitasi *drone* untuk menengahkan ke area pendaratan selain meningkatkan akurasi dari *drone* tersebut. Di dalam pengujian

metode ini, didapatkan hasil pendaratan presisi dengan akurasi kurang dari 30 cm dengan akurasi kecepatan di bawah 0.25 m/s.

2. *A Vision-Based Approach for Unmanned Aerial Vehicle Landing*[7]

   Dalam sistem pendaratan presisi secara *autonomous* yang dikerjakan oleh Patruno, dkk, dikembangkan suatu metode dengan sistem komputer visi berbasi rekognisi objek seperti yang telah dijelaskan pada subbab sebelumnya. *Drone* menggunakan referensi dari posisi kamera dan posisi area pendaratan untuk melakukan kalkulasi estimasi posisi dari *drone*. Dengan menggunakan metode ini, pendaratan presisi memiliki akurasi hingga 0.0137 m dengan sudut 1.04° pada posisi dan orientasi *drone*.

3. *Automatic Navigation and Landing of an Indoor AR Drone Quadrotor using ArUco Marker and Inertial Sensors*[17]

   Dalam penelitian yang dilakukan oleh Sani dan Karimian, *drone* juga dapat melakukan navigasi pendaratan dengan mengkombinasikan deteksi pola ArUco dan sensor inertia. Dengan menggunakan pola ArUco, *drone* dapat melakukan estimasi posisi secara 3D dengan mengkombinasikan antara posisi relatif kamera dengan posisi dari pola ArUco. Kecepatan, akselerasi, dan sudut dari pergerakan *drone* diberikan oleh sensor inertia yang terpasang pada *drone*. Dengan menggunakan Kalman Filter, maka pengukuran yang bersifat *noise* dapat dikurangi. Dengan menggunakan metode ini, maka didapatkan hasil akurasi dengan error maksimal 6 cm.

4. *Autonomous Vision-based Target Detection and Safe Landing for UAV*[18]

   Di dalam penelitian yang dilakukan oleh Rabah, dkk, menggunakan metode dengan mengkombinasikan antara pengolahan citra dengan sensor *laser rangefinder*. Untuk mendeteksi target pendaratan menggunakan metode *thresholding* dengan mengubah gambar dari RGB ke HSV. Berikutnya, *drone* akan melakukan pendaratan dengan memanfaatkan logika Fuzzy. Kontrol posisi yang dilakukan memanfaatkan informasi posisi yang didapatkan oleh *drone* berdasarkan oleh deteksi area pendaratan. Dengan menggunakan metode ini, maka *drone* berhasil melakukan pendaratan dengan baik dari ketinggian 2.5 m dengan waktu respon sekitar 33.5 detik dan akurasi kurang dari 1 m.

5. *Quadcopter Adaptive Trajectory Tracking Control: A New Approach Via Backstepping Technique*[10]

   Di dalam penelitian yang yang diajukan oleh Nguyen, dkk,

dengan memanfaatkan *Ultra-wideband* (UWB) untuk melakukan lokalisasi dan mendekati area pendaratan atau *home station*. Pada area pendaratan dipasang empat buah UWB *beacon* dan pola area pendaratan di tengahnya. Di dalam *drone* sendiri terpasang sensor UWB untuk mendeteksi *beacon*, sensor inertia, dan kamera yang mengarah ke bawah untuk mendapatkan data visual. Pola area pendaratan yang digunakan di dalam penelitian ini adalah AprilTag yang merupakan penanda yang dapat digunakan dalam berbagai kondisi pencahayaan. Untuk kontrol dari *drone* sendiri menggunakan kontrol posisi yang didapatkan dari pendeteksian AprilTag yang dikombinasikan dengan sistem kontrol PID. Dengan menggunakan metode ini, *drone* memiliki akurasi pendaratan sekitar 20 cm.

6. *Precision Landing Option in Unmanned Aerial Vehicles*[8]

   Pada penelitian yang dilakukan oleh Janousek dan Marcon menambahkan kamera infrared untuk melakukan deteksi terhadap sinar infrared. Dengan menggunakan metode ini, *drone* tidak memerlukan adanya transfer data antara area pendaratan dan *drone* sendiri. Dengan memanfaatkan metode ini, didapatkan akurasi *drone* sekitar 30 cm.

7. *Precision Landing for Drone based on Computer Vision*[10]

   Pada penelitian yang dilakukan oleh Situmorang, dkk, dikombinasikan GPS dengan visi komputer untuk melakukan pendaratan presisi. Dalam sistem ini, dilakukan pencarian titik tengah dengan mencari koordinat x dan y serta melakukan koreksi posisi setelah *drone* diarahkan ke *waypoint* dengan menggunakan GPS. Dengan menggunakan metode ini, didapatkan rata – rata error posisi sebesar 0.45 cm.

Tabel 2.1 Rangkuman pendaratan presisi penelitian sebelumnya

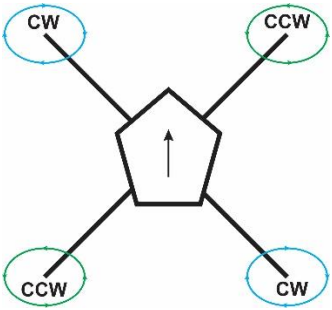
No	Makalah	Error posisi pendaratan presisi
1	A Precise and GNSS-Free Landing System on Moving Platforms for Rotary-Wing UAVs[16]	Kurang dari 30 cm
2	A Vision-based Approach for Unmanned Aerial Vehicle Landing[7]	Rata – rata 0.0137 m
3	Automatic Navigation and Landing of An Indoor AR Drone Quadrotor using ArUco Marker and Inertial Sensors[17]	Maksimal 6 cm

4	Autonomous Vision-based Target Detection and Safe Landing for UAV[18]	Kurang dari 1 m
5	Post-Mission Autonomous Return and Precision Landing of UAV[19]	Rata – rata 20 cm
6	Precision Landing Options in Unmanned Aerial Vehicles[8]	Rata – rata 30 cm
7	Precision Landing for Drone based on Computer Vision[10]	Rata – rata 0.45 cm

2.2. Drone

Unmanned Aerial Vehicle (UAV) atau pesawat tanpa awak merupakan pesawat udara yang pengoperasiannya tidak membutuhkan pilot di dalam pesawat atau dapat dikendalikan secara otomatis. UAV, atau biasa disebut *drone*, dibuat pertama kali ketika Perang Dunia I yang dikendalikan oleh radio kontrol. Pada saat ini *drone*, telah dikendalikan secara otomatis tanpa menggunakan pilot sama sekali. Hal ini dapat dilakukan dengan mengintegrasikan komponen – komponen sensor di dalamnya. Salah satu komponen yang berperan penting untuk melakukan navigasi adalah GPS (*Global Positioning System*). GPS memiliki fungsi untuk menentukan posisi koordinat dari suatu *drone* dengan cara mencocokkan dengan sinyal satelit. Salah satu kelemahan sistem ini adalah ketika GPS tidak mampu menangkap sinyal satelit dikarenakan satelit yang ada di daerah tersebut sedikit atau memang area tersebut memiliki banyak sinyal yang dapat mengganggu GPS. Hal ini dapat diatasi dengan menambahkan sensor magnetik dan sensor visual.

Salah satu jenis *drone* yang saat ini sering digunakan adalah *quadcopter*. *Quadcopter* merupakan salah satu jenis multirotor yang memiliki dua set propeller dengan *pitch* yang identik satu sama lain, masing – masing adalah *clockwise* (CW) atau *counter-clockwise* (CCW). Konfigurasi dari *quadcopter* sendiri adalah satu berpasangan dengan arah yang sebaliknya untuk menghilangkan fungsi dari *tail rotor* pada helikopter[19].

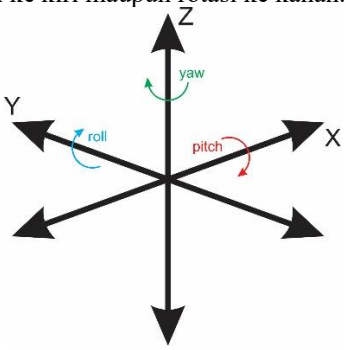


Gambar 2.1. Konfigurasi *Quadcopter*

Pada gambar 2.1 menggambarkan tentang konfigurasi X pada *drone* berjenis *quadcopter*. *Quadcopter* sendiri hanya memiliki enam DOF (*Degree of Freedom*), tetapi karena struktur yang saling berkebalikan, *quadcopter* mampu melakukan beberapa gerakan dasar yang memungkinkan *quadcopter* untuk terbang dan mencapai ketinggian tertentu. Pada gambar 2.2 menggambarkan 6 DOF dari *quadcopter.* Gerakan dasar tersebut adalah:

- *Throttle*, merupakan gerakan yang memiliki fungsi untuk meningkatkan maupun menurunkan kecepatam dari *propeller*. Dengan adanya gerakan ini, *quadcopter* melakukan gerakan secara vertikal sehingga menyebabkan ketinggian dari *drone* meningkatkan maupun menurun.
- *Roll,* merupakan gerakan yang memiliki fungsi untuk meningkatkan (atau menurunkan) kecepatan dari semua *propeller* di salah satu sisi kanan (atau kiri) dan menurunkan (atau meningkatkan) kecepatan *propeller* di sisi lainnya. Dengan adanya gerakan ini, *drone* akan mengalamai perputaran baik ke kiri maupun ke kanan sesuai dengan sisi *propeller* yang diturunkan kecepatannya. Selain itu, dengan adanya gerakan ini, *drone* juga mampu melakukan gerakan geser ke samping.
- *Pitch*, merupakan gerakan yang memiliki fungsi untuk meningkatkan (atau menurunkan) kecepatan dari semua *propeller* di salah satu sisi depan dan menurunkan (atau meningkatkan) kecepatan di sisi belakang. Dengan adanya gerakan ini, *drone* akan mengalami perputaran ke depan maupun ke belakang. Selain itu, dengan adanya gerakan ini, *drone* juga mampu melakukan gerakan maju ataupun mundur.
- *Yaw,* merupakan gerakan yang memiliki fungsi untuk meningkatkan

(atau menurunkan) kecepatan *propeller* pasangan depan-belakang dan menurunkan (atau meningkatkan) kecepatan *propeller* pasangan kiri-kanan. Dengan adanya gerakan ini, *drone* akan mengalami rotasi ke kiri maupun rotasi ke kanan.



Gambar 2.2. Skema Pergerakan *Quadcopter*

Dari uraian yang telah dijabarkan, *drone* dengan konfigurasi *quadcopter* memiliki banyak keunggulan seperti kompleksitas mekanik yang minimal, anggaran pembuatan yang relatif murah, augmentasi beban yang dapat diatur dengan mudah dan tidak memerlukan variasi *pitch* dari propeller[12].

2.2.1.MAVLink

Micro Air Vehicle Link (MAVLink) merupakan protokol komunikasi yang biasa digunakan untuk wahana tanpa awak. MAVLink bekerja dengan cara bertukar *message* antara *ground station* dengan sistem wahana tanpa awak[20]. Berdasarakan Koubaa, dkk[21], di dalam sistem MAVLink sendiri, *message* terbagi menjadi dua kelas yaitu *state messages* dan *command messages*. *State message* merupakan suatu jenis *message* atau pesan yang dikirim dari sistem wahana tanpa awak menuju ke *ground station* yang berisi tentang informasi mengenai kondisi wahana, seperti ID, lokasi, kecepatan dan ketinggian. Sedangkan, *command messages* merupakan pesan yang dikirimkan dari *ground station* ke wahana tanpa awak yang berisi perintah untuk melaksanakan misi seperti menuju ke *waypoint*, *takeoff*, maupun *landing*.

State messages diklasifikasikan menjadi dua lagi yaitu *heartbeat messages* yang memiliki fungsi untuk mengindikasikan kondisi keaktifan GCS (*Ground Control Station*) dengan cara mengirimkan *heartbeat message* secara periodik, *system status message* yang memiliki fungsi

untuk melakukan pengecekan kondisi seluruh sensor yang terdapat pada wahana, dan *global position message* yang berisikan tentang posisi 3D wahana berupa *latitude, longitude,* dan *absolute altitude*[21].

Command messages memiliki beberapa pesan yang memeberikan kemampuan kepada sistem wahana untuk melakukan suatu misi. Pada *command messages* sendiri terdapat dua komponen pesan di dalamnya yaitu *COMMAND_LONG* yang memiliki ID hingga 76 dan didefinisikan dengan 11 bidang dan *mission item command* yang memiliki ID hingga 39 dan memiliki fungsi untuk memberikan perintah kepada wahana untuk melakukan navigasi. Gambar 2.3 merupakan logo dari MAVLink.

Gambar 2.3. Logo MAVLink

2.2.2.Dronekit

Dronekit-Python merupakan suatu *library* yang bersifat *open source* dan memiliki fungsi untuk melakukan komunikasi dengan wahana tanpa awak melalui protokol MAVLink. *Open source* memiliki arti bahwa *library* ini dapat digunakan dan dikembangkan oleh siapa saja. Dronekit-Python mempermudah penggunaan MAVLink dengan memberikan koneksi dengan wahana berupa *script*, mendapatkan dan menentukan kondisi dan informasi parameter, memberikan notifikasi *asynchronous* ketika terjadi perubahan, memberikan informasi misi ke UAV, dan mampu untuk melakukan *channel override*. Gambar 2.4 merupakan logo dronekit.

Gambar 2.4. Logo Dronekit

2.3. Komputer Visi

Komputer visi (*computer vision*) merupakan transformasi data dari

gambar diam maupun gambar bergerak menjadi suatu keputusan maupun representasi baru[22]. Contoh keputusan yang dimaksud seperti menutup pintu ketika warna biru ditampilkan dan contoh dari representasi beru adalah gambar berwarna menjadi gambar dengan skala warna abu – abu. Di dalam komputer visi sendiri, terdapat pengolahan citra sebagai proses awal (*preprocessing*) sebelum gambar diolah lebih lanjut. Hal ini dilakukan untuk memperbaiki kualitas citra agar dapat dengan mudah diintepretasikan oleh manusia dan komputer seperti pemampatan citra (*image compression*), pemugaran citra (*image restoration*), perbaikan kualitas citra (*image enhancement*), dan rekonstruksi citra (*image reconstruction*).

Citra atau gambar sendiri dapat diartikan sebagai suatu keluaran dari sistem perekaman data optik dan dapat diartikan sebagai fungsi intensitas cahaya dua dimensi yang dinyatakan oleh fungsi *f(x, y)*, dimana *f* dapat menyatakan intensitas, sedangkan *x* dan *y* merupakan posisi dari suatu piksel di dalam gambar.

2.3.1.*Color Spaces*

Color spaces atau ruang warna merupakan warna yang memungkinkan terbuat dari suatu grup pewarna[23]. Ruang warna sendiri sebenarnya merupakan kombinasi dari model warna dan fungsi pemetaan yang merepresentasikan nilai piksel.

Beberapa contoh ruang warna yang sering digunakan adalah CMYK, RGB, HSV, dan *Grayscale*. Setiap dari ruang warna memiliki fungsi masing – masing di dalam pengolahan citra.

- CMYK: merupakan ruang warna yang terdiri dari *Cyan, Magenta, Yellow,* dan *Key*. Ruang warna ini biasanya digunakan di dalam sistem percetakan.
- RGB: merupakan ruang warna yang terdiri dari pewarna *Red* (merah), *Green* (hijau), dan *Blue* (biru). Warna – warna yang dikombinasikan memiliki panjang gelombang sekitar 380 hingga 780 nm. Setiap kanal warna memiliki rentang nilai dari 0 hingga 255.
- HSV: merupakan model warna yang memberikan karakterisasi dari setiap warna berdasarkan corak (*Hue)*, intensitas (*Saturation*), dan nilai (*Value*). *Hue* akan memberikan pengaruh berupa nilai warna yang memiliki rentang nilai dari 0 hingga 360[28]. *Saturation* akan memberikan nilai intensitas dari suatu warna. Sedangkan *value* merupakan tingkat kecerahan dari warna itu sendiri. Baik *saturation*

maupun *value* memiliki rentang nilai dari 0 sampai 100. Transformasi nilai RGB ke HSV adalah sebagai berikut:

$$H = \begin{cases} 0, & C_{max} = 0 \\ 60 \times \dfrac{G - B}{C_{max} - C_{min}}, & C_{max} = R \\ 60 \times \dfrac{B - R}{C_{max} - C_{min}} + 120, & C_{max} = G \\ 60 \times \dfrac{R - G}{C_{max} - C_{min}} + 240, & C_{max} = B \end{cases} \quad (2.1)$$

Persamaan (2.1) adalah rumus untuk mencari nilai *Hue* dengan G merupakan nilai hijau, R adalah nilai merah, dan B merupakan nilai biru. C_{max} merupakan nilai maksimal dari R, G, dan B serta C_{min} merupakan nilai minimal dari R, G, dan B.

$$S = \begin{cases} \dfrac{C_{max} - C_{min}}{C_{max}}, & C_{max} \neq 0 \\ 0, & C_{max} = 0 \end{cases} \quad (2.2)$$

Persamaan (2.2) adalah rumus untuk mendapatkan nilai *Saturation*.

$$V = C_{min} \quad (2.3)$$

Persamaan (2.3) adalah rumus untuk mendapatkan nilai *Value*.

2.3.2.Segmentasi Citra

Berdasarkan Budi, dkk[24], segmentasi citra merupakan suatu proses pemisahan bagian atau segmen tertentu dalam citra yang akurat. Dalam konteks citra digital. Daerah hasil segmentasi merupakan kelompok piksel yang bertetangga atau saling berhubungan. Beberapa pendekatan segmentasi citra antara lain adalah:

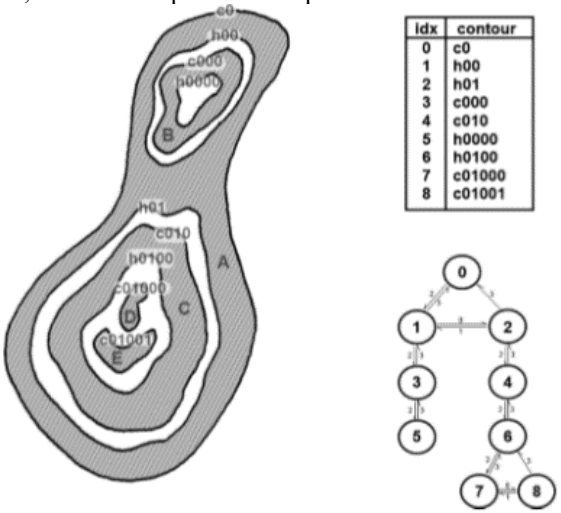
- Pendekatan batas (*boundary approach*) yang merupakan pendekatan yang dilakukan untuk mendapatkan batas antar daerah.
- Pendekatan tepi (*edge approach*) yang berfungsi untuk mengidentifikasi dan menghubungkan piksel tepi menjadi suatu batas.
- Pendekatan daerah (*region approach*) yang bertujuan untuk membagi citra berdasarkan daerah sesuai kriteria yang diinginkan.

Proses segmentasi yang digunakan sangat bervariasi dan memiliki tujuan untuk mendapatkan representasi sederhana dari suatu citra dan memperoleh suatu fitur dari citra tersebut. Penerapan metode segmentasi citra sendiri memiliki fungsi untuk menghasilkan objek berupa kumpulan

piksel yang memiliki nilai tertentu yang disebut dengan blob.

2.3.3.Kontur

Kontur merupakan suatu kurva gabungan dari titik yang kontinyu yang memiliki intensitas dan warna yang sama. Kontur sangat berguna untuk melakukan analisis bentuk dan deteksi objek. Di dalam opencv, kontur dapat dicari dengan menggunakan metode findContour(). Pada gambar 2.5[22], pada bagian kanan merepresentasikan hasil findContour(). Diketahui terdapat lima daerah berwarna tetapi kontur terbentuk dari tepi interior dan eksterior dari tiap daerah tersebut. Jadi ditemukan sembilan kontur yang terdaftar di tabel bagian kanan. Sedangkan, grafik bagian kanan bawah merepresentasikan *contour tree* yang terbentuk, dimana setiap *node* merupakan kontur.



Gambar 2.5. Konsep *Contour Tree*[1]

Dengan metode kontur di atas, dapat ditemukan luasan dari kontur dan melakukan pencarian terhadap titik tengah koordinat x dan y dari kontur yang terdeteksi tersebut.

[1] Konsep *Countour Tree* berdasarkan Kaehler dan Bradski[23]

2.3.4.OpenCV

Gambar 2.6. Logo OpenCV

Opencv merupakan *library* yang bersifat *open source* yang dapat dipakai untuk komputer visi. *Open source* berarti dapat digunakan dan dikembangkan oleh siapa saja baik dari bidang akademik maupun bukan dari bidang akademik. Gambar 2.6 merupakan logo resmi dari opencv. *Library* ini dapat digunakan dalam banyak bahasa pemrograman seperti Python, C++, Java dan bahasa pemrograman yang lainnya. Dengan adanya opencv dapat memudahkan pengambilan dan pengolahan citra gambar. Di dalam opencv terdapat beberapa modul yang dapat digunakan dan tersedia untuk pengolahan citra. Modul tersebut antara lain adalah:

- *Core*, untuk mendefinisikan struktur data dasar dan digunakan dalam seluruh modul
- *Imgproc*, untuk melakukan pemrosesan citra
- *Video*, untuk melakukan analisis video.

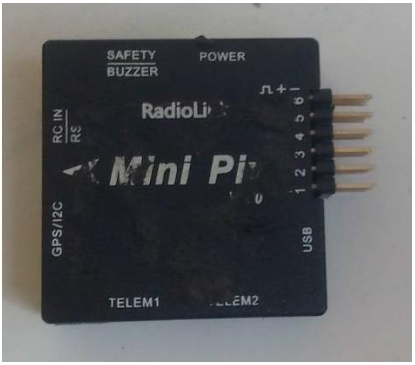
2.4. Komunikasi Serial

Komunikasi serial digunakan untuk menghubungkan dua prosesor dalam satu sistem dan memungkinkan pertukaran data antar dua prosesor tersebut. Di dalam komunikasi serial, pengiriman data dilakukan per bit secara bergantian dan berurutan. Keuntungan dari komunikasi serial berdasarkan Osisiogu[25] antara lain adalah:

- Membutuhkan kabel yang lebih sedikit sehingga dapat menghemat tempat
- *Cross talk* dapat dihindari
- Lebih murah untuk diimplementasikan.

2.5. Komponen Elektronik

2.5.1. RadioLink Mini Pix



Gambar 2.7. RadioLink Mini Pix

Flight controller merupakan kumpulan dari sensor yang memiliki tujuan untuk menjaga *drone* agar tetap seimbang sehingga *drone* dapat menjalankan misi dengan baik. Dalam tugas akhir ini, *flight controller* yang digunakan adalah RadioLink MiniPix yang memiliki fungsi yang hampir sama dengan *flight controller* Pixhawk. Keunggulan lain dari *flight controller* ini antara lain adalah dapat mempertahankan ketinggian dengan baik, *software* untuk menahan getaran dan dilengkapi dengan prosesor, barometer, akselerometer, dan kompas. Gambar 2.7 menggambarkan pinout yang terdapat pada RadioLink Mini Pix.

Berikut ini adalah spesifikasi dan fitur dari RadioLink MiniPix:

- Prosesor STM32F405VGT6
- Sensor Gyro MPU6500
- Kompas QMC5883L
- Barometer LPS22HB
- RC input berupa PPM/SBUS
- PWM input dengan jumlah 6 OneShot/ PWM output

Gambar 2.8. Pinout dari RadioLink Mini Pix[2]

2.5.2.Raspberry Pi 3 B+

Raspberry pi adalah salah satu jenis *single board computer* yang berukuran kecil dan dikembangkan oleh yayasan Raspberry Pi di Wales.

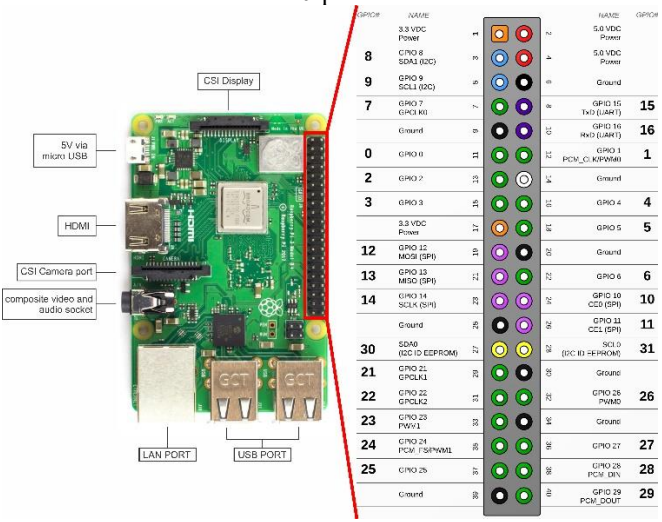
Raspberry Pi 3B+ sendiri memiliki prosesor Broadcom BCM28370B0 dengan kecepatan *clock* sampai dengan 1.4GHz dan memiliki RAM sebesar 1GB SRAM. Selain itu, tipe Raspberry pi ini dilengkapi dengan wifi dengan frekuensi 2.4GHz dan 5GHz. Rapberry pi 3B+ ini juga dilengkapi dengan empat port USB untuk dihubungkan ke perangkat tambahan seperti mouse, keybard, dll. Gambar 2.10 merupakan *pinout* dari Raspberry Pi 3B+ yang dibuat berdasarkan dari Raspberry Pi Compute 3+ *manual datasheet.*

[2] Berdasarkan *user manual* RadioLink Mini Pix

Gambar 2.9 Raspberry Pi 3B+

Berikut adalah spesifikasi dari raspberry pi 3 B+:

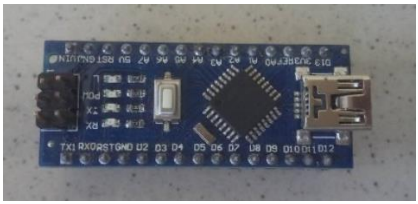
- Processor : Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz
- Memory : 1GB LPDDR2 SDRAM
- *Input* power : 5V/2.5A DC via micro USB connector, 5V DC via GPIO header, Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
- Access : Extended 40-pin GPIO header



Gambar 2. 10 PinOut Raspberry Pi 3B+[3]

[3] Berdasarkan *user manual* dari Raspberry Pi

2.5.3.Arduino Nano

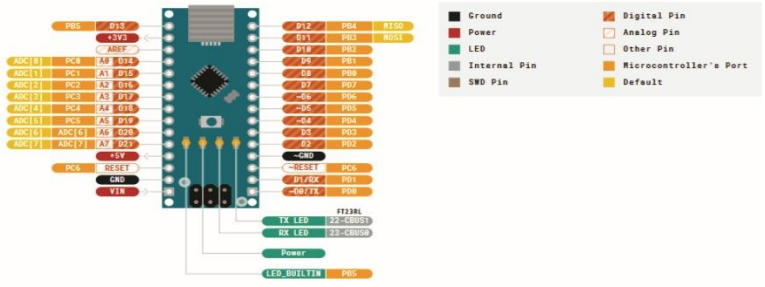


Gambar 2.11 Arduino Nano

Arduino nano merupakan salah satu jenis mikrokontroler dengan ATMega328 dan dioperasikan pada tegangan dari 0 hingga 5 V. perangkat ini sendiri telah dilengkapi dengan kecepatan clock hingga 16MHz dengan 14 port digital I/O (dengan 6 pin PWM) dan 8 port analog I/O.

Spesifikasi dari Arduino nano antara lain adalah:

- *Microcontroller* : Atmel ATmega328
- Tegangan operasi (level logika) : 5V
- Tegangan *input* (rekomendasi) : 7-12V
- Tegangan *input* (limit) :6-20V
- Pin digital I/O :14 (dengan 6 pin ouput PWM)
- Pin analog input : 8
- Flash Memory :32 KB
- SRAM :2 KB
- EEPROM :1 KB
- Kecepatan clock :16MHz



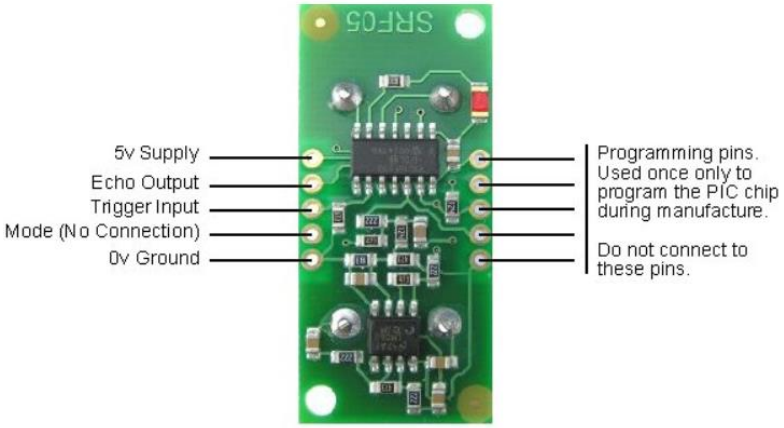
Gambar 2. 12 *Pinout* Arduino Nano[4]

[4] Berdasarkan *datasheet* dari Arduino Nano

2.5.4.SRF05 *Ultrasonic Sensor*

Gambar 2. 13 Sensor *ultrasonic* SRF 05

SRF05 merupakan salah satu jenis sensor *ultrasonic* yang merupakan pengembangan dari SRF04 dan didesain agar memiliki fleksibilitas dan jangkauan yang lebih besar. Jangkauan pada SRF05 ini hingga 4 meter. Sensor ini bekerja dengan car acara memberikan sinyal ke pin *trigger* dan selanjutnya akan ditangkap kembali oleh pin *echo* untuk dihitung lebih lanjut. Sensor ini bekerja dengan diberikan suplai daya 5V.



Gambar 2. 14 *Pinout* dari SRF05[5]

[5] Berdasarkan *datasheet* dari SRF05

2.5.5.Raspberry Pi Camera Rev 1.3

Gambar 2. 15 Raspberry Pi Camera Rev 1.3

Raspberry Pi Camera Rev 1.3 merupakan modul kamera yang dibuat secara resmi oleh Raspberry Pi Foundation. Kamera ini spesifikasi sebagai berikut:

- Resolusi : 5 Megapixel
- Mode video : 1080p30, 720p60, dan 640 x 480p60/90
- Sensor : OmniVision OV5647
- Resolusi sensor : 2592 x 1944 pixels
- Area gambar : 3.76 x 2.74 mm
- Ukuran pixel : 1.4 µm x 1.4 µm
- Ukuran optik : ¼ inch

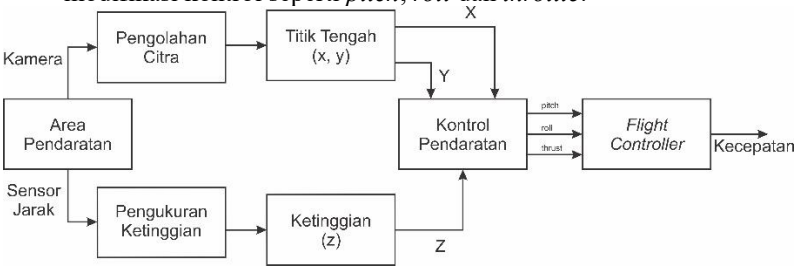
Halaman ini sengaja dikosongkan

BAB 3
PERANCANGAN SISTEM

Pada bab ini akan dibahas mengenai perancangan sistem pendaratan presisi *drone* pada objek darat bergerak. *Drone* akan melakukan pendaratan presisi pada landasan bergerak yang telah ditentukan berupa suatu papan yang memiliki ukuran 50x50cm. *Drone* akan melakukan navigasi secara manual menuju area pendaratan dan akan melakukan pendaratan otomatis pada objek darat bergerak. Setelah mencapai area pendaratan, *drone* akan melakukan pencarian titik tengah dan akan melakukan pendaratan di area pendaratan tersebut. Pendaratan akan dikatakan berhasil bila *drone* mampu mendarat pada landasan tersebut ketika landasan tersebut digerakkan. Gambar 3.1 menjelaskan sistem yang dibuat sesuai dengan deskripsi di atas.

Dalam perancangan tugas akhir ini, pemilihan *drone* yang digunakan memiliki pertimbangan sebagai berikut:

5. *Drone* mampu mempertahankan ketinggian degan baik meskipun tanpa GPS
6. Daya angkat *drone* mampu mengangkat seluruh komponen yang akan dipasang pada *drone*
7. *Drone* mampu melakukan penerbangan otomatis dan melakukan modifikasi kontrol seperti *pitch*, *roll* dan *throttle*.



Gambar 3.1. Diagram Blok Sistem secara Keseluruhan

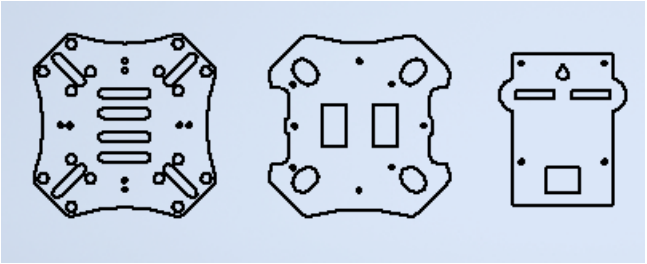
Setelah melakukan studi untuk mendapatkan *drone* sesuai dengan kriteria di atas, perancangan dibagi menjadi dua yaitu perancangan perangkat keras dan perancangan perangkat lunak. Perancangan perangkat keras pada *drone* meliputi pemasangan komponen mekanik pada *drone* dan pemasangan seluruh komponen elektronik pada *drone*. Tujuan utama dalam perancangan perangkat keras pada *drone* adalah untuk mengatur agar *drone* mampu melakukan penerbangan secara stabil karena seluruh beban *drone* terpusat pada titik berat dari *drone* atau bisa

disebut *center of gravity*. Sedangkan perancangan perangkat lunak dari *drone* meliputi pengolahan citra, pencarian titik tengah dari *landing pad*, dan metode kontrol dari *drone*. Pengolahan citra memiliki tujuan untuk melakukan kalibrasi agar *drone* mampu mendeteksi objek pendaratan dengan baik. Pencarian titik tengah dari *landing pad* bertujuan untuk mencari titik tengah dari area pendaratan agar *drone* mampu memposisikan diri tepat di tengah area pendaratan. Sedangkan, metode kontrol *drone* merupakan metode untuk melakukan pengiriman hasil pencarian titik tengah menuju ke *flight controller* sehingga *flight controller* mampu melakukan perintah sesuai yang diinginkan.

3.1. Rancangan Mekanik

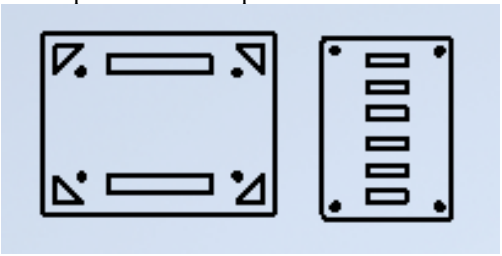
Gambar 3.2. Realisasi *Drone* Tampak Atas

Pada tugas akhir ini, *drone* akan menggunakan *quadcopter* dengan tipe *frame* X. *Frame* yang akan digunakan adalah DJI Flamewheel F450 dengan diameter 45 cm dari motor ke motor. *Drone* ini menggunakan propeller dengan ukuran 9443 untuk mendapatkan gaya angkat yang maksimal. Pada gambar 3.2 dapat dilihat bahwa keseluruhan berat *drone* berpusat di tengah *drone*. Hal ini memiliki fungsi untuk menyeimbangkan *drone* sehingga *drone* mampu terbang dengan kondisi optimal dan *drone* mampu terbang dengan stabil.



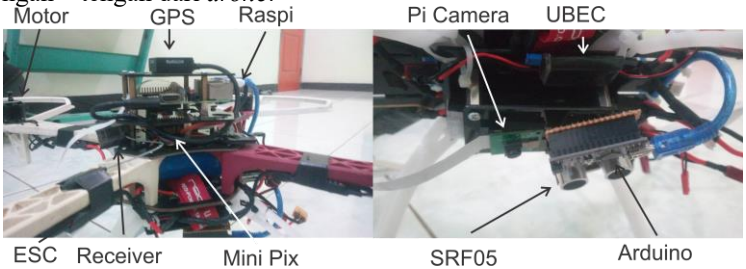
Gambar 3.3. Desain Plat Tengah

Pada gambar 3.3 merupakan desain dari plat tengah *drone*. Pada bagian ini terpasang komponen utama dari *drone* yaitu RadioLink Mini Pix, Raspberry Pi, Receiver R9DS dan GPS. Selain itu, pada plat utama kerangka *drone* terpasang ESC dan motor di setiap lengannya dan baterai 3300 mAh diantara plat bawah dan plat atas dari *drone*.



Gambar 3.4. Desain Plat Baterai Bawah

Pada gambar 3.4 merupakan desain dari plat baterai bawah yang terpasang pada *drone*. Bagian ini digunakan untuk meletakkan baterai untuk memberikan suplai daya ke Raspberry Pi, untuk menempatkan UBEC 5V 2A, untuk menempatkan Arduino Nano dan sensor jarak SR-05 di bagian bawah dari *drone* dan untuk meletakkan Pi Camera tepat di tengah – tengah dari *drone*.

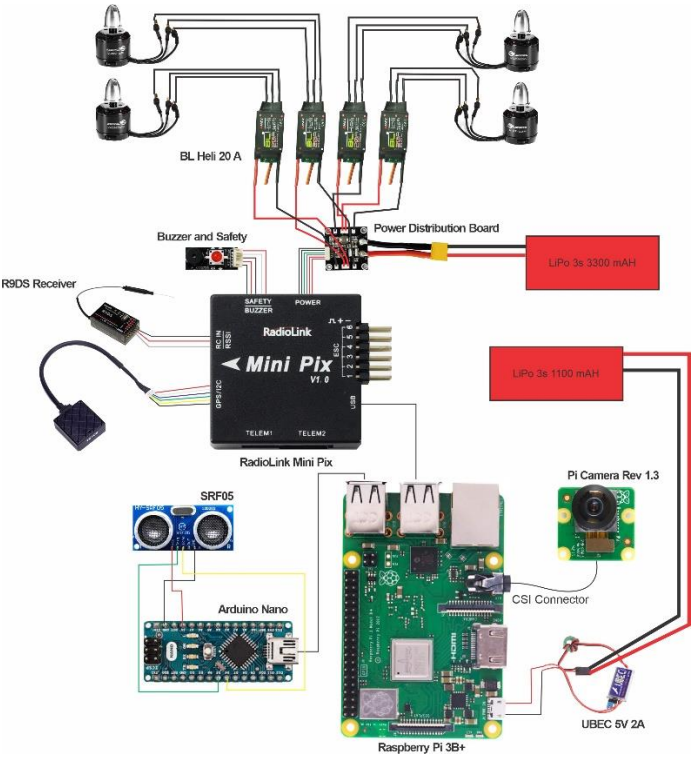


Gambar 3.5. Realisasi Penempatan Elektronik *Drone*

Pada gambar 3.5. merupakan realisasi dari penempatan seluruh komponen elektronik pada *drone*. Dapat dilihat pada *platform* tengah, komponen elektronik sebisa mungkin untuk ditempatkan di tengah. Hal ini dilakukan agar pusat massa dari *drone* sebisa mungkin di tengah agar tidak membebani *drone* ketika melakukan penyeimbangan gerakan. Selain itu, pada *platform* bawah dapat dilihat bahwa Pi Camera diletakkan pada posisi tengah dari *platform* bawah dari *drone*. Hal ini dilakukan agar mempermudah *drone* untuk melakukan pergerakan penengahan atau *pergerakan menengah* untuk menjalankan misi dengan baik.

3.2. Rancangan Elektronik

Pada bagian perancangan elektronik *drone* menjelaskan mengenai sistem perancangan elektronik pada *drone* berupa suplai tegangan dan distribusi ke setiap komponen dan *wiring* sebagai koneksi setiap komponen yang digunakan. Sumber tegangan pada *drone* ini terdapat dua yaitu dari baterai *litium polymer* (LiPo) 3 sel dengan kapasitas 3300 mAH dan LiPo dengan kapasitas 1100 mAH. Suplai dari baterai LiPo 3300 mAH digunakan untuk menyuplai empat motor *brushless* sedangkan suplai dari LiPo 1100 mAH digunakan untuk menyuplai Raspberry Pi 3B+. Hal ini dilakukan agar *drone* memiliki waktu yang lebih lama untuk melakukan penerbangan. Tegangan dari baterai LiPo 3300 mAH didistribusikan ke empat ESC Emax 20A melalui PDB (*power distribution board*) sebelum melalui brushless motor DC LDPower MT2213 920kV dengan tegangan operasi 2-4s. Suplai dari Lipo 1100 mAH digunakan untuk memberikan daya ke Raspberry Pi 3B+ dengan cara diinput dahulu ke UBEC 5V 2A agar sesuai dengan spesifikasi yaitu 5V 2,5A. Komponen – komponen lain seperti Arduino Nano, sensor jarak SR-05, Pi Camera, RadioLink MiniPix mendapatkan suplai daya dari port USB Raspberry Pi 3B+. GPS pada rangkaian listrik dari wahana hanya digunakan sebagai keamanan dengan cara mengubah mode menjadi *Loiter*. Wahana dikontrol melalui *Ground Control Station* pada laptop yang terhubung melalui *router* menggunakan aplikasi VNC Viewer. Selain itu, laptop juga terinstal program *Mission Planner* yang digunakan untuk melakukan kalibrasi dari drone mulai dari kalibrasi level, kalibrasi kompas, kalibrasi radio, dan kalibrasi ESC. Drone masih dapat dikontrol secara manual dengan menggunakan *remote control* yang terhubung antara *receiver* pada *drone.*



Gambar 3.6. Diagram Komponen Elektronik *Drone*

Dari gambar 3.6 menggambarkan diagram kelistrikan seluruh komponen yang ada pada *drone* ini. Berikut ini merupakan penjelasan mengenai seluruh komponen yang ada pada *drone* ini.

1. LDPower MT2213 920kV *Brushless DC Motor*

   Pada tugas akhir ini dipilih LDPower MT2213 920kV Brushless DC Motor dengan jumlah dua buah CW (*clockwise*) dan dua buah CCW (*counterclockwise*) serta setiap motor dipasangkan dengan *propeller* 9443. Motor ini dipilih karena memiliki *thrust* yang cukup tinggi yaitu 640g per motor sehingga bila dilakukan kalkulasi, *drone* ini mampu mengangkat beban sebesar 2560 g.

2. Emax BLHeli 20A *Electronic Speed Controller*

   Pada tugas akhir ini dipilih EMAX BLHeli 20A ESC karena mampu menyuplai tegangan ke motor dengan baik dan dapat diprogram sesuai dengan keinginan. Selain itu, komponen ini dapat

digunakan pada baterai 2 hingga 4 sel baterai.

3. RadioLink Mini Pix

   Flight controller yang digunakan pada tugas akhir ini adalah RadioLink Mini Pix yang memiliki fungsi untuk mengatur kestabilan terbang pada setiap motor dengan menggunakan kalkulasi dari sensor – sensor yang ada di dalam Mini Pix sendiri. Selain itu, Mini Pix sendiri dapat dikomunikasikan secara serial ke Raspberry Pi selain melalui port – port yang terdapat pada Mini Pix sendiri.

4. Radiolink R9DS dan RadioLink AT9S *Radio Control*

   Pengendalian *drone* manual menggunakan komunikasi antara *receiver* dan *transmitter* yang berupa *remote control*. Receiver ini menggunakan protokol SBUS dengan rentang 172 – 1792 dan pada *firmware* ArduPilot dipetakan menjadi 1000 hingga 2000. Pengendalian *drone* manual diperlukan untuk mengambil alih *drone* ketika terjadi error pada sistem.

5. *Power Distribution Board*

   Power Distribution Board (PDB) merupakan suatu komponen yang memiliki fungsi untuk membagi tegangan ke setiap ESC dan motor. Pada tugas akhir ini PDB memiliki fungsi untuk membagi tegangan dari LiPo untuk empat ESC dan motor.

6. LiPo 3s 3300 mAH dan LiPo 1100 mAH

   LiPo 3s 3300 mAH merupakan sumber tegangan yang memiliki fungsi untuk memberikan tegangan kepada empat ESC. Sedangkan LiPo 1100 mAH digunakan untuk memberikan suplai tegangan ke Raspberry Pi 3B+. Pemisahan sumber tegangan dilakukan agar *drone* memiliki kemampuan terbang lebih lama.

7. UBEC 5V 2A

   UBEC 5V 2A memiliki fungsi untuk menurunkan tegangan dari baterai LiPo 1100 mAH agar mampu memberikan suplai ke Raspberry Pi 3B+ dan komponen – komponen yang berkaitan dengan Raspberry Pi 3B+.

8. Raspberry Pi 3B+

   Raspberry Pi 3B+ pada tugas akhir ini tersambung secara serial ke Pixhawk dan Arduino Nano serta tersambung melalui konektor CSI ke PiCamera. Tugas utama dari Raspberry sendiri adalah sebagai pengolahan citra digital dan data ketinggian untuk mendapatkan data koordinat. Selanjutnya, data koordinat tersebut akan diolah menjadi data kontrol yang kemudian akan dikirimkan

menuju Pixhawk.

9. Pi Camera

   Pi Camera merupakan komponen yang memiliki fungsi untuk merekam gambar untuk selanjutnya diolah di Raspberry Pi. Pi Camera sendiri tersambung ke Raspberry Pi melalui konektor CSI.

10. RadioLink TS100 Mini M8N GPS

    GPS di dalam sistem ini hanya memiliki fungsi untuk melakukan navigasi secara manual menggunakan mode Loiter. Hal ini dilakukan ketika awal misi maupun ketika terjadi ingin melakukan *Remote Control override* atau mengambil alih kontrol menggunakan *remote control*.

11. Arduino Nano

    Dalam sistem ini, Arduino Nano memiliki fungsi untuk mengolah data ketinggian dari sensor SRF05. Setelah diperoleh data ketinggian tersebut, kemudian data tersebut akan dikirimkan ke Raspberry Pi secara serial melalui port.

12. SRF05

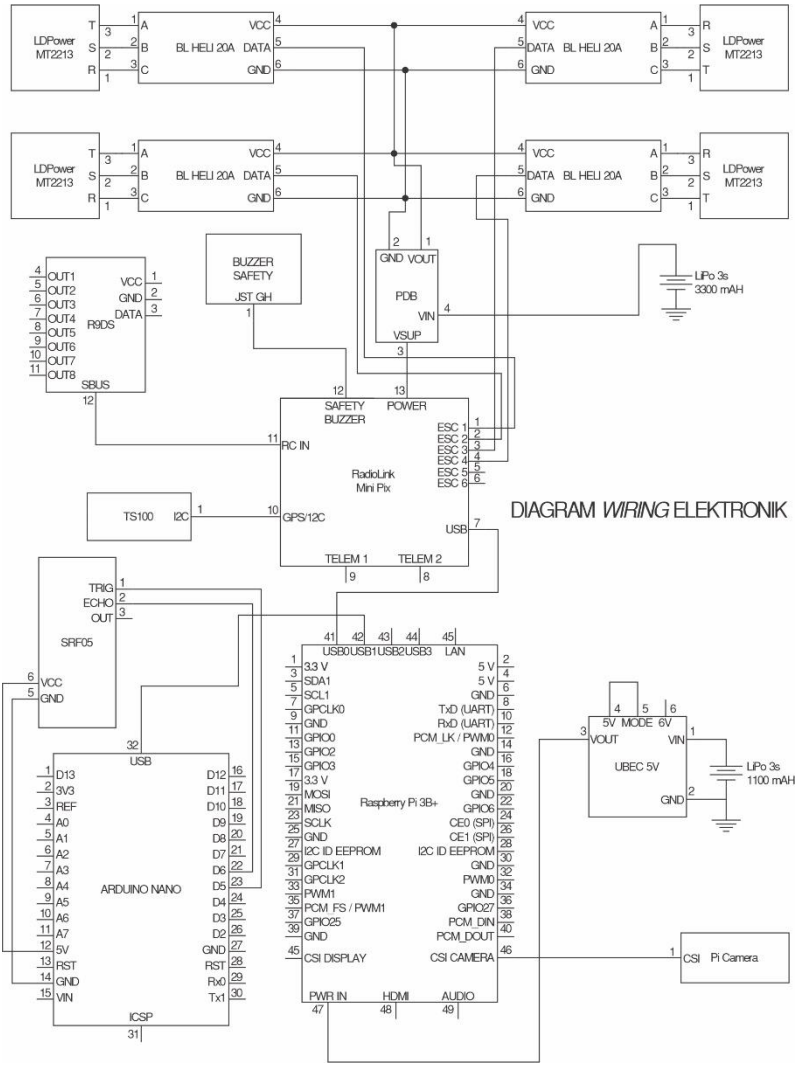
    SRF05 merupakan sensor ultrasonic yang di dalam tugas akhir ini dimanfaatkan untuk mengukur data ketinggian yang diproses dengan Arduino Nano.

13. Router WiFi

    Router WiFi di dalam tugas akhir ini memiliki fungsi untuk menghubungkan antara Raspberry Pi dengan laptop. Router WiFi dapat berupa router biasa maupun *smartphone* yang diubah fungsinya menjadi router. Dengan adanya router, kondisi *drone*n dapat terus dipantau.

14. Laptop

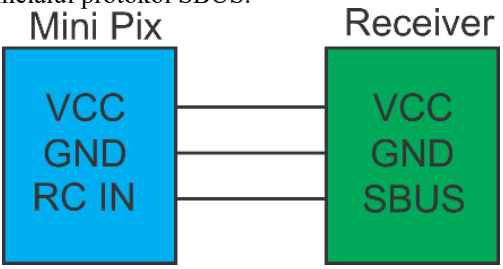
    Laptop memiliki fungsi sebagai *Ground Control Station* dan terhubung ke router dengan Raspberry Pi sehingga mampu melakukan *remote access*. Dengan adanya *Ground Control Station*, maka dapat memberikan *display* dari Raspberry Pi mengenai kondisi *drone* sekarang. Selain itu, parameter kalibrasi juga dapat diubah melalui laptop yang tersambung ke router.



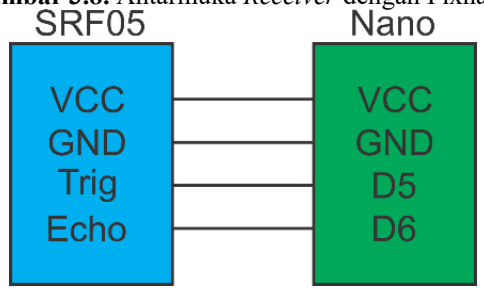
Gambar 3.7. Diagram *Wiring* Wahana

Dari diagram *wiring* diatas, dapat dilihat bahwa Pixhawk dan Arduino Nano dihubungkan secara serial melalui port USB ke Raspberry Pi. Agar dapat dikontrol secara manual baik untuk mencari area pendaratan atau ketika *override* untuk menghindari terjadinya *crash*,

receiver dihubungkan ke Pixhawk melalui RC *Input* sesuai dengan gambar 3.8. melalui protokol SBUS.



Gambar 3.8. Antarmuka *Receiver* dengan Pixhawk



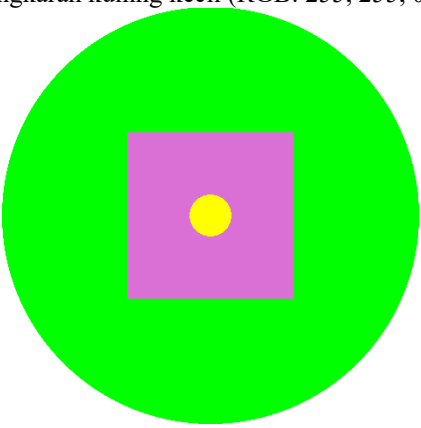
Gambar 3.9. Antarmuka SRF05 dengan Arduino Nano

Untuk mendapatkan data ketinggian, SRF05 dihubungkan dengan Arduino Nano sehingga data dapat diolah dan dikirimkan ke Raspberry Pi 3B+. SRF05 sendiri mendapatkan suplai dari Arduino Nano dengan tegangan 5V sesuai dengan gambar 3.7.

Secara umum sistem dimulai dari Raspberry Pi yang mendapatkan data citra untuk mendeteksi area pendaratan untuk selanjutnya diolah sehingga mendapatkan koordinat x dan koordinat y yang merupakan titik tengah area pendaratan. Untuk melengkapi data tersebut, Raspberry Pi mengirimkan *request* kepada Arduino Nano untuk data ketinggian. Arduino Nano lalu mengolah data dari sensor ultrasonic SRF05 dan mengirimkan balik data ke Raspberry Pi. Raspberry Pi kemudian mengolah data tersebut menjadi koordinat z. Dari koordinat – koordinat tersebut, kemudian Raspberry Pi mengolah data – data tersebut menjadi data kecepatan dan mengirimkan perintah ke *flight controller* Pixhawk untuk diterjemahkan ke ESC menjadi kecepatan putar dari motor.

3.3. Perancangan Komputer Visi dan Kontrol

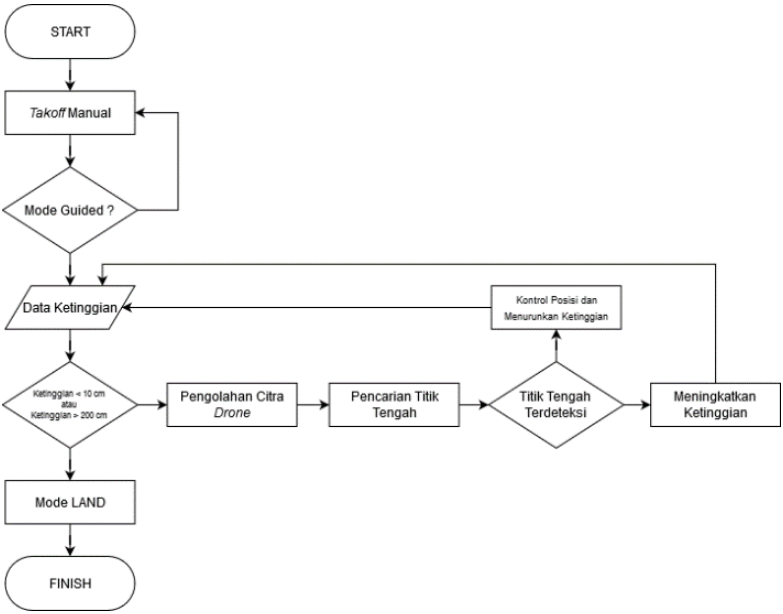
Pada bagian perancangan komputer visi dan kontrol akan dijelaskan mengenai tahapan – tahapan dari pendeteksian area pendaratan, pengolahan citra, dan kontrol gerakan pada *drone*. Tujuan akhirnya adalah pendeteksian area pendaratan dan melakukan kontrol berupa pergerakan menengah ke titik tengah area pendaratan dan melakukan pendaratan secara presisi. Pada gambar 3.10 merupakan desain dari area pendaratan dengan ukuran diameter 50 cm. Pada area pendaratan sendiri terdapat tiga bagian yaitu lingkaran hijau (RGB: 0, 255, 0), kotak tengah (RGB: 228, 112, 214), dan lingkaran kuning kecil (RGB: 255, 255, 0).



Gambar 3.10. Desain Area Pendaratan

Dari gambar 3.11, dapat dilihat mengenai *flowchart* atau diagram alir dari misi pendaratan yang akan digunakan dalam tugas akhir ini. Pertama, *drone* akan melakukan *thresholding* untuk menemukan lingkaran luar dari area pendaratan dan akan mencari konturnya. Apabila lingkaran luar ditemukan, maka akan mencari titik tengah dan melakukan kontrol untuk menengahkan dan menurunkan *throttle* dari *drone*. Sebaliknya, apabila *drone* tidak menemukan kontur maka akan menambah ketinggian hingga menemukan kontur tersebut. Selanjutnya, *drone* akan mencari kotak dengan cara yang sama seperti sebelumnya. Ketika *drone* tidak berhasil menermukan bentuk kotak maka *drone* akan menaikkan ketinggian dan melakukan pengecekan lingkaran luar lalu kotak. Setelah *drone* berhasil menemukan kotak, maka *drone* akan berusaha mencari lingkaran di dalam. Apabila tidak berhasil maka, *drone* akan menambah ketinggian dan mendeteksi dari awal. Setelah *drone*

memiliki ketinggian kurang dari 10 cm, maka *drone* akan mengganti mode menjadi mode LAND dimana *drone* akan menurunkan *thrust* hingga berhasil melakukan pendaratan dan *drone* akan melakukan *disarm* secara otomatis dan misi telah berhasil dilakukan.



Gambar 3.11. *Flowchart* Misi Pendaratan

3.3.1.Pengolahan Citra

Bagian pengolahan citra ini berfungsi untuk mendeteksi area pendaratan. Pada bagian sebelumnya telah dijelaskan mengenai spesifikasi warna dari area pendaratan. Hal ini dilakukan agar mempermudah pendeteksian warna. Selain itu, setiap jarak yang telah dideteksi oleh sensor ketinggian akan mendeteksi tiap bagian yang berbeda. Sesuai dengan gambar 3.12, tangkapan citra pada tiap ketinggian memiliki efek cahaya yang berbeda sehingga memungkinkan hal yang seharusnya tidak terdeteksi menjadi terdeteksi ataupun sebaliknya.

Gambar 3.12. Tangkapan Citra pada Tiap Ketinggian

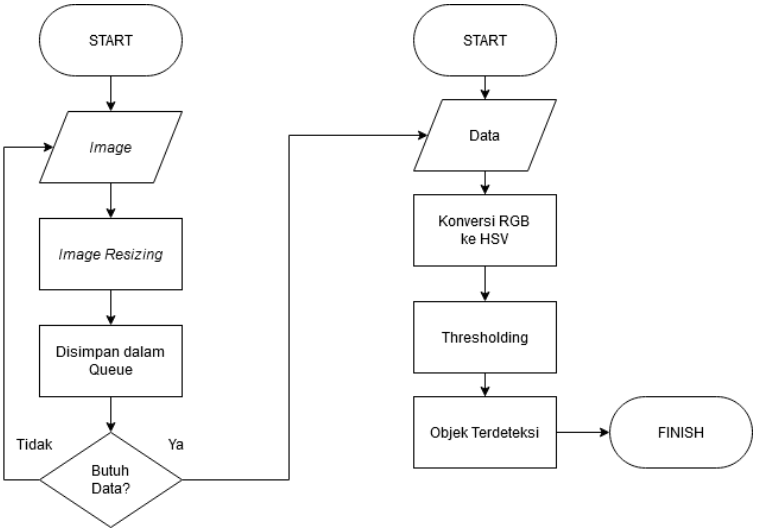
Di dalam pengolahan citra, terdapat dua bagian penting yaitu bagian perekaman data dan bagian pengolahan citra. Bagian perekaman data dipisahkan dengan bagian pengolahan citra dengan membuat *thread* sendiri dengan fungsi agar data yang diolah sebisa mungkin *real-time*. Hal ini dilakukan dengan cara data yang direkam dimasukkan dalam sebuah nilai tunggu *queue*. Sebelum data dimasukkan ke dalam nilai tunggu, data diubah ukurannya menjadi 320 piksel x 240 piksel sehingga didapatkan resolusi video sebesar 240p dan data dirotasi terlebih dahulu sebesar 180 derajat untuk menyesuaikan dengan posisi kamera yang terpasang pada *drone*. Isi di dalam nilai ini selalu diperbaharui setiap saat dengan cara mengosongkan dan mengisi baru secara terus menerus hingga *thread* utama membutuhkan data citra untuk kemudian diolah. *Pseudo-code* pada bagian ini adalah sebagai berikut:

```
camNumber = camNumber
cap = VideoCapture(camNumber)
cap.set(CAP_PROP_FRAME_WIDTH, DEFAULT_WIDTH)
cap.set(CAP_PROP_FRAME_HEIGHT, DEFAULT_HEIGHT)
qCamera = Queue()
tCamera = Thread(target=._reader)
tCamera.daemon = True
tCamera.start()

def _reader():
    while True:
        ret, frame = cap.read()
        frame = cv.rotate(frame, cv.ROTATE_180)
        if not ret:
            break
        if not qCamera.empty():
            try:
                qCamera.get_nowait()
            except:
                pass
        qCamera.put((ret, frame))

def read():
    return qCamera.get()
```

Gambar 3.13. Perbandingan Gambar tiap Proses

Data yang telah direkam tersebut, kemudian diolah lebih lanjut hingga mendapatkan objek yang diinginkan dan dalam kasus ini adalah objek area pendaratan. Data tersebut mula – mula akan di *blur* terlebih dahulu dengan menggunakan *GaussianBlur* agar dapat mempermudah pendeteksian dari area pendaratan sesuai dengan gambar 3.13. Selanjutnya, data citra akan dikonversi dari RGB menjadi HSV untuk memudahkan kalibrasi pencarian warna dari objek yang diinginkan. Setiap kali ingin memulai penerbangan nilai HSV tiap – tiap objek akan selalu dikalibrasi. Selain itu, nilai hasil kalibrasi dari tiap – tiap mode penerbangan akan disimpan di dalam *file* yang setiap akan memulai penerbangan akan selalu dipanggil kembali tiap nilainya. Nilai HSV sendiri terbagi menjadi enam jenis yaitu *lower hue, upper hue, lower saturation, upper saturation, lower value,* dan *upper value*. Data HSV kemudian dijadikan data citra biner dengan menggunakan *syntax* cv.inRange untuk memisahkan antara objek yang diinginkan dan kemudian dilakukan *erode* untuk mengurangi *noise* dari objek – objek lain yang mungkin terdeteksi bersamaan dengan objek yang diinginkan. *Eroding* sendiri dilakukan sebanyak sekali iterasi agar objek penting lain tidak terpotong. *Pseudo-code* pada bagian ini adalah sebagai berikut:

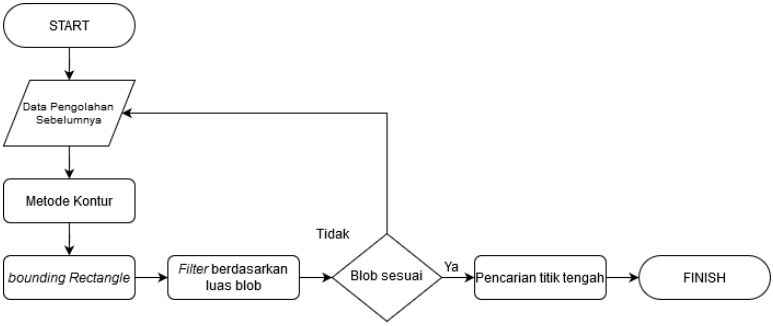
```
frame = cv.GaussianBlur(frame, BLUR_GAUSSIAN, sigmaX=0)
frameHSV = cv.cvtColor(frame, cv.COLOR_BGR2HSV)

lowerRange = np.array([lowerHue, lowerSat, lowerVal])
upperRange = np.array([upperHue, upperSat, upperVal])
maskHSV = cv.inRange(frameHSV, lowerRange, upperRange)
erodeMask = cv.erode(maskHSV, MORPH_KERNEL, iterations=1)

contours, _ = cv.findContours(erodeMask, cv.RETR_TREE,
cv.CHAIN_APPROX_SIMPLE)
```



Gambar 3.14. Diagram Blok Pemrosesan Citra

Gambar 3.14 merupakan diagram blok proses pengolahan citra yang dilakukan. Pada bagian *thresholding* sendiri, nilai dari HSV akan berubah – ubah sesuai dengan ketinggian dan objek yang ingin dicari. Dengan menggunakan metode ini, *drone* dapat mengenali area pendaratan tanpa dengan baik dan mengurangi pengaruh dari cahaya yang dapat membuat warna area pendaratan berubah.

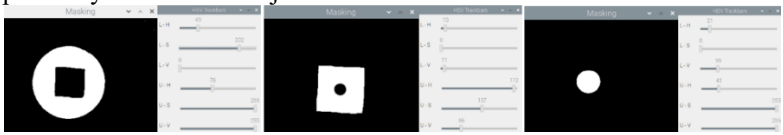
3.3.2.Pencarian Titik Tengah Landasan



Gambar 3.15. *Flowchart* Pencarian Titik Tengah

Gambar 3.15 merupakan *flowchart* dari pencarian titik tengah

landasan. Hasil pengolahan citra yang telah dijelaskan pada bagian sebelumnya akan mengenali area area pendaratan menggunakan metode pencarian kontur. Di setiap ketinggian memiliki kalibrasi *thresholding* HSV yang berbeda. Ini dilakukan untuk menghilangkan *noise* yang terjadi akibat perbedaan warna karena pantulan cahaya yang berbeda. Pada gambar 3.16 merupakan *thresholding* objek berdasarkan ketinggian. Kalibrasi rentang HSV pun harus selalu dilakukan sebelum memulai misi karena adanya kemungkinan perbedaan warna akibat perbedaan pencahayaan di area menjalankan misi.



Gambar 3.16. *Thresholding* berbagai Ketinggian

Setelah mendapatkan *thresholding* dengan gambar diatas berikutnya dilakukan pencarian kontur dengan menggunakan *cv2.findContour.* Dengan menggunakan metode ini, maka data objek yang diinginkan dapat dipisahkan dengan data *background* agar dapat diolah lebih lanjut. *Pseudo-code* dari bagian ini adalah sebagai berikut:

```
contours, _ = cv.findContours(dilationMask, cv.RETR_TREE,
cv.CHAIN_APPROX_SIMPLE)
```

Dari kontur yang telah ditemukan dengan metode *findContour*, dengan menggunakan *cv2.boundingRect*, area pendaratan dicari koordinat ujung kiri atas, panjang, dan lebarnya. Dengan menggunakan metode ini, maka akan dicari koordinat dari area tersebut dengan menggunakan rumus titik berat sebagai berikut:

$$x = \frac{A_{area} \times x_{area}}{A_{area}} \tag{3.1}$$

$$y = \frac{A_{area} \times y_{area}}{A_{area}} \tag{3.2}$$

Pseudo-code dari metode *boundingRect* adalah sebagai berikut:

```
x, y, width, height = cv.boundingRect(contour)
```

Selanjutnya, dari *width* atau panjang kontur yang ditemukan dan dari *height* akan dilakukan pemisahan dari luas area. Hal ini memperkecil kesalahan pendeteksian yang terjadi karena objek lain yang mungkin berada di dekat area pendaratan.

Dengan (x, y) merupakan titik tengah dari area yang dideteksi.

A_{area} merupakan luas dari area yang dideteksi dan x_{area} merupakan panjang area dari koordinat x serta y_{area} merupakan panjang area dari koordinat y. *Pseudo-code* dari bagian ini adalah sebagai berikut:

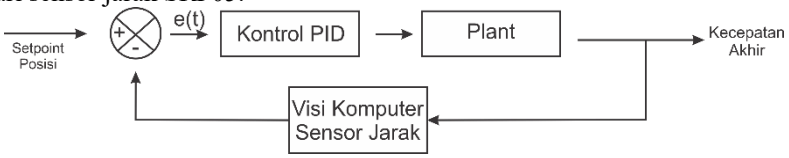
```
area = w * h
if w * h > areaMin:
    cogXFinal = contourX / area
    cogYFinal = contourY / area
    return cogXFinal, cogYFinal
```

Untuk mempermudah *display*, maka titik tengah dan area yang terdeteksi akan digambar dengan metode *rect* untuk menggambar area yang terdeteksi. *Pseudo-code* dari bagian ini adalah sebagai berikut:

```
cv.rectangle(frame, (x, y), (x + w, y + h), RED, thickness=3)
```

3.3.3.Metode Kontrol *Drone*

Metode kontrol *drone* ini dikombinasikan antara koordinat x dan y dari pendeteksian pada bagian sebelumnya dan koordinat z yang didapat dari sensor jarak SRF05.



Gambar 3.17. Diagram Blok Kontrol *Drone*

Gambar 3.17 merupakan diagram blok kontrol *drone* yang mampu menjelaskan keseluruhan sistem kontrol. Sebelumnya ditentukan setpoint yang berupa setengah dari panjang dan lebar dari *frame*. Dari setpoint tersebut, maka akan dibandingkan dengan posisi koordinat x maupun y yang terdeteksi saat ini. Selisih dari perbandingan tersebut kemudian dikalkulasikan menjadi kecepatan gerak *drone* dengan rumus:

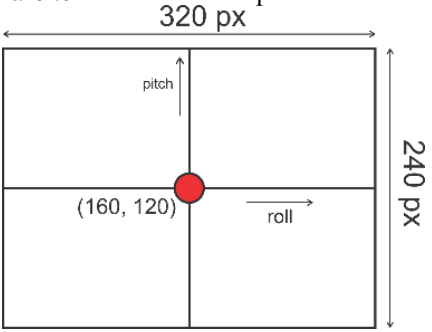
$$V_{now} = \frac{V_{max}}{Error} \tag{3.3}$$

dengan V_{now} yang merupakan kecepatan yang diinginkan, V_{max} merupakan kecepatan maksimal dari pergerakan *drone*, dan $Error$ merupakan selisih dari *setpoint* dengan koordinat yang terdeteksi. Dengan adanya rumus ini juga dapat menjadi penentu untuk melakukan *tuning* PID. Selanjutnya, penghitungan PID dengan menggunakan rumus:

$$V = Kp \times Err + Ki \times \left(Err + Err_{prev}\right) + Kd \times (Err - Err_{prev}) \tag{3.4}$$

dengan V merupakan kecepatan akhir, (Kp, Ki, Kd) merupakan

konstanta PID yang akan di *tune*, Err merupakan selisih dari *setpoint* dengan koordinat yang terdeteksi, dan E_{prev} merupakan error dari hasil kalkulasi sebelumnya. *Tuning* dilakukan secara manual untuk mendapatkan hasil yang baik sesuai dengan kondisi lapangan. Dengan adanya kontrol PID dapat membuat pergerakan penengahan dari *drone* variatif sesuai dengan posisi *drone* terhadap area pendaratan. Semakin dekat maka kecepatan akan semakin pelan dan semakin jauh dari area pendaratan maka *drone* akan semakin cepat.



Gambar 3.18. Referensi *Frame* Kamera pada Pergerakan *Drone*

Dari perhitungan – perhitungan di atas maka akan mendapatkan kecepatan untuk mendekati *setpoint* yang sesuai dengan gambar 3.19. Dari gambar 3.18, dapat dijelaskan bahwa panjang *frame* akan mempengaruhi gerak *roll* dari *drone*, dan lebar *frame* akan mempengaruhi gerak *pitch* dari *drone*. Untuk mengatur *throttle* dari *drone* sendiri nilai nya akan konstan sehingga tidak terpengaruh oleh error dari ketinggian yang terdeteksi. Selain itu, bila koordinat x positif akan membuat *drone* melakukan *roll* ke kanan dan sebaliknya, untuk y positif akan membuat *drone* melakukan *pitch* ke depan dan sebaliknya, sedangkan untuk nilai z positif akan menyebabkan *throttle* menurun dan sebaliknya. Apabila, *drone* tidak mendeteksi adanya area pendaratan maka, pengolahan citra akan memberikan nilai kurang dari 0. *Pseudo-code* pada bagian ini adalah sebagai berikut:

```
if xCenter < 0 and yCenter < 0:
    vx, vy, vz = 0, 0, -0.2

else:
    pidX.update(xCenter)
    pidY.update(yCenter)
    vx = pidY.output
    vy = pidX.output * (-1)
    vz = 0.2
```

Setelah kecepatan dari tiap koordinat dikalkulasi, maka Raspberry Pi akan mengirimkan MAVLink ke *flight controller* menggunakan *library* dronekit. Dengan menggunakan dronekit, *drone* akan di set terbang menggunakan mode GUIDED dan akan terus melakukan pengecekan apabila *drone* akan di *override* oleh RC apabila terjadi error. Apabila *drone* telah mendarat maka *drone* akan *disarm* secara otomatis.

Di dalam kontrol pada MAVLink, *drone* menggunakan *message* dengan tipe SET_POSITION_TARGET_LOCAL_NED yang akan melakukan pemosisian dari *drone* berdasarkan posisi, kecepatan dan arah muka dari *drone*. Berikutnya, *drone* akan diset orientasinya dengan menggunakan MAV_FRAME_BODY_OFFSET_NED dimana akan membuat komponen kecepatan dari *drone* akan relatif dengan arah kepala *drone* berdasarkan posisi sekarang. Selanjutnya dilakukan pemberian bitmask untuk mengontol kecepatan vx, vy, dan vz serta menghiraukan komponen – komponen lain seperti *yaw, yaw rate*, posisi x, posisi y, dan posisi z. *Pseudo-code* dari bagian ini adalah sebagai berikut:

```
msg =
vehicle.message_factory.set_position_target_local_ned_encode(
    0,
    0, 0,
    mavutil.mavlink.MAV_FRAME_BODY_OFFSET_NED,
    DEFAULT_MASK, # 0b0000111111000111
    0, 0, 0,
    vx, vy, vz
    0, 0, 0,
    0, 0)
vehicle.send_mavlink(msg)
```

3.4. Fitur Pendukung

Pada bagian fitur pendukung ini akan dijelaskan tambahan – tambahan yang dapat membantu kelangsungan misi. Fitur pendukung

antara lain adalah pengolahan data ketinggian, pengiriman data serial, dan penyimpanan dan pengeluaran nilai kalibrasi.

3.4.1.Pengolahan Data Ketinggian

Data ketinggian diolah di Arduino dengan menggunakan sensor SRF05. Proses pengambilan data ketinggian sendiri dilakukan dengan membuat suatu *thread* baru. Hal ini dilakukan agar tidak memberatkan proses yang terjadi pada Raspberry Pi sehingga nilai ketinggian sebisa mungkin *real-time*.

Thread ini hampir sama dengan *thread* untuk mengambil data gambar secara *real-time* dengan cara menyimpan data pada nilai tunggu *queue* khusus untuk data ketinggian. Pengiriman data ketinggian dari Arduino ke Raspberry dilakukan dengan cara *request* dari Raspberry Pi sehingga tidak terjadi *buffer* data dengan menerima data berupa *string* 'a' dan Arduino akan mengirim balik data jarak ke Raspberry Pi. Ketika pemrosesan data dari *thread* utama membutuhkan data ketinggian, maka *thread* utama dapat mengambil data tersebut di nilai tunggu *queue* data ketinggian. Selain itu, untuk mencegah terjadinya error pengiriman data diberikan *syntax try* dan *except* dengan mengirimkan data jarak sebelumnya, apabila error terjadi. *Pseudo-code* pada bagian ini adalah sebagai berikut:

```
qSRF = Queue()
tSRF = Thread(target=self._distance)
tSRF.daemon = True
tSRF.start()
def _distance(self):
    while True:
       arduinoSerial.flush()
       arduinoSerial.write(b'a')
       srf_distance =
arduinoSerial.readline().decode('ascii').strip()
       try:
          distance = int(srf_distance) / 100.0
          if not qSRF.empty():
             try: qSRF.get_nowait()
             except: pass
          self.qSRF.put(distance)
       except: pass
```

```
distance = qSRF.get()
if distance is None or distance is 0:
    return prevDistance
else:
    prevDistance = distance
    return distance
```

Untuk penghitungan jarak sendiri menggunakan *library* yang telah tersedia pada Arduino.

Cara kerja dari SRF05 sendiri adalah mengirim sinyal *ultrasonic* dan dari pin *Trigger* yang diaktifkan dan akan menerima waktu *PulseIn* dari pin *Echo*. Selanjutnya akan dihitung jaraknya dengan menggunakan rumus:

$$dist = \frac{(dur/29)}{2} \tag{3.5}$$

dengan $dist$ merupakan jarak hasil perhitungan dan dur merupakan lama waktu dari pengiriman sinyal *ultrasonic* hingga ditangkap kembali oleh sensor. Berikut ini adalah *syntax* dari pengolahan, penerimaan, dan pengiriman data dengan Arduino:

```
void setup() {
  // put your setup code here, to run once:
  pinMode(TRIG, OUTPUT);
  pinMode(ECHO, INPUT);
  Serial.begin(9600);}
void loop() {
  if(Serial.available()>0){
    userInput = Serial.read();
    if(userInput == 'a'){
        // put your main code here, to run repeatedly:
        digitalWrite(TRIG, LOW);
        delayMicroseconds(2);
        digitalWrite(TRIG, HIGH);
        delayMicroseconds(10);
        digitalWrite(TRIG, LOW);

        const unsigned long dur = pulseIn(ECHO, HIGH);
        int distance = dur/29/2;
        if(dur==0 || distance > 320){
          Serial.println(0);}
        else{
          Serial.println(distance); }}}}
```

3.4.2.Penyimpanan dan Pengeluaran Nilai Kalibrasi

Penyimpanan dan pengeluaran ini dilakukan agar program menjadi lebih efektif dengan cara menyimpan *file* hasil kalibrasi ke suatu *file* dengan format txt berdasarkan area yang akan dideteksi. Tujuan algoritma ini adalah agar nilai kalibrasi tidak berubah – ubah dan dapat disesuaikan dengan keadaan. Terdapat enam data yang disimpan disetiap *file* nya yaitu *lower hue, upper hue, lower sat, upper sat, lower value,* dan *upper value* yang dipisah oleh tanda ‘,’. Setiap *file* yang akan dibuka juga diberikan kode berupa *int* 1 dan *int* 2 untuk memudahkan membuka file. Berikut *syntax* untuk menyimpan nilai kalibrasi sehingga dapat digunakan di dalam program utama:

```
def saveFile(detectionType, lowerHue, lowerSat, lowerVal,
upperHue, upperSat, upperVal):
    if detectionType == 1:
        file = 'inner.txt'
    elif detectionType == 2:
        file = 'middle.txt'
    elif detectionType == 3:
        file = 'outer.txt'
    else:
        file = None

    if file is not None:
        with open(file, 'w') as write:

write.write(f"{lowerHue},{lowerSat},{lowerVal},{upperHue},{upp
erSat},{upperVal}")
    else:
        print("File save error")
        exit()
```

Berikut ini adalah *syntax* untuk mengeluarkan nilai setiap hasil kalibrasi pada program utama:

```
def openFile(detectionType):
    if detectionType == 1:
        file = 'inner.txt'
    elif detectionType == 2:
        file = 'middle.txt'
    elif detectionType == 3:
        file = 'outer.txt'
    else:
        file = None

    if file is not None:
        with open(file, 'r') as read:
            rangeOfHSV = read.readline().split(',')
            lowerHue, lowerSat, lowerVal, upperHue, upperSat,
upperVal = rangeOfHSV
            return int(lowerHue), int(lowerSat),
int(lowerVal), int(upperHue), int(upperSat), int(upperVal)
    else:
        print("Open file error")
```

Halaman ini sengaja dikosongkan

BAB 4
PENGUJIAN DAN ANALISIS

Pada bab pengujian dan analisis akan dibahas mengenai pengujian dan analisa sistem. Pengujian dilakukan pada setiap bagian perancangan mulai dari pengujian perangkat keras dengan cara mengamati *drone* yang mampu terbang dengan stabil dengan berbagai macam mode penerbangan. Pengujian kedua adalah pengujian *drone* untuk melakukan pendaratan secara presisi pada area pendaratan yang tidak digerakkan. Pengujian yang ketiga adalah pengujian *drone* untuk melakukan pendaratan secara presisi pada area pendaraan yang digerakkan secara manual. Pengujian sendiri dilakukan di beberapa tempat yaitu di Lapangan Gedung Forensik ITS, Lapangan Gedung Transportasi Laut ITS, dan Perumahan Sukolilo Park Regency.

4.1. Pengujian Sensor Jarak

Pengujian sensor jarak memiliki tujuan untuk melakukan pengecekan keberhasilan sensor di dalam melakukan pengukuran jarak dan mengecek rentang jarak yang dapat dideteksi oleh sensor jarak. Sensor jarak yang digunakan adalah sensor *ultrasonic* SRF05 yang akan diuji dengan menempatkan suatu plat datar pada jarak tertentu dari sensor SRF05 dan akan diambil data untuk melihat akurasi dari sensor jarak di dalam mendeteksi jarak.

Tabel 4. 1 Pengujian Jarak 100 cm

No	Jarak (cm)	Terbaca (cm)
1	100	100.00
2		99.00
3		90.00
4		100.00
5		100.00
6		101.00
7		88.00
8		88.00
9		100.00
10		100.00
Rata - rata		96.60

Percobaan pertama adalah dengan menguji sensor dengan jarak

aktual sebesar 100 cm. Dari pengujian ini didapatkan hasil jarak rata – rata sekitar 96.60 cm. Pengujian yang tidak sesuai dapat terjadi karena sensitivitas sensor terhadap pembacaan, sehingga ketika jarak terlalu berubah maka pembacaan menjadi lebih kritis.

Tabel 4. 2 Tabel Pengujian Jarak 200 cm

No	Jarak (cm)	Terbaca (cm)
1	200	201.00
2		200.00
3		201.00
4		201.00
5		200.00
6		200.00
7		201.00
8		201.00
9		201.00
10		201.00
Rata - rata		200.70

Percobaan kedua adalah dengan menguji sensor dengan jarak aktual sebesar 200 cm. Dari pengujian ini didapatkan rata – rata jarak sekitar 200.70 cm. Ini membuktikan bahwa sensor jarak mampu mendeteksi jarak dengan baik meskipun memiliki sensitivitas yang tinggi.

Tabel 4. 3 Tabel Pengujian Jarak 250 cm

No	Jarak (cm)	Terbaca (cm)
1	250	135.00
2		247.00
3		249.00
4		250.00
5		205.00
6		249.00
7		248.00
8		248.00
9		248.00
10		249.00
Rata - rata		232.80

Percobaan ketiga adalah dengan menguji sensor dengan jarak aktual sebesar 250 cm. Dari pengujian ini didapatkan rata – rata jarak sekitar 232.80 cm. Dari pengujian ini dapat dilihat bahwa pembacaan sensor jarak yang sudah tidak akurat. Hal ini dapat terjadi karena permukaan

yang tidak datar ataupun ada benda – benda lain yang terdeteksi. Selain itu, hal ini juga dapat disebabkan oleh kemampuan dari sensor *ultrasonic* sendiri yang tidak mampu untuk mendeteksi dengan baik ketika jarak lebih dari 200 cm.

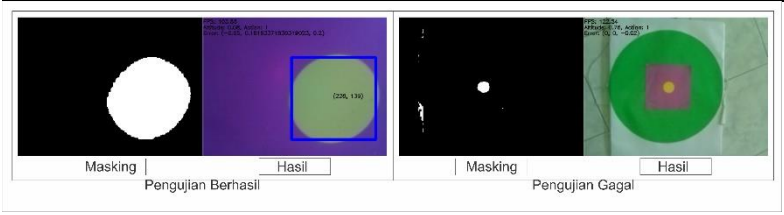
Dari pengujian sensor jarak ini, didapatkan hasil bahwa sensor jarak SRF05 memiliki kemampuan untuk melakukan pendeteksian jarak yang cukup akurat dengan rentang dari 0 – 200 cm.

4.2. Pengujian Pendeteksian Objek

Pengujian pendeteksian objek memiliki tujuan untuk melakukan pengecekan pendeteksian objek sesuai dengan objek yang ingin dideteksi dan jarak pendeteksian objek yang optimal. Pada pengujian ini, akan dilakukan pendeteksian objek sesuai dengan jarak hingga objek tidak terdeteksi dengan baik.

Tabel 4. 4 Pendeteksian Jarak Dekat

No	Jarak (cm)	Hasil
1	10	Berhasil
2	20	Berhasil
3	25	Berhasil
4	30	Berhasil
5	35	Berhasil
6	40	Berhasil
7	45	Berhasil
8	50	Berhasil
9	60	Tidak Berhasil
10	75	Tidak Berhasil



Gambar 4.1. Hasil Pengujian Jarak Dekat

Pada percobaan jarak dekat, dilakukan uji coba dengan melakukan pendeteksian lingkaran kuning. Dari seluruh percobaan yang dilakukan didapatkan hasil bahwa pendeteksian yang dilakukan untuk mendeteksi objek jarak dekat berhasil dilakukan untuk jarak 0 – 50 cm. Sedangkan

untuk jarak 60 cm ke atas gagal dilakukan. Hal ini dapat terjadi, karena objek dapat terdeteksi tetapi tidak lolos tahap *filtering* luas yang dilakukan untuk mengurangi *noise*. Sesuai dengan gambar 4.1 sebelah kiri dapat dilihat bahwa objek sudah terdeteksi pada tahap *thresholding* dan *filtering*. Akan tetapi, pada gambar di sebelah kanan dapat dilihat bahwa obje sudah terdeteksi pada tahap *thresholding* namun tidak terdeteksi pada tahap *filtering* berdasarkan luas. Dapat dilihat pula, *filtering* memiliki fungsi untuk menghindari objek – objek lain yang kemungkinan terdeteksi seperti pada gambar 4.1.

Tabel 4. 5 Pendeteksian Jarak Menengah

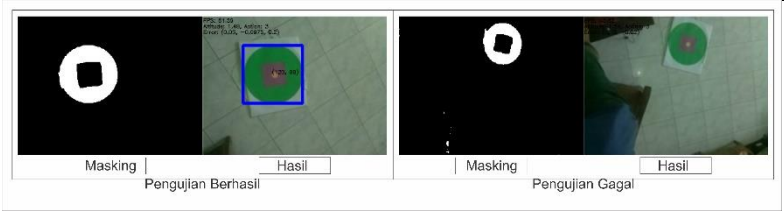
No	Jarak (cm)	Hasil
1	60	Berhasil
2	65	Berhasil
3	70	Berhasil
4	75	Berhasil
5	80	Berhasil
6	85	Berhasil
7	90	Berhasil
8	100	Tidak Berhasil
9	110	Tidak Berhasil
10	120	Tidak Berhasil

Gambar 4.2. Hasil Pengujian Jarak Menengah

Pada pengujian jarak menengah dilakukan dengan cara pendeteksian kotak berwarna ungu. Dari seluruh pengujian didapatkan hasil bahwa pendeteksian yang dilakukan untuk mendeteksi objek jarak menengah berhasil dilakukan untuk jarak antara 60 – 90 cm. Sedangkan untuk jarak 100 cm ke atas gagal dilakukan. Hal ini dapat terjadi karena objek yang terlalu gelap atau kalibrasi yang kurang sesuai sehingga objek tidak dapat terdeteksi seperti gambar 4.2 dimana objek tidak terdeteksi secara sempurna sehingga tidak lolos tahap *filtering* luas.

Tabel 4. 6 Pendeteksian Jarak Jauh

No	Jarak (cm)	Hasil
1	100	Berhasil
2	105	Berhasil
3	110	Berhasil
4	115	Berhasil
5	120	Berhasil
6	125	Berhasil
7	130	Berhasil
8	135	Tidak Berhasil
9	140	Tidak Berhasil
10	150	Tidak Berhasil



Gambar 4.3. Hasil Pendeteksian Jarak Jauh

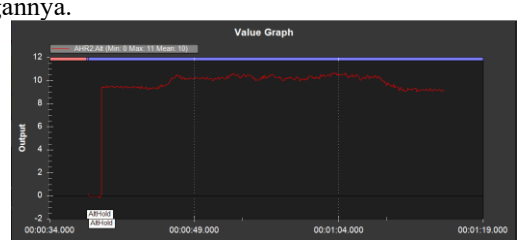
Pada pengujian jarak jauh dilakukan dengan cara pendeteksian lingkaran berwarna hijau. Dari seluruh pengujian didapatkan hasil bahwa pendeteksian yang dilakukan untuk mendeteksi objek jarak jauh berhasil dilakukan untuk jarak antara 100 – 130 cm. Sedangkan untuk jarak 135 cm ke atas gagal dilakukan. Hal ini dapat terjadi karena kalibrasi yang kurang sesuai sehingga objek tidak dapat terdeteksi seperti gambar 4.2 dimana objek tidak terdeteksi secara sempurna sehingga tidak lolos tahap *filtering* luas. Selain itu, hal ini juga disebabkan terlalu kecilnya objek yang terdeteksi sehingga hanya lolos pendeteksian *thresholding* tanpa lolos pendeteksian berdasarkan *filtering* luas.

Dari keseluruhan percobaan didapatkan hasil bahwa pendeteksian jarak dekat akan efektif pada jarak 0 – 50 cm, jarak menengah akan efektif pada jarak 60 – 90 cm, dan jarak jauh akan efektif pada jarak 100 – 130 cm. Ini membuktikan bahwa pemisahan pendeteksian berdasarkan jarak akan membantu *drone* di dalam melakukan pendaratan sehingga pendaratan dapat dilakukan dengan efektif karena pendeteksian objek yang spesifik.

4.3. Pengujian Navigasi Manual

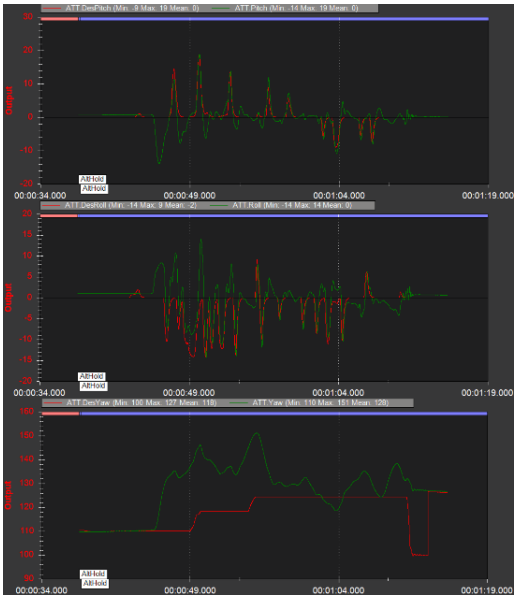
Pada bagian ini, *drone* akan melakukan pengujian navigasi manual. Pengujian ini dilakukan untuk mengecek kestabilan *drone* di dalam melakukan navigasinya. Pengujian ini dilakukan secara manual yang dikontrol dengan menggunakan *remote control*. Pengujian dilakukan dengan menggunakan beberapa mode penerbangan. Mode penerbangan manual yang dicoba antara lain adalah mode *Altitude Hold*, mode *Stabilize*, dan mode *Loiter*. Ketiga mode ini dipilih karena memiliki kesamaan dalam hal kontrol otomatis kestabilan *drone* untuk *pitch*, *yaw,* dan *roll*.

Mode *Altitude Hold* adalah salah satu mode yang dapat digunakan di dalam melakukan navigasi manual pada *drone*. Mode ini dilengkapi dengan sensor ketinggian sehingga pada mode ini, *drone* akan mempertahankan ketinggian nya dimana *pitch, yaw,* dan *roll* dapat dikontrol secara normal. Di dalam mode ini, *drone* akan mempertahankan ketinggian nya apabila *throttle* diberikan sekitar 50%. Apabila *throttle* diberikan kurang dari 50%, maka *drone* akan turun dan sebaliknya bila diberikan lebih dari 50%, maka *drone* akan naik. Sesuai dengan gambar 4.1, dapat dilihat, bahwa *drone* relatif tetap mempertahankan keseimbangannya.



Gambar 4.4. Grafik *Altitude* Mode *Altitude Hold*

Dari gambar 4.1, dapat dilihat bahwa dengan menggunakan mode *Altitude Hold*, *drone* akan berusaha untuk mempertahankan ketinggiannya. Sehingga perubahan ketinggian yang terjadi tidak terlalu signifikan.



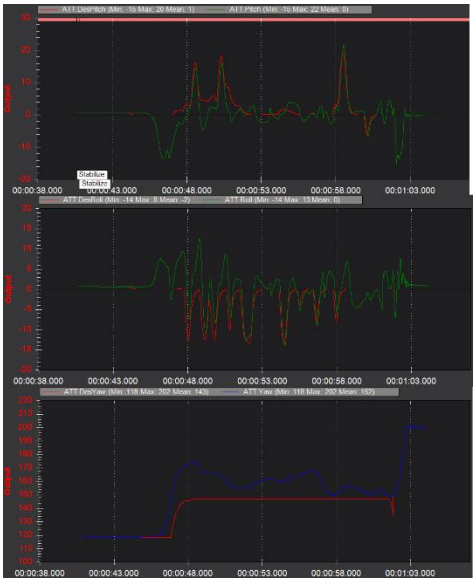
Gambar 4.5. Grafik *Pitch, Yaw, Roll* pada mode *Altitude Hold*

Dari gambar 4.5, dapat dilihat bahwa *drone* memiliki kecenderungan untuk melakukan *roll* ke kanan dan *pitch* ke belakang. Sedangkan *yaw* relatif selaras dengan kontrol dari *remote control*. Dapat dikatakan bahwa *drone* di dalam mode *Altitude Hold* tidak terdapat penyesuaian secara otomatis terhadap posisi. Jadi *drone* akan melakukan *drifting* sesuai dengan kondisi lingkungan maupun kondisi dari persebaran massa pada *drone*.

Mode kedua yang diujikan adalah mode *Stabilize*, dimana *drone* hanya mampu mempertahankan level posisinya saja tanpa mampu mempertahankan posisi kontrol dan ketinggiannya. Hal ini terjadi karena pada mode ini, *drone* tidak dilengkapi dengn sensor ketinggian maupun sensor untuk melakukan koreksi posisi.

Gambar 4.6. Grafik *Altitude* mode *Stabilize*

Dari gambar 4.6, dapat dilihat bahwa *altitude* mengalami perubahan yang signifikan setiap detiknya. Hal ini menandakan di dalam mode *Stabilize*, *flight controller* tidak mampu melakukan kontrol posisi secara otomatis dan kontrol posisi harus dilakukan secara manual dengan menggunakan *remote control*.



Gambar 4.7. Grafik *Pitch, Yaw*, *Roll* pada mode *Stabilize*

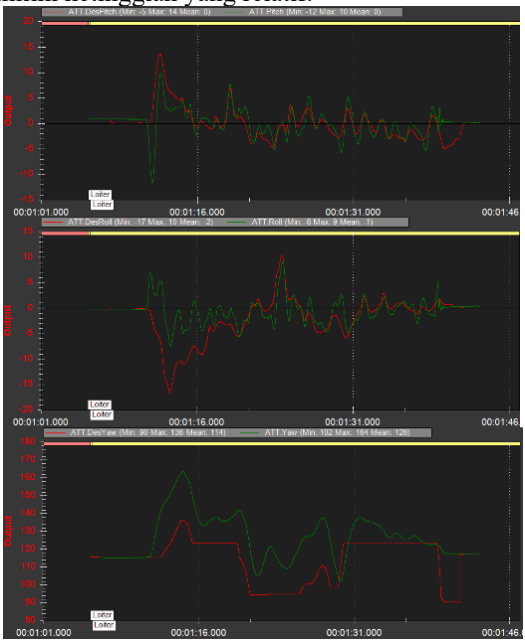
Dari grafik pada gambar 4.7 dapat dilihat bahwa *drone* tidak mampu mempertahankan posisinya, dan memiliki kecenderungan untuk *roll* ke kanan dan *pitch* ke belakang yang menandakan bahwa tidak adanya kontrol koreksi posisi.

Mode berikut yang akan diuji coba adalah mode *Loiter*, pada mode ini *drone* dilengkapi dengan sensor ketinggian dan sensor untuk

melakukan koreksi posisi dengan menggunakan GPS.

Gambar 4.8. Grafik *Altitude* mode *Loiter*

Dari grafik yang ditunjukkan pada gambar 4.8, dapat dilihat setelah *drone* melakukan *takeoff* dan telah sampai pada ketinggian tertentu, *drone* akan tetap memiliki ketinggian yang relatif.



Gambar 4.9. *Pitch, Yaw, Roll* pada mode *Loiter*

Dari grafik yang didapatkan pada gambar 4.9, dapat dilihat baik dari kontrol *pitch, yaw,* dan *roll* tidak berbeda jauh dengan kontrol dari *remote control*. Ini membuktikan bahwa *drone* melakukan kontrol sesuai dengan perintah yang diberikan oleh *remote control* dan juga membuktikan bahwa *drone* mampu mempertahankan posisi dengan baik.

Dari keseluruhan uji coba ketiga mode peneerbangan diatas,

ditemukan bahwa *drone* memiliki kecenderungan untuk melakukan *roll* ke kanan dan *pitch* ke belakang saat melakukan *takeoff* meskipun masih relatif stabil. Dengan adanya penambahan sensor ketinggian maka *drone* mampu untuk mempertahankan ketinggian dengan baik. Selain itu, dengan penambahan suatu sensor atau komponen yang mampu mempertahankan posisi, maka *drone* mampu memepertahankan posisi dengan baik seperti pada mode *Loiter* yang dikoreksi posisinya oleh GPS.

4.4. Pengujian Pendaratan *Drone* pada Area Pendaratan

Pada subbab ini dilakukan pengujian pendaratan *drone* pada area pendaratan di darat tanpa menggerakkan area pendaratan. Tujuan dari pengujian ini adalah untuk menentukan kecepatan pemosisian *drone* dengan menggunakan kontrol *pitch,* dan *roll*. Selain itu, pengujian ini dilakukan untuk menguji efektivitas dari sistem kontrol PID yang digunakan di dalam melakukan kontrol dari drone. Pengujian sendiri dilakukan menjadi tiga tahapan untuk melihat kecepatan yang paling sesuai bagi *drone* untuk melakukan pemosisian. Pengujian dilakukan dengan cara penerbangan secara manual oleh pilot hingga menemukan area pendaratan dan mengganti mode ke mode *Guided* untuk menjalankan program. Drone sendiri akan diuji dengan kecepatan turun sebesar 20 cm per detik dengan ketinggian maksimal 2 m sebelum drone memasuki mode *force Land*. *Drone* dikatakan mendarat dengan presisi apabila terdapat bagian dari *drone* yang berada di atas area pendaratan.

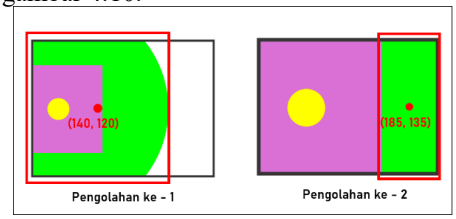
Pada percobaan pertama dengan mengatur PID X (P=0.00002, I=0.0, D=0.0) dan PID Y (P=0.00002, I=0.0, D=0.0) sehingga mendapatkan kecepatan minimal pergerakan hingga 0.02 mm/pengolahan. Dengan menggunakan pengaturan ini didapatkan deviasi rata – rata sekitar 46.40 cm dan waktu pendaratan rata – rata sekitar 23. 47 detik. Dengan menggunakan pengaturan kecepatan posisi sebagai berikut maka didapatkan deviasi minimal sebesar 33 cm dan deviasi maksimal sebesar 66 cm.

Tabel 4. 7 Tabel Percobaan Pertama

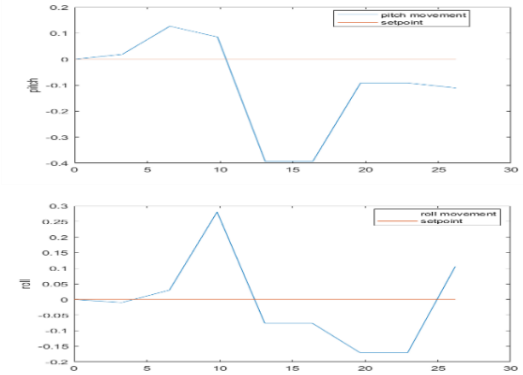
Pengujian	Deviasi (cm)	Waktu (s)
1	66.00	22.99
2	61.00	31.83
3	46.00	19.04
4	34.00	20.95
5	44.00	23.78

6	40.00	29.77
7	45.00	17.86
8	33.00	26.21
9	40.00	19.18
10	55.00	23.13
Rata - rata	46.40	23.47

Dari gambar 4.11 yang merupakan data dari pengujian ke-8 dari percobaan pertama dapat dilihat bahwa *drone* hanya memiliki delapan pergerakan sehingga *drone* tidak mampu melakukan pergerakan menengah dengan baik. Hal ini mungkin dapat disebabkan karena lambatnya pendeteksian program yang tidak sejalan dengan cepatnya pemberian perintah MAVLink sehingga *drone* masih menjalankan perintah deteksi dan kontrol sebelumnya meskipun ketinggian dari *drone* sudah berkurang. Hal tersebut terus terjadi hingga *drone* melakukan pendaratan tanpa adanya koreksi ketinggian yang dijelaskan secara visual sesuai dengan gambar 4.10.



Gambar 4.10. Visualiasi Pengolahan



Gambar 4.11. Pergerakan *Pitch* dan *Roll* Percobaan Pertama

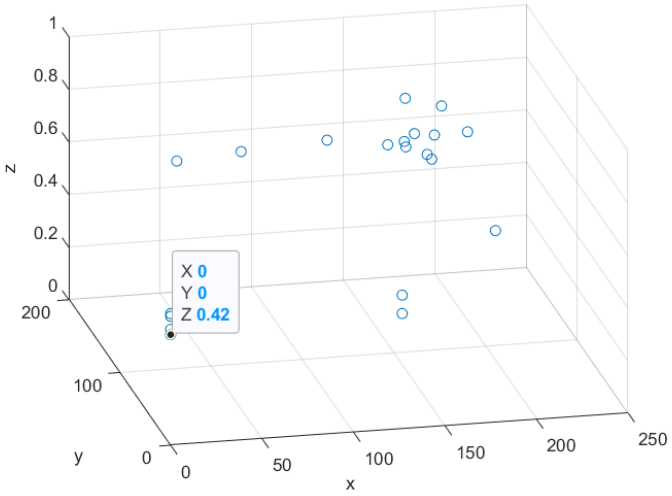
Pada percobaan kedua dengan mengatur PID X (P=0.0002,

I=0.0, D=0.00005) dan PID Y (P=0.00025, I=0.0, D=0.00001) sehingga mendapatkan kecepatan minimal pergerakan hingga 0.2 mm/pengolahan. Dengan menggunakan pengaturan ini didapatkan deviasi rata – rata sekitar 29.70 cm dan waktu pendaratan rata – rata sekitar 25.65. Deviasi minimal dari pengaturan ini adalah 12 cm dan deviasi maksimalnya adalah 52 cm. Pada percobaan kelima yang mendapatkan error maksimal terjadi karena lambatnya gerakan pergerakan menengah sehingga ketika ketinggian *drone* sudah mulai berkurang, objek yang seharusnya dideteksi pada ketinggian tersebut tidak terdeteksi. Berikut ini adalah tabel percobaan kedua.

Tabel 4. 8 Tabel Percobaan Kedua

Pengujian	Deviasi (cm)	Waktu (s)
1	38.00	28.33
2	30.00	37.04
3	34.00	25.66
4	33.00	20.53
5	52.00	20.42
6	26.00	15.32
7	37.00	23.15
8	18.00	33.78
9	12.00	24.17
10	17.00	28.14
Rata - rata	29.70	25.65

Pada gambar 4.12 yang didapatkan dari pengujian ke-9 dari percobaan kedua dapat dilihat bahwa terdapat jarak yang cukup lebar dari tiap titik pada lintasan pendaratan yang membuktikan bahwa pergerakan menengah yang dilakukan terlalu lambat. Penyebab lain lambatnya hal ini juga dapat disebabkan oleh kurangnya kecepatan di dalam eksekusi setiap algoritma di dalam program.



Gambar 4.12. Lintasan Pendaratan pada Percobaan Kedua

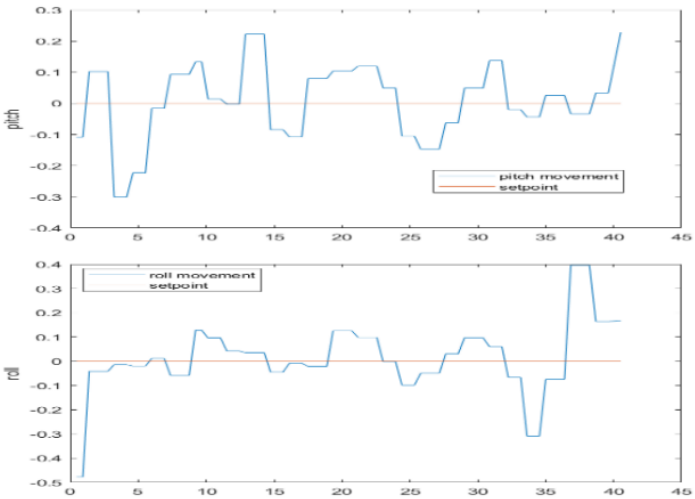
Pada percobaan ketiga dengan mengatur PID X (P=0.0025, I=0.0, D=0.0005) dan PID Y (P=0.0025, I=0.0, D=0.0002) sehingga mendapatkan kecepatan minimal pergerakan hingga 2.5 mm/pengolahan. Dengan menggunakan pengaturan ini didapatkan deviasi rata – rata sekitar 17.2 cm dan waktu pendaratan rata – rata sekitar 34.93. Deviasi minimal dari pengaturan ini adalah 9 cm dan deviasi maksimalnya adalah 38 cm. Pada pengujian ke-7 yang mendapatkan error paling tinggi dapat disebabkan oleh terlalu cepatnya kecepatan turun sehingga *drone* belum selesai melakukan *pergerakan menengah* tetapi ketinggian sudah berubah. Ketika ketinggian kurang dari 10 cm, *drone* akan mengganti mode menjadi mode "LAND" sehingga program akan mati dan posisi *drone* belum tepat berada di tengah.

Tabel 4. 9 Tabel Percobaan Ketiga

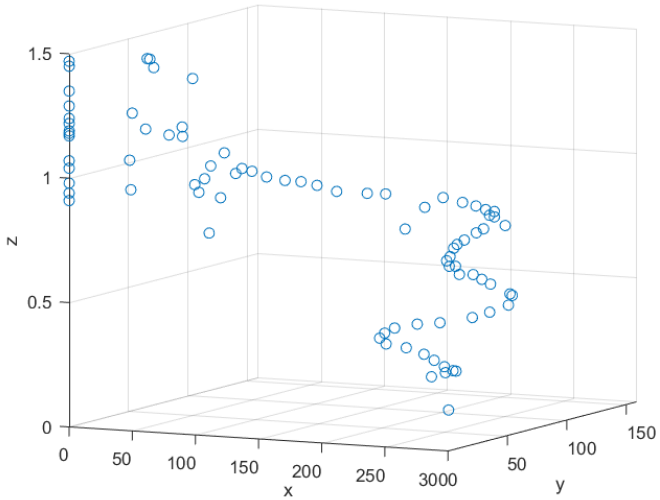
Pengujian	Deviasi (cm)	Waktu (s)
1	20.00	35.97
2	25.00	32.47
3	10.00	37.36
4	10.00	34.42
5	15.00	35.56
6	12.00	25.24

7	38.00	36.93
8	9.00	40.53
9	23.00	29.98
10	10.00	40.84
Rata - rata	17.20	34.93

Dari grafik pada gambar 4.13 dapat dilihat bahwa dengan meningkatkan kecepatan atau dalam hal ini mengubah PID akan membuat *drone* menjadi responsive sehingga *drone* memiliki kecenderungan untuk melakukan pergerakan menengah dengan lebih cepat. Pada gambar 4.14 juga dapat dilihat bahwa pergerakan *drone* untuk melakukan pergerakan menengah menjadi lebih baik.



Gambar 4. 13. Pergerakan *Pitch* dan *Roll* Percobaan Ketiga



Gambar 4. 14 Lintasan Pendaratan Percobaan Ketiga

Dari ketiga percobaan diatas, dapat dilihat bahwa *drone* mampu mendarat secara presisi di atas area pendaratan dengan pertimbangan salah satu bagian dari *drone* berada diatas area pendaratan. Dapat dilihat bahwa dengan kecepatan koreksi posisi yang baik per pengolahan dapat membuat *drone* melakukan pergerakan koreksi posisi dengan lebih baik. Selain itu, semakin tinggi akurasi dari proses koreksi posisi, maka akan semakin lama waktu pengolahan data.

4.5. Pengujian *Drone* pada Area Pendaratan Darat Bergerak

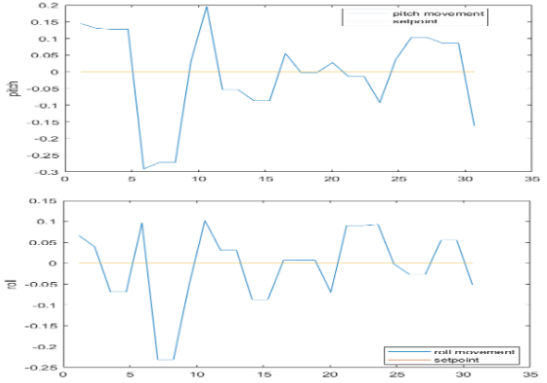
Pada subbab ini dilakukan pengujian pendaratan *drone* pada area pendaratan di darat dengan menggerakkan area pendaratan. Tujuan dari percobaan ini adalah untuk mengukur akurasi *drone* dengan kecepatan pergeseran posisi minimal sebesar 2.5 mm/pengolahan. Pada pengujian ini, area pendaratan dilakukan dengan penerbangan manual menuju ke area pendaratan. Selanjutnya, area pendaratan akan digerakkan dengan tali dengan kecepatan maksimal hingga 10 cm/detik. *Drone* diharapkan mampu melakukan pendaratan secara presisi pada area pendaratan. Sama dengan subbab sebelumnya, pendaratan dikatakan presisi bila terdapat bagian *drone* yang berada tepat di atas area pendaratan.

Pengujian sendiri dilakukan dalam dua tahap yaitu dengan kecepatan vertikal sebesar 50 cm/detik dan kecepatan vertikal sebesar 20 cm/detik. Setiap percobaan dilakukan sebanyak 10 kali pengujian dengan ketinggian awal maksimal sebesar 2 m.

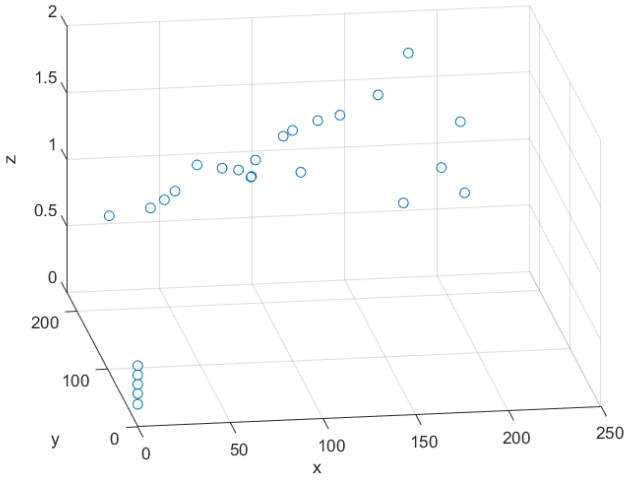
Tabel 4. 10 Tabel Pengujian Percobaan Pertama Area Pendaratan Bergerak

Pengujian	Deviasi (cm)	Waktu (s)
1	144.00	32.45
2	80.00	20.87
3	120.00	17.76
4	80.00	29.45
5	73.00	21.87
6	95.00	21.34
7	120.00	26.54
8	56.00	23.76
9	45.00	30.67
10	120.00	18.45
Rata - rata	93.30	24.32

Pada percobaan pertama dengan pengaturan kecepatan vertikal sebesar 50 cm/detik. Pada percobaan ini didapatkan hasil dengan deviasi rata – rata sekitar 93.30 cm dan rata – rata waktu sekitar 24.32 detik. Deviasi minimal yang terjadi adalah 45 cm dan deviasi maksimal yang terjadi adalah 144 cm.



Gambar 4. 15 Pergerakan *Pitch* dan *Roll* Percobaan Pertama



Gambar 4. 16 Lintasan Pergerakan Percobaan Pertama Area Pendaratan Bergerak

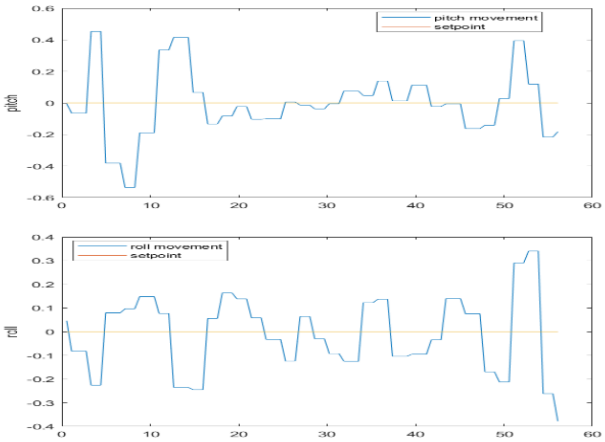
Pada percobaan pertama dapat dilihat pada grafik yang terdapat pada gambar 4.15, bahwa data yang didapatkan sedikit sehingga pengolahan data menjadi kurang baik. Hal ini dapat terjadi karena terlalu cepatnya *drone* untuk melakukan pergerakan vertikal yang tidak berbanding dengan kecepatan pergerakan ke titik tengah dari area pendaratan. Pada gambar 4.16 yang menjelaskan tentang lintasan pergerakan juga dapat dilihat bahwa posisi *drone* yang berpindah dari satu titik ke titik yang lainnya relatif jauh yang menandakan bahwa pendeteksian kurang optimal karena terlalu cepatnya pergerakan turun. Selain itu dapat dilihat pula dari waktu berjalannya program hingga selesai memiliki waktu dibawah 35 detik.

Tabel 4. 11 Tabel Pengujian Percobaan Kedua Area Pendaratan Bergerak

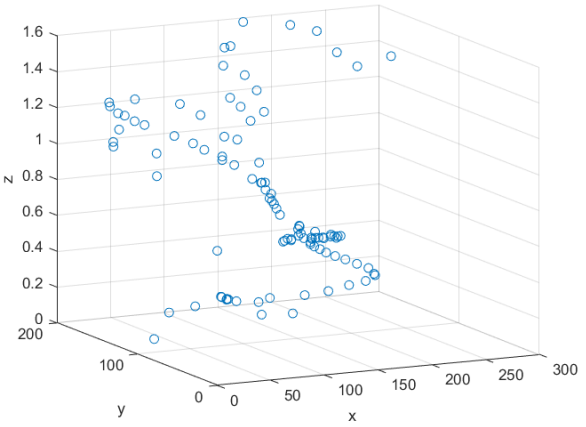
Pengujian	Deviasi(cm)	Waktu (s)
1	15.00	52.44
2	11.00	42.83
3	10.00	66.08
4	92.00	44.06
5	15.00	54.43
6	23.00	68.97
7	16.00	32.84

8	103.50	68.97
9	57.00	55.03
10	10.00	56.09
Rata - rata	35.25	54.17

Pada percobaan kedua dengan pengaturan kecepatan vertikal sebesar 20 cm/detik. Pada percobaan ini didapatkan hasil dengan deviasi rata – rata sekitar 35.25 cm dan rata – rata waktu sekitar 54.17 detik. Deviasi minimal yang terjadi adalah 10 cm dan deviasi maksimal yang terjadi adalah 103.5 cm. Pada pengujian ke-4 dan ke-9 yang memiliki error lebih dari 50 cm terjadi karena kesulitan pilot untuk melakukan penengahan atau pendeteksian pertama ke area pendaratan sebelum diganti menjadi mode *Guided*. Ketika hal ini terjadi, maka *drone* memiliki kemungkinan untuk mendeteksi objek lain tanpa adanya koreksi dengan menambahkan ketinggian untuk mendeteksi objek. Pada percobaan ke-8 yang memiliki error lebih dari 1 m terjadi karena *drone* melewati ambang batas ketinggian maksimal untuk menjalankan program atau dengan kata lain *drone* terbang lebih dari 2 meter sebelum menjalankan program sehingga sistem keamanan *drone* berjalan yaitu akan mengganti mode secara paksa menjadi mode *Land*.



Gambar 4. 17 Pergerakan *Pitch* dan *Roll* Percobaan Kedua



Gambar 4. 18 Lintasan Pergerakan Percobaan Kedua Area Pendaratan Bergerak

Dari grafik yang ditunjukkan pada gambar 4.17 dapat dilihat bahwa pergerakan *drone* baik pergerakan *roll* maupun *pitch* sangat fluktuatif. Hal ini menandakan bahwa *drone* terus melakukan koreksi posisi dengan baik karena banyaknya data dan pengolahan yang diberikan ke MAVLink. Pada gambar 4.18 juga dapat dilihat bahwa posisi *drone* untuk melakukan penengahan juga berjalan dengan baik dan bertahap.

Dari percobaan di subbab ini, didapatkan hasil bahwa *drone* mampu melakukan pendaratan presisi pada *platform* darat bergerak dengan akurasi yang cukup tinggi dengan pertimbangan *offset* pergerakan area pendaratan ketika penggantian mode penerbangan.

4.6. Spesifikasi Pendaratan Presisi pada Area Pendaratan Bergerak

Dari keseluruhan pengujian yang telah dilakukan didapatkan spesifikasi yang menunjukkan kemampuan *drone* di dalam penelitian ini untuk melakukan pendaratan pada *platform* darat bergerak sebagai berikut:

1. *Drone* mampu melakukan pendaratan pada *platform* darat bergerak dengan ketinggian awal hingga 200 cm sebelum memasuki mode *force Land*.
2. Objek yang akan dideteksi harus selalu dilakukan kalibrasi sehingga

drone dapat melakukan pendeteksian dengan lebih baik.
3. Pengolahan data citra dilakukan ketika data yang masuk berupa data citra dengan resolusi gambar 240p.
4. *Drone* bekerja secara efektif di dalam mode *Guided*.
5. *Drone* mampu melakukan pendaratan presisi pada *platform* darat bergerak dengan kecepatan pergerakan area pendaratan hinga 10 cm/detik.
6. *Drone* mampu melakukan pendaratan presisi pada *platform* darat bergerak dengan kecepatan koreksi posisi minimal hingga 2.5 mm/pengolahan.
7. *Drone* mampu melakukan pendaratan presisi pada *platform* darat bergerak dengan kecepatan turun hingga 20 cm/detik.
8. *Drone* mamu melakukan pendaratan presisi pada *platform* darat bergerak dengan akurasi hingga 1 m dan deviasi rata – rata sekitar 35.25 cm.

4.7. Perbandingan antara Metode Pendaratan Presisi

Pendaratan presisi dapat dihitung dengan cara melakukan pengukuran dengan menarik garis lurus dari titik tengah area pendaratan ke titik tengah dari *drone*. Setelah mempelajari studi literatur dapat disimpulkan bahwa pendaratan presisi merupakan berbagai cara yang dilakukan untuk melakukan pendaratan pada satu titik secara tepat dengan berbagai macam metode pendukung. Salah satu metode yang dikembangkan adalah dengan metode berbasis visi komputer. Pada tugas akhir ini dilakukan pengujian navigasi pendaratan pada area pendaratan tidak bergerak dengan rata – rata error sebesar 17.2 cm dengan rata – rata waktu 34.93 detik dan pengujian navigasi pada area darat bergerak dengan rata – rata error sebesar 35.25 cm dengan error minimal sebesar 10 cm yang memiliki rata – rata waktu sebesar 54.17 detik.

Tabel 4. 12 Rangkuman Pendaratan Presisi berbagai Sumber

No	Makalah	Error posisi pendaratan presisi
1	A Precise and GNSS-Free Landing System on Moving Platforms for Rotary-Wing UAVs[16]	Kurang dari 30 cm
2	A Vision-based Approach for Unmanned Aerial Vehicle Landing[7]	Rata – rata 0.0137 m
3	Automatic Navigation and Landing of An	Maksimal 6 cm

	Indoor AR Drone Quadrotor using ArUco Marker and Inertial Sensors[17]	
4	Autonomous Vision-based Target Detection and Safe Landing for UAV[18]	Kurang dari 1 m
5	Post-Mission Autonomous Return and Precision Landing of UAV[19]	Rata – rata 20 cm
6	Precision Landing Options in Unmanned Aerial Vehicles[8]	Rata – rata 30 cm
7	Precision Landing for Drone based on Computer Vision[10]	Rata – rata 0.45 cm
8	Perancangan Sistem Pendaratan Presisi *Drone* pada Objek Darat Bergerak berbasis Visi Komputer	Rata – rata 35.25 cm

Tabel 4.12 merupakan rangkuman berbagai macam penelitian mengenai sistem pendaratan presisi pada berbagai sumber yang dibandingkan dengan penelitian ini dengan berbagai macam kondisi yang berbeda sesuai dengan kebutuhan penelitian.

Halaman ini sengaja dikosongkan

BAB 5
PENUTUP

5.1. Kesimpulan

Berdasarkan hasil pengujian dan analisa data yang dilakukan, dapat disimpulkan:

1. Metode yang diusulkan berpotensi untuk menggantikan peran *Global Positioning System* dalam melakukan koreksi posisi ketika melakukan pendaratan.
2. Penambahan sensor ketinggian dan sensor untuk melakukan koreksi posisi sangat berpengaruh untuk menjaga stabilitas ketinggian dan posisi dari *drone* ketika melakukan penerbangan.
3. *Drone* mampu melakukan pendaratan pada area pendaratan yang diam dengan kecepatan turun sebesar 20 cm/s dan kecepatan perbaikan posisi hingga minimal 2.5 mm/pengolahan pada resolusi 240p dengan deviasi rata – rata sebesar 17.20 cm, waktu rata – rata 34.93 detik dan akurasi hingga 38 cm yang dihitung berdasarkan jarak antara pusat dari *drone* hingga pusat area pendaratan.
4. *Drone* mampu melakukan pendaratan presisi pada area pendaratan yang digerakkan dengan kecepatan hingga 10 cm/s dengan kecepatan pergerakan posisi hingga minimal 2.5 mm/pengolahan dan kecepatan turun sebesar 20 cm/s pada resolusi 240 p dengan deviasi rata – rata sebesar 35.25 cm, waktu rata – rata sebesar 54.17 dan akurasi hingga 1 m yang dihitung dari pusat dari *drone* hingga pusat area pendaratan.

5.2. Saran

Perlu dikembangkan metode koreksi ketika *drone* tidak mampu mendeteksi area pendaratan dengan memperbaiki algoritma kontrol posisi dengan cara menambahkan ketinggian dari *drone* agar area pendaratan dapat terlihat dengan maksimal atau dengan penambahan Kalman Filter pada algoritma pendeteksian.

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

[1] M. Półka, S. Ptak, and Ł. Kuziora, "The Use of UAV's for Search and Rescue Operations," *Procedia Eng.*, vol. 192, pp. 748–752, 2017, doi: 10.1016/j.proeng.2017.06.129.

[2] Ali Al-Sheary and Ali Almagbile, "Crowd Monitoring System Using Unmanned Aerial Vehicle (UAV)," *J. Civ. Eng. Archit.*, vol. 11, no. 11, Nov. 2017, doi: 10.17265/1934-7359/2017.11.004.

[3] J. Kwak and Y. Sung, "Autonomous UAV Flight Control for GPS-Based Navigation," *IEEE Access*, vol. 6, pp. 37947–37955, 2018, doi: 10.1109/ACCESS.2018.2854712.

[4] K. N. Tahar and S. S. Kamarudin, "UAV ONBOARD GPS IN POSITIONING DETERMINATION," *ISPRS - Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.*, vol. XLI-B1, pp. 1037–1042, Jun. 2016, doi: 10.5194/isprsarchives-XLI-B1-1037-2016.

[5] A. R. Vetrella and G. Fasano, "Cooperative UAV navigation under nominal GPS coverage and in GPS-challenging environments," in *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, Bologna, Italy, Sep. 2016, pp. 1–5, doi: 10.1109/RTSI.2016.7740606.

[6] X. Qin and T. Wang, "Visual-based Tracking and Control Algorithm Design for Quadcopter UAV," in *2019 Chinese Control And Decision Conference (CCDC)*, Nanchang, China, Jun. 2019, pp. 3564–3569, doi: 10.1109/CCDC.2019.8832545.

[7] C. Patruno, M. Nitti, A. Petitti, E. Stella, and T. D'Orazio, "A Vision-Based Approach for Unmanned Aerial Vehicle Landing," *J. Intell. Robot. Syst.*, vol. 95, no. 2, pp. 645–664, Aug. 2019, doi: 10.1007/s10846-018-0933-2.

[8] J. Janousek and P. Marcon, "Precision landing options in unmanned aerial vehicles," in *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, Swinoujście, May 2018, pp. 58–60, doi: 10.1109/IIPHDW.2018.8388325.

[9] A. El-Badawy and R. Rashad, "Ultrasonic Rangefinder Spikes Rejection Using Discrete Wavelet Transform: Application to UAV," *J. Sens. Technol.*, vol. 05, no. 02, pp. 45–53, 2015, doi: 10.4236/jst.2015.52005.

[10] P. M. Situmorang, A. N. Irfansyah, and M. Attamimi, "Precision Landing for Drone based on Computer Vision," p. 6.

[11] A. T. Nguyen, N. Xuan-Mung, and S.-K. Hong, “Quadcopter Adaptive Trajectory Tracking Control: A New Approach via Backstepping Technique,” *Appl. Sci.*, vol. 9, no. 18, p. 3873, Sep. 2019, doi: 10.3390/app9183873.
[12] S. Vyas, C. Rajurkar, and K. Venkatasubramanian, “Unmanned Secured Delivery System,” *Int. J. Recent Innov. Trends Comput. Commun.*, vol. 4, no. 3, p. 5.
[13] O. Tatale, N. Anekar, S. Phatak, and S. Sarkale, “Quadcopter: Design, Construction and Testing,” p. 8.
[14] T. Krajnik, M. Nitsche, S. Pedre, L. Preucil, and M. E. Mejail, “A simple visual navigation system for an UAV,” in *International Multi-Conference on Systems, Sygnals & Devices*, Chemnitz, Germany, Mar. 2012, pp. 1–6, doi: 10.1109/SSD.2012.6198031.
[15] N. Gageik, M. Strohmeier, and S. Montenegro, “An Autonomous UAV with an Optical Flow Sensor for Positioning and Navigation,” *Int. J. Adv. Robot. Syst.*, vol. 10, no. 10, p. 341, Oct. 2013, doi: 10.5772/56813.
[16] H. Anis, A. H. I. Fadhillah, S. Darma, and S. Soekirno, “Automatic Quadcopter Control Avoiding Obstacle Using Camera with Integrated Ultrasonic Sensor,” *J. Phys. Conf. Ser.*, vol. 1011, p. 012046, Apr. 2018, doi: 10.1088/1742-6596/1011/1/012046.
[17] F. Alarcón, M. García, I. Maza, A. Viguria, and A. Ollero, “A Precise and GNSS-Free Landing System on Moving Platforms for Rotary-Wing UAVs,” *Sensors*, vol. 19, no. 4, p. 886, Feb. 2019, doi: 10.3390/s19040886.
[18] M. F. Sani and G. Karimian, “Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors,” in *2017 International Conference on Computer and Drone Applications (IConDA)*, Kuching, Nov. 2017, pp. 102–107, doi: 10.1109/ICONDA.2017.8270408.
[19] M. Rabah, A. Rohan, M. Talha, K.-H. Nam, and S. H. Kim, “Autonomous Vision-based Target Detection and Safe Landing for UAV,” *Int. J. Control Autom. Syst.*, vol. 16, no. 6, pp. 3013–3025, Dec. 2018, doi: 10.1007/s12555-018-0017-x.
[20] G. Ostoji, S. Stankovski, and B. Teji, “Design, Control and Application of Quadcopter,” p. 7.
[21] “Introduction · MAVLink Developer Guide.” https://mavlink.io/en/ (accessed Apr. 20, 2020).
[22] A. Koubaa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M.

Khalgui, “Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey,” *IEEE Access*, vol. 7, pp. 87658–87680, 2019, doi: 10.1109/ACCESS.2019.2924410.

[23] A. Kaehler and G. Bradski, “COMPUTER VISION IN C++ WITH THE OPENCV LIBRARY,” p. 1018.

[24] N. Pm and D. R. M. Chezian, “VARIOUS COLOUR SPACES AND COLOUR SPACE CONVERSION ALGORITHMS,” p. 5, 2010.

[25] B. Y. Budi Putranto, W. Hapsari, and K. Wijana, “SEGMENTASI WARNA CITRA DENGAN DETEKSI WARNA HSV UNTUK MENDETEKSI OBJEK,” *J. Inform.*, vol. 6, no. 2, Feb. 2011, doi: 10.21460/inf.2010.62.81.

[26] Ukachi Osisiogu, “Seminar Paper on Serial Communication,” 2015, doi: 10.13140/RG.2.2.32809.85605.

Halaman ini sengaja dikosongkan

LAMPIRAN A

Program Utama

```
import time
import sys
from control import Drone, PID
from sensor import Camera, objectDetection, calculateFPS,
checkQuit, SRFThread, showFrame
from aconfigs import *
import argparse
from timeit import default_timer as timer
import dataRecord
import os

def parse():
    # Parse the arguments
    parser = argparse.ArgumentParser(prog='Dynamic Landing',
                            description='Tugas akhir Dionisius')
    parser.add_argument('--cam', default=0, help='0 - PiCamera, 1
- Logitech')
    args = parser.parse_args()
    # Get variables from parsing
    camNum = int(args.cam)
    return camNum

def setup():
    """
    Setting the drone
    :return:
    """
    camNumber = parse()
    drone = Drone()
    camera = Camera(camNumber=camNumber)
    altitude = SRFThread(ARDU_PORT)
    return drone, camera, altitude

def landingControl(camera, drone, altitude):
    rec = dataRecord.Record()
    option = 0
    groundPoint = altitude.readSRF()

    xCenter = -1
    yCenter = -1

    pidX = PID(P=KP_X, I=KI_X, D=KD_X)
    pidX.setPoint = SET_X
```

```
    pidY = PID(P=KP_Y, I=KI_Y, D=KD_Y)
    pidY.setPoint = SET_Y

    xList = []
    yList = []
    optList = []
    altitudeList = []
    pitchList = []
    yawList = []
    rollList = []

    drone.setMode("STABILIZE")
    prevMode = "STABILIZE"

    while True:
        ret, frame = camera.read()
        start = timer()

        if drone.checkMode() is not prevMode:
            prevMode = drone.checkMode()
            print(drone.checkMode())

        if drone.checkMode() is "GUIDED" or drone.checkMode() is
"GUIDED_NOGPS" or drone.checkMode() is "LAND":
            curDist = altitude.readSRF()
            if curDist <= SHORT:
                if curDist <= LAND:
                    option = 4
                    xCenter, yCenter, frame =
objectDetection(frame, option)
                    drone.setMode("LAND")
                    print(f"Drone landing : {curDist}")
                    if curDist <= groundPoint + 0.01:
                        break
                elif LAND < curDist <= SHORT:
                    option = 6
                    xCenter, yCenter, frame =
objectDetection(frame, option)
                    if xCenter < 0 and yCenter < 0:
                        option = 1
                        xCenter, yCenter, frame =
objectDetection(frame, option)
            elif SHORT < curDist <= MIDDLE:
                option = 2
                xCenter, yCenter, frame = objectDetection(frame,
option)
                if xCenter < 0 and yCenter < 0:
                    option = 5
```

```
                    xCenter, yCenter, frame =
objectDetection(frame, option)
            elif MIDDLE < curDist <= LONG:
                option = 3
                xCenter, yCenter, frame = objectDetection(frame,
option)
                if xCenter < 0 and yCenter < 0:
                    option = 7
                    xCenter, yCenter, frame =
objectDetection(frame, option)
            else:
                drone.setMode("LAND")
                print(f"Drone max altitude offset reached :
{curDist} m")
                break

            # CALCULATE PID
            xCenter = xCenter + 1
            yCenter = yCenter + 1

            if xCenter < 0 and yCenter < 0:
                vx, vy, vz = 0, 0, -0.2
            else:
                pidX.update(xCenter)
                pidY.update(yCenter)
                vx = pidY.output
                vy = pidX.output * (-1)
                vz = 0.2

                drone.sendMavlink(vx, vy, vz)
                fps = f"{str(round(1 / (timer() - start), 2))}"

                calculateFPS(fps, curDist, option, frame,
(xCenter, yCenter))

                xList.append(xCenter)
                yList.append(yCenter)
                optList.append(option)
                altitudeList.append(curDist)

                pitchList.append(drone.vehicle._pitchspeed)
                yawList.append(drone.vehicle._yawspeed)
                rollList.append(drone.vehicle._rollspeed)
        else:
            showFrame(frame)
            pidY.clearPID()
            pidX.clearPID()
```

```
        if checkQuit():
            if drone.checkMode() is "GUIDED" or drone.checkMode()
is "GUIDED_NOGPS":
                drone.setMode("LAND")
            else:
                pass
            print("Drone now landing...")
            break

    camera.release()
    drone.disArm()

    if drone.checkMode() is "GUIDED" or drone.checkMode() is
"GUIDED_NOGPS" or drone.checkMode() is "LAND":
        drone.disArm()
        # rec.stopRecord()
        rec.recordData(xList, yList, altitudeList, optList,
pitchList, yawList, rollList)
        csvSize = os.path.getsize(rec.csvName)
        if csvSize < 30:
            rec.deleteData(rec.csvName)
        else:
            print(f"{rec.csvName} is written !!!")


def main():
    """
    Main program
    :return:
    """
    try:
        # Setup vehicle
        startTimer = timer()
        drone, camera, altitude = setup()
        print(f"Drone will land at {altitude.readSRF()}")
        landingControl(camera, drone, altitude)

        print(f"Program ending at {timer() - startTimer}")
        sys.exit()
    except:
        sys.exc_info()


if __name__ == "__main__":
    main()
```

LAMPIRAN B

Library Sensor untuk Kamera dan SRF

```
from cv2 import VideoCapture, CAP_PROP_FRAME_WIDTH,
CAP_PROP_FRAME_HEIGHT
import cv2 as cv
from queue import Queue
from threading import Thread
from time import sleep
from serial import Serial
from aconfigs import *
from datetime import datetime


class Camera:
    def __init__(self, camNumber):
        self.camNumber = camNumber

        # Connect to Camera
        self.cap = VideoCapture(camNumber)
        self.cap.set(CAP_PROP_FRAME_WIDTH, DEFAULT_WIDTH)
        self.cap.set(CAP_PROP_FRAME_HEIGHT, DEFAULT_HEIGHT)

        # Camera Threading
        self.qCamera = Queue()
        self.tCamera = Thread(target=self._reader)
        self.tCamera.daemon = True
        self.tCamera.start()

    def _reader(self):
        while True:
            ret, frame = self.cap.read()
            frame = cv.rotate(frame, cv.ROTATE_180)
            if not ret:
                break
            if not self.qCamera.empty():
                try:
                    self.qCamera.get_nowait()
                except:
                    pass
            self.qCamera.put((ret, frame))

    def read(self):
        return self.qCamera.get()

    def release(self):
        self.cap.release()
        cv.destroyAllWindows()
```

```
class SRFThread:
    def __init__(self, port):
        # Serial connection to Arduino
        self.port = port
        print("Connecting to arduino...")
        self.arduinoSerial = Serial(port=self.port,
baudrate=ARDU_BAUD, timeout=1)
        sleep(3)
        print("Arduino connected")
        self.prevDistance = 0.01

        # SRF Threading
        self.qSRF = Queue()
        tSRF = Thread(target=self._distance)
        tSRF.daemon = True
        tSRF.start()

    def _distance(self):
        while True:
            self.arduinoSerial.flush()
            self.arduinoSerial.write(b'a')
            srf_distance =
self.arduinoSerial.readline().decode('ascii').strip()

            try:
                distance = int(srf_distance) / 100.0
                if not self.qSRF.empty():
                    try:
                        self.qSRF.get_nowait()
                    except:
                        pass
                self.qSRF.put(distance)
            except:
                pass

    def readSRF(self):
        distance = self.qSRF.get()
        if distance is None or distance is 0:
            return self.prevDistance
        else:
            self.prevDistance = distance
            return distance

def objectDetection(frame, option):
    # frame = cv.bilateralFilter(frame, 10, 40, 40)
    frame = cv.GaussianBlur(frame, BLUR_GAUSSIAN, sigmaX=0)
```

```
    frameHSV = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
    lowerHue, lowerSat, lowerVal, upperHue, upperSat, upperVal =
openFile(option)

    # HSV value of landing pad
    lowerRange = np.array([lowerHue, lowerSat, lowerVal])
    upperRange = np.array([upperHue, upperSat, upperVal])
    maskHSV = cv.inRange(frameHSV, lowerRange, upperRange)
    dilationMask = cv.erode(maskHSV, MORPH_KERNEL, iterations=1)

    # Find contour
    contours, _ = cv.findContours(dilationMask, cv.RETR_TREE,
cv.CHAIN_APPROX_SIMPLE)

    # Declaration of center
    cogX = int(0)
    cogY = int(0)
    area = int(1)
    landingFound = False

    # Find contour using center of gravity
    for contour in contours:
        x, y, w, h = cv.boundingRect(contour)

        areaLuas = w * h
        if option == 1:
            minim = MAX_1
        elif option == 2:
            minim = MAX_2
        elif option == 3:
            minim = MAX_3
        elif option == 4:
            minim = 2000
        elif option == 5:
            minim = TRANS_MID
        else:
            minim = 500

        if w * h > minim:
            landingFound = True

            x1 = int(((x + x + w) / 2))
            y1 = int(((y + y + h) / 2))
            area += int(w * h)

            cogX += int(x1 * w * h)
            cogY += int(y1 * w * h)
            cv.rectangle(frame, (x, y), (x + w, y + h), RED,
```

```
thickness=3)

    if landingFound:
        cogXFinal = int(cogX / area)
        cogYFinal = int(cogY / area)
        return cogXFinal, cogYFinal, frame
    else:
        cogXFinal = -1
        cogYFinal = -1
        return cogXFinal, cogYFinal, frame


def calculateFPS(fps, dist, action, frame, centerPos):
    cv.putText(frame, f"{centerPos}", centerPos,
cv.FONT_HERSHEY_SIMPLEX, 0.3, (0, 0, 0))
    cv.putText(frame, f"FPS: {fps}", (1, 10),
cv.FONT_HERSHEY_SIMPLEX, 0.3, (0, 0, 0))
    cv.putText(frame, f"Altitude: {dist}, Action: {action}", (1,
20), cv.FONT_HERSHEY_SIMPLEX, 0.3, (0, 0, 0))
    cv.imshow("Result", frame)

def showFrame(frame):
    cv.imshow("Result", frame)

def checkQuit():
    key = cv.waitKey(60) & 0xFF
    if key == 27:
        return True
```

LAMPIRAN C

Library Kontrol *Drone*

```
from __future__ import print_function
from dronekit import connect, VehicleMode, LocationGlobal,
LocationGlobalRelative
from pymavlink import mavutil
import time
from threading import Thread
import math
from aconfigs import *
import sensor
from math import radians, cos, sin

class Drone:
    def __init__(self):
        """
        Drone initialize
        """
        print('Vehicle : Connecting to vehicle on: %s' %
MINIPIX_PORT)
        self.vehicle = connect(MINIPIX_PORT,
baud=MINIPIX_BAUDRATE, wait_ready=VEHICLE_WAIT_READY)

        # Time
        self.time_now = None
        self.time_past = None
        self.time_out = None

    def sendMavlink(self, vx, vy, vz):
        self.set_velocity_body(vx, vy, vz)

    def set_velocity_body(self, vx, vy, vz):
        msg =
self.vehicle.message_factory.set_position_target_local_ned_encode(
            0,
            0, 0,
            mavutil.mavlink.MAV_FRAME_BODY_OFFSET_NED,
            DEFAULT_MASK,  # -- BITMASK -> Consider only the
velocities
            0, 0, 0,  # -- POSITION
            vx, vy, vz,  # -- VELOCITY
            0, 0, 0,  # -- ACCELERATIONS
            0, 0)
        self.vehicle.send_mavlink(msg)

    def setMode(self, mode):
```

```
        self.vehicle.mode = VehicleMode(mode)

        timeout = time.time() + VEHICLE_SWITCH_MODE_TIMEOUT
        while self.vehicle.mode.name is not mode:
            if time.time() >= timeout:
                return False
        return True

    def checkMode(self, compare=None):
        """
        :return: current vehicle mode
        """
        if compare is None:
            return self.vehicle.mode.name
        else:
            return self.vehicle.mode == VehicleMode(compare)

    def disArm(self):
        self.vehicle.mode = VehicleMode("LAND")
        self.vehicle.armed = False
        self.vehicle.close()


class PID:
    def __init__(self, P=0.2, I=0.0, D=0.0, speedCutoff=MAX_SPEED,
current_time=None):
        self.Kp = P
        self.Ki = I
        self.Kd = D
        self.setPoint = 0
        self.speedCutoff = speedCutoff

        self.sample_time = 0.00
        self.current_time = current_time if current_time is not
None else time.time()
        self.last_time = self.current_time

        self.PTerm = 0.0
        self.ITerm = 0.0
        self.DTerm = 0.0
        self.last_error = 0.0

        self.int_error = 0.0
        self.windup_guard = 20.0

        self.output = 0.0

    def clearPID(self):
```

```
        self.PTerm = 0.0
        self.ITerm = 0.0
        self.DTerm = 0.0
        self.last_error = 0.0

        # Integral protector
        self.int_error = 0.0
        self.windup_guard = 20.0

        self.output = 0.0

    def update(self, value, current_time=None):
        error = self.setPoint - value
        self.current_time = current_time if current_time is not
None else time.time()
        delta_time = self.current_time - self.last_time
        delta_error = error - self.last_error

        if delta_time >= self.sample_time:
            self.PTerm = self.Kp * error
            self.ITerm += error * delta_time

            if self.ITerm < - self.windup_guard:
                self.ITerm = - self.windup_guard
            elif self.ITerm > self.windup_guard:
                self.ITerm = self.windup_guard

            self.DTerm = 0.0
            if delta_time > 0:
                self.DTerm = delta_error / delta_time

            self.last_time = self.current_time
            self.last_error = error

            self.output = self.PTerm + (self.Ki * self.ITerm) +
(self.Kd * self.DTerm)

            if self.speedCutoff is not None:
                if abs(self.output) > self.speedCutoff:
                    self.output = (self.output / abs(self.output))
/ self.speedCutoff
                    return self.output
                else:
                    return self.output

    def setKp(self, proportional_gain):
        self.Kp = proportional_gain
```

```
def setKi(self, integral_gain):
    self.Ki = integral_gain

def setKd(self, derivative_gain):
    self.Kd = derivative_gain

def setWindup(self, windup):
    self.windup_guard = windup

def setSampleTime(self, sample_time):
    self.sample_time = sample_time
```

LAMPIRAN D

Library Pengaturan Drone

```
"""
CONFIGURATION
"""

import numpy as np

# CONTROL
MINIPIX_BAUDRATE = 115200
MINIPIX_PORT = "/dev/ttyACM0"
VEHICLE_WAIT_READY = False
VEHICLE_SWITCH_MODE_TIMEOUT = 4

DEFAULT_TAKEOFF = 0.58
SMOOTH_TAKEOFF = 0.53

MAX_1 = 20000  # 20000
MAX_2 = 2000  # 6000
MAX_3 = 5000  # 16500
TRANS_MID = 20000
TRANS_LOW = 10000
TRANS_UP = 5000
RANGE_LAND = 15000

LONG = 2
MIDDLE = 0.8
SHORT = 0.5
LAND = 0.1

ALTITUDE_DEFAULT = 0.5

# PID
SET_X = 160
SET_Y = 120
KP_X = 0.0025
KI_X = 0.0000
KD_X = 0.0005
KP_Y = 0.0025
KI_Y = 0.0000
KD_Y = 0.0002

MAX_SPEED = 0.5

# Type Mask Mavlink
DEFAULT_MASK = 0b0000111111000111  # Velocity only
GAIN_HEIGHT = 0b0000111111011111  # For gaining height
```

```
# VISION
CAMERA_DEFAULT = 0
DEFAULT_WIDTH = 320
DEFAULT_HEIGHT = 240
DEFAULT_MAX_VALUE = 255
DEFAULT_MAX_VALUE_HUE = 179
DEFAULT_CAMERA = 0
BLUR_GAUSSIAN = (3, 3)
MORPH_KERNEL = np.ones(BLUR_GAUSSIAN, np.uint8)
GREEN = (0, 255, 0)
RED = (255, 0, 0)
BLUE = (0, 0, 255)
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)

# ARDUINO
ARDU_PORT = "/dev/ttyUSB0"
ARDU_BAUD = 9600

# Check
CHECK_CAMERA = True

def nothing(x):
    pass

def openFile(detectionType):
    if detectionType == 1:
        file = 'inner.txt'
    elif detectionType == 2:
        file = 'middle.txt'
    elif detectionType == 3:
        file = 'outer.txt'
    elif detectionType == 4:
        file = 'landrange.txt'
    elif detectionType == 5:
        file = 'transmid.txt'
    elif detectionType == 6:
        file = 'translow.txt'
    elif detectionType == 7:
        file = 'transup.txt'
    else:
        file = None

    if file is not None:
        with open(file, 'r') as read:
            rangeOfHSV = read.readline().split(',')
            lowerHue, lowerSat, lowerVal, upperHue, upperSat,
```

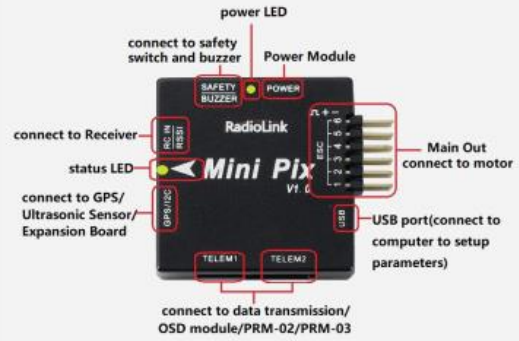
```
upperVal = rangeOfHSV
            return int(lowerHue), int(lowerSat), int(lowerVal),
int(upperHue), int(upperSat), int(upperVal)
    else:
        print("Open file error")
        exit()

def saveFile(detectionType, lowerHue, lowerSat, lowerVal,
upperHue, upperSat, upperVal):
    if detectionType == 1:
        file = 'inner.txt'
    elif detectionType == 2:
        file = 'middle.txt'
    elif detectionType == 3:
        file = 'outer.txt'
    elif detectionType == 4:
        file = 'landrange.txt'
    elif detectionType == 5:
        file = 'transmid.txt'
    elif detectionType == 6:
        file = 'translow.txt'
    elif detectionType == 7:
        file = 'transup.txt'
    else:
        file = None

    if file is not None:
        with open(file, 'w') as write:

write.write(f"{lowerHue},{lowerSat},{lowerVal},{upperHue},{upperSa
t},{upperVal}")
    else:
        print("File save error")
        exit()
```

LAMPIRAN E

Datasheet Komponen

1. *Flight Controller* RadioLink Mini Pix

1.2 Connectors



4

Parameters

Hardware	Processor	STM32F405VGT6
Sensor	Gyro	MPU6500
	Compass	QMC5883L
	Barometer	LPS22HB
	Connector	JST GH connector
	Mavlink UART	2(with CTSRTS)
	GPS UART	1
	RC In signal input	PPM/SBUS
	RSSI signal strength input	PWM/3.3V
	I2C	1
	PWM input	6 OneShot/PWM output
	Buzzer	1
	Switch	1
	Power	1
Compatible Models	Fixed wing/3 to 6 Multicopter/Helicopter/Cars/Boats	
Operating Parameters	PM operate voltage	5.1V
	USB voltage	5V+-0.3V
	Servo voltage	Do not support input
	Temperature	-40~80℃
Size & Weight	Size	39*39*12mm
	Weight	12g(without wires)

2. Raspberry Pi 3B+

2 Features

2.1 Hardware

- Low cost
- Low power
- High availability
- High reliability
  - Tested over millions of Raspberry Pis Produced to date
  - Module IO pins have 15 micro-inch hard gold plating over 2.5 micron Nickel

2.2 Peripherals

- 48x GPIO
- 2x I2C
- 2x SPI
- 2x UART
- 2x SD/SDIO
- 1x HDMI 1.3a
- 1x USB2 HOST/OTG
- 1x DPI (Parallel RGB Display)
- 1x NAND interface (SMI)
- 1x 4-lane CSI Camera Interface (up to 1Gbps per lane)
- 1x 2-lane CSI Camera Interface (up to 1Gbps per lane)
- 1x 4-lane DSI Display Interface (up to 1Gbps per lane)
- 1x 2-lane DSI Display Interface (up to 1Gbps per lane)

2.3 Software

- ARMv8 Instruction Set
- Mature and stable Linux software stack
  - Latest Linux Kernel support
  - Many drivers upstreamed
  - Stable and well supported userland
  - Full availability of GPU functions using standard APIs

5 Pin Assignments

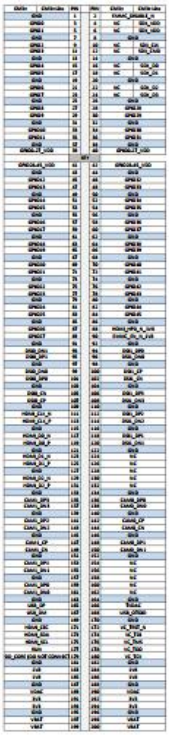


Table 2: Compute Module 3+ SODIMM Connector Pinout

Table 2 gives the Compute Module 3+ pinout and Table 3 gives the pin functions.

Pin Name	DIR	Voltage Ref	PDN[a] State	If Unused	Description/Notes
RUN and Boot Control (see text for usage guide)					
RUN	I	3V3[b]	Pull High	Leave open	Has internal 10k pull up
EMMC_DISABLE_N	I	3V3[b]	Pull High	Leave open	Has internal 10k pull up
EMMC_EN_N_1V8	O	1V8	Pull High	Leave open	Has internal 2k2 pull up
GPIO					
GPIO[27:0]	I/O	GPIO0-27_VDD	Pull or Hi-Z[e]	Leave open	GPIO Bank 0
GPIO[45:28]	I/O	GPIO28-45_VDD	Pull or Hi-Z[e]	Leave open	GPIO Bank 1
Primary SD Interface[d,e]					
SDX_CLK	O	SDX_VDD	Pull High	Leave open	Primary SD interface CLK
SDX_CMD	I/O	SDX_VDD	Pull High	Leave open	Primary SD interface CMD
SDX_Dx	I/O	SDX_VDD	Pull High	Leave open	Primary SD interface DATA
USB Interface					
USB_Dx	I/O	-	Z	Leave open	Serial interface
USB_OTGID	I	3V3		Tie to GND	OTG pin detect
HDMI Interface					
HDMI_SCL	I/O	3V3[b]	Z[f]	Leave open	DDC Clock (5.5V tolerant)
HDMI_SDA	I/O	3V3[b]	Z[f]	Leave open	DDC Data (5.5V tolerant)
HDMI_CEC	I/O	3V3	Z	Leave open	CEC (has internal 27k pull up)
HDMI_CLKx	O	-	Z	Leave open	HDMI serial clock
HDMI_Dx	O	-	Z	Leave open	HDMI serial data
HDMI_HPD_N_1V8	I	1V8	Pull High	Leave open	HDMI hotplug detect
CAM0 (CSI0) 2-lane Interface					
CAM0_Cx	I	-	Z	Leave open	Serial clock
CAM0_Dx	I	-	Z	Leave open	Serial data
CAM1 (CSI1) 4-lane Interface					
CAM1_Cx	I	-	Z	Leave open	Serial clock
CAM1_Dx	I	-	Z	Leave open	Serial data
DSI0 (Display 0) 2-lane Interface					
DSI0_Cx	O	-	Z	Leave open	Serial clock
DSI0_Dx	O	-	Z	Leave open	Serial data
DSI1 (Display 1) 4-lane Interface					
DSI1_Cx	O	-	Z	Leave open	Serial clock
DSI1_Dx	O	-	Z	Leave open	Serial data
TV Out					
TVDAC	O	-	Z	Leave open	Composite video DAC output
JTAG Interface					
TMS	I	3V3	Z	Leave open	Has internal 50k pull up
TRST_N	I	3V3	Z	Leave open	Has internal 50k pull up
TCK	I	3V3	Z	Leave open	Has internal 50k pull up
TDI	I	3V3	Z	Leave open	Has internal 50k pull up
TDO	O	3V3	O	Leave open	Has internal 50k pull up

[a] The PDN column indicates power-down state (when RUN pin LOW)
[b] Must be driven by an open-collector driver
[c] GPIO have software enabled pulls which keep state over power-down
[d] Only available on Lite variants
[e] The CM will always try to boot from this interface first
[f] Requires external pull-up resistor to 5V as per HDMI spec

Table 3: Pin Functions

6 Electrical Specification

Caution! Stresses above those listed in Table 4 may cause permanent damage to the device. This is a stress rating only; functional operation of the device under these or any other conditions above those listed in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Symbol	Parameter	Minimum	Maximum	Unit
VBAT	Core SMPS Supply	-0.5	6.0	V
3V3	3V3 Supply Voltage	-0.5	4.10	V
1V8	1V8 Supply Voltage	-0.5	2.10	V
VDAC	TV DAC Supply	-0.5	4.10	V
GPIO0-27_VDD	GPIO0-27 I/O Supply Voltage	-0.5	4.10	V
GPIO28-45_VDD	GPIO28-45 I/O Supply Voltage	-0.5	4.10	V
SDX_VDD	Primary SD/eMMC Supply Voltage	-0.5	4.10	V

Table 4: Absolute Maximum Ratings

DC Characteristics are defined in Table 5

3. SRF05

SRF05-HY - Ultra-Sonic Ranger

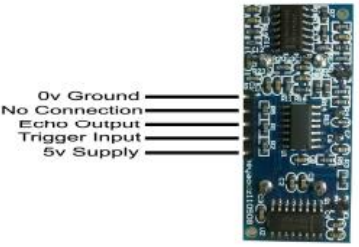
Technical Specification

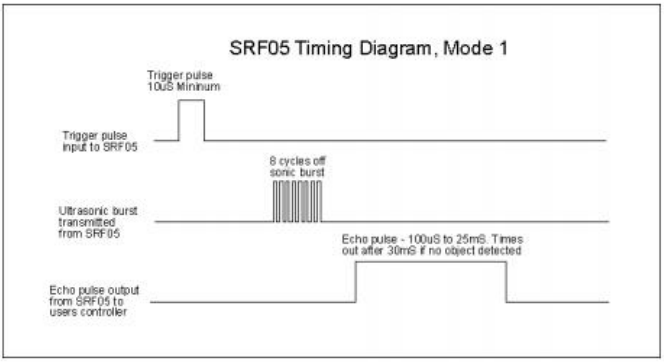
Introduction

The SRF05-HY is an evolutionary step from the SRF04-HY, and has been designed to increase flexibility, increase range, and to reduce costs still further. As such, the SRF05-HY is fully compatible with the SRF04-HY. Range is increased from 3 meters to 4 meters. A new operating mode (tying the mode pin to ground) allows the SRF05-HY to use a single pin for both trigger and echo, thereby saving valuable pins on your controller. When the mode pin is left unconnected, the SRF05-HY operates with separate trigger and echo pins, like the SRF04. The SRF05-HY includes a small delay before the echo pulse to give slower controllers such as the Basic Stamp and Picaxe time to execute their pulse in commands.

Mode 1 - SRF04-HYcompatible - Separate Trigger and Echo

This mode uses separate trigger and echo pins, and is the simplest mode to use. All code examples for the SRF04-HY will work for the SRF05-HY in this mode. To use this mode, just leave the mode pin unconnected - the SRF05-HY has an internal pull up resistor on this pin.





Mode 2 - Single pin for both Trigger and Echo
This mode uses a single pin for both Trigger and Echo signals, and is designed to save valuable pins on embedded controllers. To use this mode, connect the mode pin to the 0v Ground pin. The echo signal will appear on the same pin as the trigger signal. The SRF05-HY will not raise the echo line until 700uS after the end of the trigger signal. You have that long to turn the trigger pin around and make it an input and to have your pulse measuring code ready. The PULSIN command found on many popular controllers does this automatically.

Calculating the Distance
The SRF05-HY Timing diagrams are shown above for each mode. You only need to supply a short 10uS pulse to the trigger input to start the ranging. The SRF05-HY will send out an 8 cycle burst of ultrasound at 40khz and raise its echo line high (or trigger line in mode 2). It then listens for an echo, and as soon as it detects one it lowers the echo line again. The echo line is therefore a pulse whose width is proportional to the distance to the object. By timing the pulse it is possible to calculate the range in inches/centimeters or anything else. If nothing is detected then the SRF05-HY will lower its echo line anyway after about 30mS.

The SRF04-HYprovides an echo pulse proportional to distance. If the width of the pulse is measured in uS, then dividing by 58 will give you the distance in cm, or dividing by 148 will give the distance in inches. uS/58=cm or uS/148=inches.

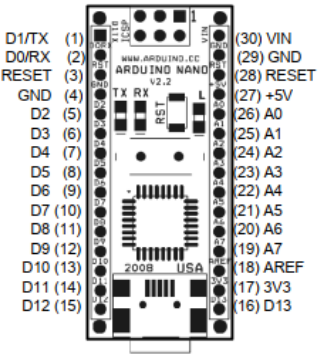
The SRF05-HY can be triggered as fast as every 50mS, or 20 times each second. You should wait 50ms before the next trigger, even if the SRF05-HY detects a close object and the echo pulse is shorter. This is to ensure the ultrasonic "beep" has faded away and will not cause a false echo on the next ranging.

The other set of 5 pins
The 5 pins marked "programming pins" are used once only during manufacture to program the Flash memory on the PIC16F630 chip. The PIC16F630's programming pins are also used for other functions on the SRF05-HY, so make sure you don't connect anything to these pins, or you will disrupt the modules operation.

Changing beam pattern and beam width
You can't! This is a question which crops up regularly, however there is no easy way to reduce or change the beam width that I'm aware of. The beam pattern of the SRF05-HY is conical with the width of the beam being a function of the surface area of the transducers and is fixed. The beam pattern of the transducers used on the SRF05-HY, taken from the manufacturers data sheet, is shown below.

4. Arduino Nano

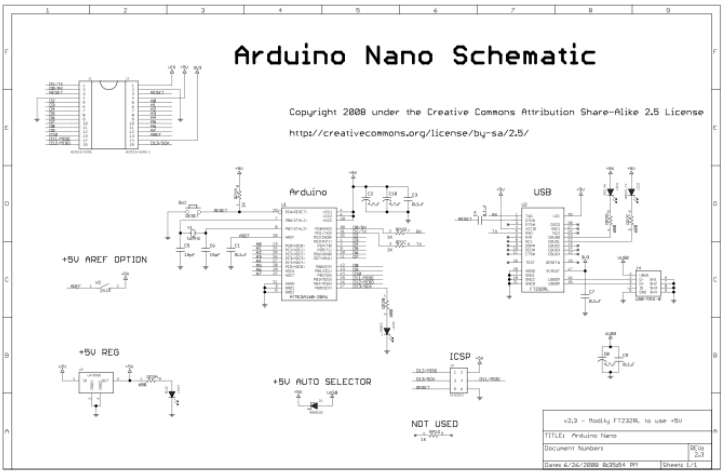
Arduino Nano Pin Layout



Pin No.	Name	Type	Description
1-2, 5-16	D0-D13	I/O	Digital input/output port 0 to 13
3, 28	RESET	Input	Reset (active low)
4, 29	GND	PWR	Supply ground
17	3V3	Output	+3.3V output (from FTDI)
18	AREF	Input	ADC reference
19-26	A7-A0	Input	Analog input channel 0 to 7
27	+5V	Output or Input	+5V output (from on-board regulator) or +5V (input from external power supply)
30	VIN	PWR	Supply voltage

Arduino Nano Bill of Material

Item Number	Qty.	Ref. Dest.	Description	Mfg. P/N	MFG	Vendor P/N	Vendor
1	5	C1,C3,C4,C7,C9	Capacitor, 0.1uF 50V 10% Ceramic X7R 0805	C0805C104K5RACTU	Kemet	80-C0805C104K5R	Mouser
2	3	C2,C8,C10	Capacitor, 4.7uF 10V 10% Tantalum Case A	T491A475K010AT	Kemet	80-T491A475K010	Mouser
3	2	C5,C6	Capacitor, 18pF 50V 5% Ceramic NOP/COG 0805	C0805C180J5GACTU	Kemet	80-C0805C180J5G	Mouser
4	1	D1	Diode, Schottky 0.5A 20V	MBR0520LT1G	ONSemi	863-MBR0520LT1G	Mouser
5	1	J1,J2	Headers, 36PS 1 Row	68000-136HLF	FCI	649-68000-136HLF	Mouser
6	1	J4	Connector, Mini-B Recept Rt. Angle	67503-1020	Molex	538-67503-1020	Mouser
7	1	J5	Headers, 72PS 2 Rows	67996-272HLF	FCI	649-67996-272HLF	Mouser
8	1	LD1	LED, Super Bright RED 100mcd 640nm 120degree 0805	APT2012SRCPRV	Kingbright	604-APT2012SRCPRV	Mouser
9	1	LD2	LED, Super Bright GREEN 50mcd 570nm 110degree 0805	APHCM2012CGCK-F01	Kingbright	604-APHCM2012CGCK	Mouser
10	1	LD3	LED, Super Bright ORANGE 160mcd 601nm 110degree 0805	APHCM2012SECK-F01	Kingbright	04-APHCM2012SECK	Mouser
11	1	LD4	LED, Super Bright BLUE 80mcd 470nm 110degree 0805	LTST-C170TBKT	Lite-On Inc	160-1579-1-ND	Digikey
12	1	R1	Resistor Pack, 1K +/-5% 62.5mW 4RES SMD	YC164-JR-071KL	Yageo	YC164J-1.0KCT-ND	Digikey
13	1	R2	Resistor Pack, 680 +/-5% 62.5mW 4RES SMD	YC164-JR-07680RL	Yageo	YC164J-680CT-ND	Digikey
14	1	SW1	Switch, Momentary Tact SPST 150gf 3.0x2.5mm	B3U-1000P	Omron	SW1020CT-ND	Digikey
15	1	U1	IC, Microcontroller RISC 16kB Flash, 0.5kB EEPROM, 23 I/O Pins	ATmega168-20AU	Atmel	556-ATMEGA168-20AU	Mouser
16	1	U2	IC, USB to SERIAL UART 28 Pins SSOP	FT232RL	FTDI	895-FT232RL	Mouser
17	1	U3	IC, Voltage regulator 5V, 500mA SOT-223	UA78M05CDCYRG3	TI	595-UA78M05CDCYRG3	Mouser
18	1	Y1	Cystal, 16MHz +/-20ppm HC-49/US Low Profile	ABL-16.000MHZ-B2	Abracon	815-ABL-16-B2	Mouser



5. Pi Camera

Hardware specification

	Camera Module v1	Camera Module v2	HQ Camera
Net price	$25	$25	$50
Size	Around 25 × 24 × 9 mm		38 x 38 x 18.4mm (excluding lens)
Weight	3g	3g	
Still resolution	5 Megapixels	8 Megapixels	12.3 Megapixels
Video modes	1080p30, 720p60 and 640 × 480p60/90	1080p30, 720p60 and 640 × 480p60/90	1080p30, 720p60 and 640 × 480p60/90
Linux integration	V4L2 driver available	V4L2 driver available	V4L2 driver available
C programming API	OpenMAX IL and others available	OpenMAX IL and others available	
Sensor	OmniVision OV5647	Sony IMX219	Sony IMX477
Sensor resolution	2592 × 1944 pixels	3280 × 2464 pixels	4056 x 3040 pixels
Sensor image area	3.76 × 2.74 mm	3.68 x 2.76 mm (4.6 mm diagonal)	6.287mm x 4.712 mm (7.9mm diagonal)
Pixel size	1.4 µm × 1.4 µm	1.12 µm x 1.12 µm	1.55 µm x 1.55 µm
Optical size	1/4"	1/4"	
Full-frame SLR lens equivalent	35 mm		
S/N ratio	36 dB		
Dynamic range	67 dB @ 8x gain		
Sensitivity	680 mV/lux-sec		
Dark current	16 mV/sec @ 60 C		
Well capacity	4.3 Ke-		
Fixed focus	1 m to infinity		N/A
Focal length	3.60 mm +/- 0.01	3.04 mm	Depends on lens
Horizontal field of view	53.50 +/- 0.13 degrees	62.2 degrees	Depends on lens
Vertical field of view	41.41 +/- 0.11 degrees	48.8 degrees	Depends on lens
Focal ratio (F-Stop)	2.9	2.0	Depends on lens

Hardware features

Available	Implemented
Chief ray angle correction	Yes
Global and rolling shutter	Rolling shutter
Automatic exposure control (AEC)	No - done by ISP instead
Automatic white balance (AWB)	No - done by ISP instead
Automatic black level calibration (ABLC)	No - done by ISP instead
Automatic 50/60 Hz luminance detection	No - done by ISP instead
Frame rate up to 120 fps	Max 90fps. Limitations on frame size for the higher frame rates (VGA only for above 47fps)
AEC/AGC 16-zone size/position /weight control	No - done by ISP instead
Mirror and flip	Yes
Cropping	No - done by ISP instead (except 1080p mode)
Lens correction	No - done by ISP instead
Defective pixel cancelling	No - done by ISP instead
10-bit RAW RGB data	Yes - format conversions available via GPU
Support for LED and flash strobe mode	LED flash
Support for internal and external frame synchronisation for frame exposure mode	No
Support for 2 × 2 binning for better SNR in low light conditions	Anything output res below 1296 x 976 will use the 2 x 2 binned mode
Support for horizontal and vertical sub-sampling	Yes, via binning and skipping
On-chip phase lock loop (PLL)	Yes
Standard serial SCCB interface	Yes
Digital video port (DVP) parallel output interface	No
MIPI interface (two lanes)	Yes
32 bytes of embedded one-time programmable (OTP) memory	No
Embedded 1.5V regulator for core power	Yes

Software features

Full camera software documentation can be found here.

Picture formats	JPEG (accelerated), JPEG + RAW, GIF, BMP, PNG, YUV420, RGB888
Video formats	raw h.264 (accelerated)
Effects	negative, solarise, posterize, whiteboard, blackboard, sketch, denoise, emboss, oilpaint, hatch, gpen, pastel, watercolour, film, blur, saturation
Exposure modes	auto, night, nightpreview, backlight, spotlight, sports, snow, beach, verylong, fixedfps, antishake, fireworks
Metering modes	average, spot, backlit, matrix
Automatic white balance modes	off, auto, sun, cloud, shade, tungsten, fluorescent, incandescent, flash, horizon
Triggers	Keypress, UNIX signal, timeout
Extra modes	demo, burst/timelapse, circular buffer, video with motion vectors, segmented video, live preview on 3D models

6. Hobbywing Ubec 5V 3A

User Manual of 3A UBEC

3 Amp Switch-Mode UBEC

1. Why do you need UBEC?

The UBEC is a switch-mode DC regulator separated from ESC (ESC---Electronic Speed Controller for brushless/brushed motor), it will take high-voltage (5.5V to 26V) power from your battery pack and convert it to a consistent safe voltage for your receiver, gyro and servos.

For traditional speed controller with a built-in BEC, it will very likely have only a limited ability to supply power to your receiver and servos without overheating. If you are using a high-voltage battery pack or have heavy servo load, you should consult the ESC specifications to determine what the stated recommendations or limitations are. In general, if you are using a 4 cells lithium battery pack, or more than a 12 cells Nickel based battery pack, you should consider using an UBEC because in such a case the built-in BEC of the ESC can support only 2 servos, which means it is not suitable for RC helicopter and big aircraft.

2. Specification:

2.1. **Output:** 5V/3A and 6V/3A switchable
2.2. **Ripple:** <50mVp-p(@2A/12V)
2.3. **Input:** 5.5V-26V (2 to 6 cells Lipo battery pack, 5 to 18 cells NiMH battery pack)
2.4. **Size:** 51mm*16.6mm*8.5mm (length*width*height)
2.5. **Weight:** 11.5g

3. Features:

3.1. The UBEC is an advanced switching regulator with over-current and over-heat protection function, and the maximum efficiency of the system is nearly 90%.
3.2. The small size and the light weight make it very convenient to use.
3.3. The comparison of linear BEC and switch-mode BEC: When using a lithium battery pack more than 3S, a switch-mode BEC has much higher efficiency than linear BEC.
- For a traditional linear BEC, For example, a 4S lithium battery pack has a typical voltage of 14.8V, in order to let BEC output 5V/1A, the current flow into the BEC is at least 1A, so the power on BEC is 14.8V* 1A=14.8W. But the useful output power is only 5V*1A=5W, so the efficiency of the linear mode BEC is just 5W/14.8W=33.8%, the redundant power 14.8W-5W=9.8W changes to heat, which makes the BEC very hot.
- For a switch-mode BEC in the above case, in order to let BEC output 5V/1A, the current flow into BEC is only 0.38A (actual test data), so the power on BEC is 14.8V* 0.38A =5.6W, and the efficiency of BEC is 5W/5.6W=89.3%.

3.4. Don't worry about the polarity of battery pack. If the polarity is not correct, the UBEC can't work, but it will not be destroyed. What you need to do is just swap the battery pack polarity.

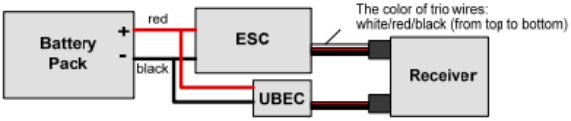
3.5. A shield covers almost all the electronic components on PCB, and a ferrite ring is attached with the output wires to decrease the electromagnetic interference.

4. How to use UBEC?

4.1. **Important hint: Switch mode UBEC may cause some electromagnetic interference to receiver, Please install the UBEC with a distance at least 5cm away from the receiver.**

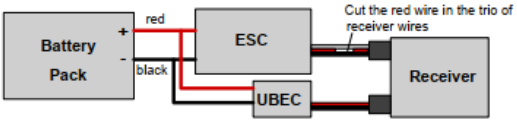
4.2. **When ESC HAS NOT built-in BEC**

No change is needed for the ESC, just connect the input cables of UBEC with the battery, and plug the output cable (connector) of the UBEC into one spare channel of the receiver.



4.3. **When ESC HAS built-in BEC**

You must disable the built-in BEC of the ESC, that is, you need to cut the red wire in the trio of Rx (receiver) wires. Simply use a pair of wire cutters to remove a short section of the red wire near the receiver connector, and insulate the cut wire with a bit of electrical tape.



Suggestion: You can use a sharp screwdriver to take the pin (with red wire) out from the BEC connector of the ESC, and then insulate it with a bit of electrical tape for further use, so you needn't cut the red wire by this method.

Take the pin with red wire out

7. GPS TS100

TS100

Radiolink MINI M8N GPS TS100, benefits from 15 years of professional wireless experiences of Radiolink engineers, exceed the limitation of IC sensitivity index from circuit schematic design to PCB placement.

50 centimeter position accuracy. Positioning 20 satellites in 6 seconds at open ground. Valley station-keeping ability. Appearance patent.

TS100 is compatible with flight controller Radiolink MINI PIX and PIXHAWK(the multicopter/helicopter firmware from V3.5.4, the fixed wing firmware from V3.8.0). Other flight controller also can use if do not need use compass of TS100.

Radiolink MINI M8N GPS TS100 Configuration

GPS decoder chip: Radiolink M8N GPS, with u-blox UBX-M8030(M8), 72-channel, MMIC BGA715L7 from Infineon, is much better than single GNSS 7N.

Concurrent reception of GPS/QZSS L1 C/A,GLONASS L10F, BeiDou B1, two GNSS working at the same time.

SBASL1 C/A: WAAS, EGNOS, MSAS

Geomagnetic: QMC5883L which with same technology as HMC5983 from Honeywell

Antenna : 2.5dbl high gain and selectivity ceramic antenna

Power amplify IC: MMIC BGA715L7 from Infineon

Double Filter: SAWF(Surface acoustic wave filter) form Murata

Parameter

1) Positional Accuracy: 50 centimeter precision when working with concurrent GNSS.
2) Velocity precision: 0.1m/s Max update rate: up to 10Hz
3) Max height: 50000m Max speed: 515m/s
4) Max acceleration: 4G
5) Sensitivity

Tracking & Nav.: -167dBm, Reacquisition: -163dBm, Cold start: -151dBm, Hot start: -159dBm.

6) Time to first fix: Cold start: 26s, Hot start:1s.
7) Connect ports

Power supply: voltage 5VDC+-5%, current 50-55mA

8) Ports

A. GPS UART interface, baud rate: 1.2K/4.8K/9.6K/19.2K/38.4K/57.6K/112.5K

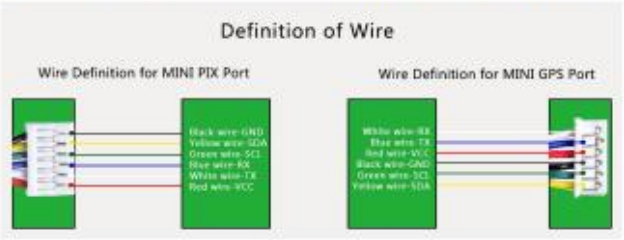
B. Geomagnetic I2C interface

2

Definition of Connector

Connect to flight controller: Black wire-GND, Yellow wire-SDA, Green wire-SCL, Blue wire-RX, White wire-TX, Red wire-VCC

GPS Mainboard : White wire-RX, Blue wire-TX, Red wire-VCC, Black wire-GND, Green wire-SCL, Yellow wire-SDA



TS100 Connect to Mini Pix

8. Emax BLHeli ESC 20A

EMAX User Instruction for BLHeli Series ESC

Thank you for purchasing EMAX ESC, please read this manual carefully before you use the ESC and strictly follow the instructions. EMAX accepts no liability for damage(s) or injuries incurred directly or indirectly from the use of this product, or modification of this product. Due to unforeseen changes or product upgrades in design, appearance, performance, the information contained in this manual is subject to change without notice.

A. Features

A1: Use authentic electronic components to ensure high quality and enhance the current endurance ability of the ESC.

A2: Based on BLHeli firmware, optimized for high performance with great linearity and much quicker throttle response.

A3: Special designed for multirotors, and compatible with fixed-wing aircrafts and helicopters.

A4: Multiple protection features including Low-voltage cut-off protection / over-heat protection / throttle signal loss protection.

A5: Throttle range can be configured and is fully compatible with all receivers, providing smooth, linear and precise throttle response.

A6: All parameters can be programmed via using a transmitter, including default settings.

B.Product specification

Item	Continuous Current	Burst current (10S)	Li-xx Battery (cell)	Dimention L*W*H(mm)	Weight (g) wires Included	BEC Mode	BEC Output	Programmable
EMAX BLHeli-6A	6A	8A	1-2	22×13×5.5	6	Linear	0.8A/5V	YES
EMAX BLHeli -12A	12A	15A	2-4	42×20×8	11	Linear	1A/5V	YES
EMAX BLHeli -20A	20A	25A	2-4	52×26×7	28	Linear	2A/5V	YES
EMAX BLHeli -25A	25A	30A	2-4	52×26×7	28	Linear	2A/5V	YES
EMAX BLHeli -30A	30A	40A	2-4	52×26×7	28	Linear	2A/5V	YES
EMAX BLHeli -30A-OPTO	30A	40A	2-6	67×26×10	25	----	----	YES
EMAX BLHeli -40A-UBEC	40A	50A	2-6	73×28×12	41	Switch	3A/5V	YES
EMAX BLHeli -50A-UBEC	50A	60A	2-6	73×28×12	41	Switch	3A/5V	YES
EMAX BLHeli -60A-UBEC	60A	80A	2-6	73×36×12	63	Switch	5A/5V	YES
EMAX BLHeli -80A-UBEC	80A	100A	2-6	86×38×12	81	Switch	5A/5V	YES

C. Instructions

C1.Normal startup procedures

Move throttle stick to the bottom position and then switch on transmitter→ Connect battery pack to ESC→ The long "beep" sound should be emitted , means the bottom point of throttle range has been detected→ Several "beep" tones should be emitted to present the amount of battery cells→ When self-test is finished, a "♪ 1 2 3" tune should be emitted→ Move throttle stick upwards to go flying.

C2.Throttle range setting procedures (when users change a transmitter, throttle range setting is recommended.)

Switch on the transmitter, move throttle stick to the top position→ Connect battery pack to ESC→ Two "beep" sounds should be emitted, means the top point of throttle range has been confirmed and saved→ Move throttle stick to the bottom position (within 2s), a long "beep" sound should be emitted , means the bottom point of throttle range has been detected→ Several "beep" tones should be emitted to present the amount of battery cells→ When self-test is finished, a "♪ 1 2 3" tune should be emitted, Move throttle stick upwards to go flying.

If the throttle stick is neither at the bottom position nor the top position after powered on, it will constantly make "beep" sounds.

D. Programmable parameters

D1. Brake Type: There are two options: **OFF**, ON. The default is OFF.

D2. Timing Mode: There are five options: Low: **0°**, Mid-low: **8°**, **Middle:15°**, Mid-high:23° and High:30°. The default is Middle: 15°. Low advance timing is recommended for high inductance and low KV motors. High advance timing is recommended for low inductance and high KV outrunner motors. For some high KV motors, if it shakes while rotating in high speed, the High timing mode is recommended.

D3. Start Force: There are 13 options: 0.031, 0.047, 0.063, 0.094, 0.125, 0.188, 0.25, 0.38, 0.50, 0.75, 1.00, 1.25, 1.50. The default is 0.75. Select the corresponding start force according to the load of motor.

D4. Curve Mode: There are 4 options: **OFF**, Low, Middle and High. The default is OFF.

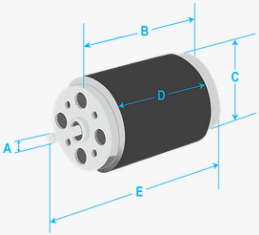
D5. Control Frequency : 2 options: **8KHz** and 22KHz. The default is 8KHz. This option is the drive frequency of the motors.

D6. Low-voltage Protection: 4 options: OFF, 2.8V/cell, **3.0V/cell**, 3.2V/cell. The default is 3.0V/cell. the system will automatically identify the battery cell. E.g. suppose there're 3 cells, if the voltage is lower than 9V, the system will work according to the current cutoff option.

9. LDPower MT2213 920kV Brushless Motor

PRODUCT DESCRIPTION | PRODUCT SPECIFICATIONS | VIDEOS | REVIEWS | COMMUNITY DISCUSSIONS | MANUALS/FILES

Kv(rpm/v)	920.00	Max Current (Motor) (A)	18.00
Resistance (mh)	0.00	Max Voltage (V)	14.00
Power (W)	302.40	Shaft A(mm)	6.00
Length B (mm)	27.00	Can Diameter C(mm)	28.00
Can Length D(mm)	14.00	Total Length E(mm)	41.00



BIODATA PENULIS

Dionisius Yose Deofili Abadi, lahir di Malang, 11 Oktober 1997. Penulis menyelesaikan pendidikan dasar di SDK Santa Maria II, Malang. Kemudian melanjutkan pendidikan tingkat menengah di SMPN 5 Malang dan SMAN 3 Malang. Pada tahun 2016, melanjutkan studi di Departemen Teknik Elektro, Fakultas Teknologi Elektro dan Informatika Cerdas, Intitut Teknologi Sepuluh Nopember Surabaya. Selama kuliah, penulis aktif sebagai peneliti pada tim Soeromiber (*Vertical Take-Off Landing*) yang tergabung dalam tim robot terbang Bayucaraka ITS, mengikuti beberapa kepanitiaan di kampus, dan sebagai asisten laboratorium Elektronika.