SISTEM PENDARATAN OTOMATIS *QUADCOPTER* DENGAN PENGOLAHAN CITRA MENGGUNAKAN METODE *DOUGLAS PEUCKER*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Teknik

Disusun oleh:
Cindy Lilian
NIM: 145150301111054

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

PENGESAHAN

SISTEM PENDARATAN OTOMATIS *QUADCOPTER* DENGAN PENGOLAHAN CITRA MENGGUNAKAN METODE *DOUGLAS PEUCKER*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Teknik

Disusun Oleh :
Cindy Lilian
NIM: 145150301111054

Skripsi ini telah diuji dan dinyatakan lulus pada
07 Juni 2018
Telah diperiksa dan disetujui oleh:

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

Mengetahui
Ketua Jurusan Teknik Informatika

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

PERNYATAAN ORISINALITAS

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

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

Malang, 07 Juni 2018

Cindy Lilian

NIM: 145150301111054

KATA PENGANTAR

Puji syukur kehadirat Allah SWT yang telah melimpahkan rahmat, taufik dan hidayah-Nya sehingga laporan skripsi yang berjudul "Sistem Pendaratan Otomatis *Quadcopter* Dengan Pengolahan Citra Menggunakan Metode *Douglas Peucker*" ini dapat terselesaikan. Laporan skripsi ini disusun dalam rangka memenuhi persyaratan untuk memperoleh gelar Sarjana Komputer di Fakultas Ilmu Komputer

Penulis menyadari bahwa penyusunan laporan skripsi ini tidak lepas dari bantuan berbagai pihak baik secara langsung maupun tidak langsung. Dalam kesempatan ini penulis ingin menyampaikan ucapan terima kasih atas bantuannya kepada semua pihak, sehingga penulis dapat menyelesaikan laporan ini dengan baik. Ucapan terima kasih tersebut khususnya kepada :

1. Allah yang Maha Esa yang selalu memberikan petunjuk dan hikmah dalam penulisan ini.
2. Orang Tua dan Keluarga atas nasehat, kasih sayang serta dukungan materil dan moril.
3. Gembong Edhi Setyawan, S.T, M.T dan Wijaya Kurniawan, S.T, M.T selaku dosen pembimbing yang telah memberikan dukungan, kritik dan saran yang membangun serta ilmu dalam membimbing penulis hingga pengerjaan skripsi ini selesai.
4. Bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D. selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya Malang.
5. Bapak Heru Nurwarsito, Ir., M.Kom. selaku Wakil Dekan I Bidang Akademik Fakultas Ilmu Komputer Universitas Brawijaya Malang.
6. Bapak Tri Astoto Kurniawan, S.T., M.T., Ph.D. selaku Ketua Jurusan Teknik Informatika.
7. Bapak Sabriansyah Rizqika Akbar, S.T, M.Eng. selaku Ketua Program Studi Teknik Komputer Universitas Brawijaya Malang.
8. Seluruh civitas akademika Informatika Universitas Brawijaya dan teman-teman Teknik Komputer Angkatan 2014 yang telah banyak memberi bantuan dan dukungan selama peneliti menempuh studi di Teknik Komputer Universitas Brawijaya dan selama penyelesaian skripsi ini.
9. Ayang Setiyo Putri, Achmad Baecuni, Dimas Angger, Sabita Wildani, Yusril Dewantara, Andyan Bina, Sabitha Wildani, Amroy Casro, Fajar Miftakhul selaku senior *quadcopter research* dan teman seperjuangan *quadcopter* Mesra, Enno, Muliya, Haqqi, Satria serta rekan-rekan lainnya yang turut memberikan motivasi dan hingga pengerjaan skripsi ini selesai.
10. Muhammad Yaqub atas dukungan dan kesediaan dalam meluangkan waktu untuk menjadi teman diskusi.

11. Dan orang-orang yang selalu mendukung serta mendoakan kelancaran proses skripsi ini yang tidak bisa disebutkan satu per satu atas semua doa dan dukungannya.

Dengan segala keterbatasan pengetahuan yang dimiliki, penulis sadar bahwa penulisan laporan skripsi ini masih jauh dari kesempurnaan. Sehingga penulis berharap agar laporan skripsi ini dapat bermanfaat dan merupakan salah satu informasi yang berguna bagi pembaca.

Malang, 07 Juni 2018

Penulis

Liliancindy916@gmail.com

ABSTRAK

Unmanned Aerial Vehicle (UAV) telah terbukti sebagai alat yang bermanfaat dalam berbagai bidang. UAV atau *quadcopter* dilengkapi dengan sejumlah sensor dan sistem pencitraan yang dapat melakukan berbagai hal berbasis citra. Hal ini membuktikan bahwa pemanfaatan *quadcopter* dapat dialokasikan untuk tujuan tertentu, misalnya tujuan mencapai mekanisme pendaratan. Pendaratan merupakan fase yang dapat menyebabkan kerusakan yang *fatal* pada *quadcopter* jika tidak mendarat pada tempatnya. Maka dari itu *quadcopter* dirancang agar dapat mendarat secara otomatis pada suatu objek dalam berbagai bentuk menggunakan algoritma *douglas-peucker*. Pertama melakukan konversi ruang warna RGB ke *grayscale* agar objek mudah terdeteksi lalu menghilangkan *noise* menggunakan *blur*. Kemudian mendeteksi tepian objek menggunakan *canny* dan dilanjutkan dengan pencarian nilai kontur yang sesuai agar mendapatkan tepi-tepi dari objek sehingga *quadcopter* bisa mengenali masing-masing bentuk objek dari tepian yang didapatkan serta dibantu dengan penghitungan sisi objek menggunakan metode *douglas-peucker.* Lalu ditampilkan pada sebuah *frame* yang dibagi menjadi 9 *grid* dan melakukan pergerakan otomatis sesuai posisi *grid* hingga pendaratan otomatis. Pengujian dilakukan dengan menguji keakuratan deteksi objek dengan pergerakan yang berbeda dengan kecepatan yang berbeda serta menguji ketepatan pendaratan dengan ketinggian berbeda. Hasil menunjukan bahwa, pada kecepatan 0,3 m/s didapatkan persentase ketepatan deteksi objek yang baik yaitu 100% dan didapatkan persentase ketepatan pendaratan terbaik yaitu 83,3% pada ketinggian 125cm dengan hasil rata-rata *delay* pendaratan otomatis senilai 3,42 detik.

Kata kunci: *Quadcopter*, Pendaratan Otomatis, Pengolahan Citra, *Douglas Peucker.*

ABSTRACT

Unmanned Aerial Vehicle (UAV) has proven to be a useful tool in many fields. UAV or quadcopter is equipped with a number of sensors and imaging systems that can perform various image-based things. This proves that quadcopter utilization can be allocated for a particular purpose, for example the purpose of reaching the landing mechanism. The landing is a phase that can cause fatal damage to the quadcopter if it does not land in its place. Thus the quadcopter is designed to automatically land on an object in various forms using the douglas-peucker algorithm. First convert the RGB color space to grayscale for easy to detect objects and eliminate noise using blur. Then it detects the edges of the object using canny and proceeds to search for the appropriate contour values to get the edges of the object so that the quadcopter can recognize each object shape from the edge obtained and assisted with the object counting using the douglas-peucker method. Then it will be displayed on a frame that is divided into 9 grids and performs automatic movement according to the grid position until perform autnomous landing. Testing is done by testing the accuracy of object detection with different movements of different speeds and testing the accuracy of landing with different heights. The result shows that, at the speed of 0.3 m / s, the best accuracy detection percentage percentage is 100% and the best landing accuracy percentage is 83,3% at 125cm with result of automatic landing delay average 3,42 second.

Keywords : Quadcopter, Automatic Landing, Image Processing, Douglas Peucker.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I PENDAHULUAN

1.1 Latar belakang

Unmanned Aerial Vehicle (UAV) atau pesawat tak berawak telah banyak terbukti sebagai alat yang bermanfaat bagi masyarakat saat ini. Penggunaan UAV dalam berbagai bidang antara lain pengawasan dan pengamatan dalam militer, serangan melaui udara dan laporan *survey* jika daerah tersebut dalam bahaya agar manusia dapat mengantisipasi dengan segera (Gadda, 2013). *Quadcopter* adalah salah satu jenis pesawat tanpa awak yang memiliki empat buah baling-baling (Hanafi, 2014). Pesawat tanpa awak berjenis *quadrotor* ini pada tahun terakhir telah banyak dipergunakan sebagai bahan dalam berbagai tema penelitian dan juga aplikasi. Seperti penelitian Micro – *quadcopter AR.Drone* yang kini telah tersedia secara komersial dan sudah bukan hal yang asing untuk dijadikan sebagai bahan penelitian. *AR.Drone* memiliki segelintir keuntungan yang dapat diperoleh seperti, ukurannya yang kecil, memiliki kestabilan yang baik serta dapat beroperasi baik itu didalam ruangan yang terbatas ataupun diluar ruangan. *AR.Drone* telah dilengkapi dengan sejumlah sensor dan sistem pencitraan/visualisasi yang memungkinkan melakukan berbagai hal berbasis citra. Contoh permasalahan visualisasi yang tengah populer dan sedang dikembangkan pada beberapa pesawat tanpa awak ini adalah sebuah pendeteksian target dengan pengolahan citra digital melalui kamera yang terintegrasi. Masalah ini bisa diselesaikan dengan metode yang beragam, antara lain dengan mencocokan pola atau bentuk (*shape/pattern matching*) (Dang et al, 2013).

Metode tersebut menunjukan bahwa telah terjadi kemajuan dalam bidang *engineering* yang memungkinkan untuk menerapkan metode-metode kecerdasan yang dapat ditanamkan untuk suatu tujuan tertentu. Tujuan yang ingin dicapai berupa sebuah misi, misalnya misi mekanisme pendaratan. Pendaratan otomatis merupakan salah satu misi yang biasa diterapkan pada pesawat tak berawak. Dengan adanya pendaratan otomatis, hal-hal yang tidak diinginkan selama mekanisme *landing* bisa diminimalisir (Hanafi, 2014).

Menurut Hamdani et al (2013), *quadcopter* secara harfiah dikendalikan oleh manusia dari jarak yang jauh dengan radio transmitter sehingga sering mengalami kesalahan terhadap pengendalian dalam mekanisme *landing*. Hal ini terjadi karena sebuah mekanisme *landing* merupakan salah satu mekanisme paling kritis dalam pengendalian *quadcopter* yang membutuhkan keakuratan untuk menghindari hal-hal yang tidak diinginkan seperti kerusakan yg fatal pada *quadcopter* saat mendarat bukan pada tempatnya. Alasan lain pendaratan merupakan fase paling krusial pada penerbangan *quadcopter* karena walaupun sudah ada metode yang dikembangkan yaitu dengan penambahan perangkat *GPS*(*Global Positioning System)* untuk mendukung sistem pendaratan secara otomatis, ternyata memilki kelemahan yang fatal. Ketika berada di daerah yang lingkungannya tidak dapat dijangkau oleh sinyal GPS atau memperoleh sinyal

yang minim, maka sudah bisa dipastikan pendaratan tidak bisa dilakukan pada set poin yang terdapat pada GPS. Sehingga perangkat GPS tidak dapat digunakan sebagai acuan untuk mendarat. Maka dari itu diperlukan adanya sebuah mekanisme yang dapat membantu dalam mengatasi faktor kesalahan selama *landing*. Beberapa penelitian yang telah berhasil melakukan sistem pendaratan yaitu oleh Hamdani pada tahun 2013, metode yang yang digunakan adalah *Behavior-Based Intelligent Fuzzy Control,* penelitian lainnya oleh Aprilian pada tahun 2017 yang menggunakan koordinat GPS untuk sistem pendaratan otomatis dan Gaol yang menggunakan *Linier Quadratic Regulator (LQR)* sebagai metode yang juga dapat melakukan pendaratan otomatis. Maka dari itu diperlukan suatu metode sebagai solusi untuk mengatasi masalah berbasis citra dalam mengenali objek landasan pada sistem pendaratan otomatis.

Algoritma *Douglas-Peucker* merupakan algoritma yang berfungsi untuk mengoptimalkan proses pengolahan citra. Algoritma ini sering digunakan pada saat penentuan jalur pada sebuah peta yang memiliki fitur *zoom*. Jika diperkecil, maka jalur yang akan digambar lebih kecil karena yang akan di ambil hanya titik-titik tertentu dan dianggap penting (Stefano, 2016).

Pada penelitian ini menerapkan sistem pendaratan otomatis berbasis pengolahan citra menggunakan beberapa metode sebagai pendukung pendeteksian objek landasan dalam bentuk deteksi tepi dengan berbagai bentuk objek seperti segitiga, kotak dan segilima agar tidak muncul suatu kerancuan selama berlangsungnya pendeteksian bentuk objek. Pertama ruang warna RGB dari objek akan di konversi terlebih dahulu ke *Grayscale*. Kemudian mendeteksi tepian objek menggunakan *canny* dan dilanjutkan dengan pencarian nilai kontur yang sesuai agar mendapatkan tepi-tepi dari objek sehingga *quadcopter* bisa mengenali masing-masing bentuk objek dari tepian yang didapatkan serta dibantu dengan penghitungan sisi objek menggunakan metode *douglas peucker.* Lalu ditampilkan pada sebuah *frame* yang dibagi menjadi 9 *grid* dan melakukan pergerakan otomatis sesuai posisi *grid* ketika mendeteksi objek hingga mencapai pendaratan otomatis.

1.2 Rumusan masalah

Berdasarkan latar belakang yang telah diuraikan diatas, maka rumusan masalah dari penelitian ini adalah sebagai berikut.

1. Bagaimana implementasi metode *douglas peucker* pada *quadcopter* untuk dapat mendeteksi objek landasan dengan tepat?
2. Berapa kecepatan dan ketinggian yang digunakan untuk dapat mendeteksi dan mendarat pada objek segitiga, kotak dan segilima?
3. Bagaimana *quadcopter* dapat mencari titik tengah pada objek untuk melakukan pendaratan secara otomatis?
4. Bagaimana hasil ketepatan dan waktu yang dibutuhkan untuk mendarat secara otomatis?

1.3 Tujuan

Sesuai dengan beberapa rumusan masalah yang telah dijabarkan, maka penelitian ini dibuat dengan tujuan adalah sebgai berikut.

1. Untuk mengetahui implementasi metode *douglas peucker* pada *quadcopter* agar dapat mendeteksi objek landasan dengan tepat.
2. Untuk mengetahui kecepatan dan ketinggian yang digunakan untuk dapat mendeteksi dan mendarat pada objek segitiga, kotak dan segilima.
3. Untuk mengetahui cara *quadcopter* dapat mencari titik tengah pada objek untuk melakukan pendaratan secara otomatis.
4. Untuk mengetahui ketepatan dan waktu yang dibutuhkan untuk melakukan pendaratan secara otomatis.

1.4 Manfaat

Manfaat dari penelitian ini adalah:

1. Memberikan pengetahuan lebih luas terhadap pembaca terkait penelitian *quacopter* mengenai metode pengolahan citra seperti *canny edge detection*, RGB ke *Grayscale*, *Douglas Peucker* dan penerapan *quadcopter* menggunakan *Robot Operating System* (*ROS*).
2. Untuk memungkinkan *quadcopter* melakukan pendaratan otomatis tepat pada objek yang digunakan sebagai acuan landasan untuk mendarat berdasarkan pengolahan citra digital.
3. Sebagai bentuk penelitian terkait sistem pendaratan otomatis yang nantinya bisa dikembangkan lebih lanjut oleh peneliti lain dalam pemanfaatan *quadcopter* khususnya pada bidang pengolahan citra yang lebih baik.

1.5 Batasan masalah

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

1. Sistem diuji coba pada *indoor*.
2. Menggunakan *Parrot AR Drone* tipe 2.0 sebagai pengujian.
3. *Quadcopter* terbang dengan tujuan mencapai mode *autonomus landing* dengan pengolahan citra digital dalam bentuk pendeteksian objek landasan yang berupa segitiga, kotak dan segilima.
4. Ketinggian yang diperlukan untuk terbang yaitu pada rentang 125 cm hingga 200 cm untuk melakukan pendaratan.
5. Hanya mendeteksi tiga bentuk landasan yaitu segitiga, kotak dan segilima.
6. Pergerakan hanya dalam satu jalur untuk melakukan pendaratan.

1.6 Sistematika pembahasan

Sistematika penulisan dalam skripsi ini sebagai berikut:

BAB 1 Pendahuluan

Pada bab ini menguraikan tentang Latar Belakang yaitu asal-usul dari penelitian skripsi dilakukan, Rumusan Masalah yang merupakan masalah yang terdapat pada latar belakang, Batasan Masalah, Tujuan yang merupakan tujuan dari penilitian bedasarkan rumusan masalah yang telah didapatkan, Manfaat yang berupa manfaat dari penilitian yang akan dilakukan, dan Sistematika Penulisan yaitu merupakan sistematika pada penulisan penilitan skripsi.

BAB 2 Tinjauan Pustaka dan Dasar Teori

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

BAB 3 Metode Penelitian

Menguraikan tentang metode dan langkah kerja yang terdiri dari studi literatur yang telah dipelajari dan digunakan pada penilitian, analisis kebutuhan sistem yaitu melakukan analasis kebutuhan yang dibutuhkan pada penilitian, perancangan sistem yaitu melakukan perancangan pada penelitian sistem baik perangkat keras maupun perangkat lunak, implementasi dan analisis serta pengambilan kesimpulan bedasarkan hasil pengujian.

BAB 4 Analisis Kebutuhan

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

BAB 5 Perancangan dan Implementasi

Pada bab ini menguraikan proses perancangan yang terdapat pada sistem perangkat keras dan perangkat lunak dan melakukan implementasi berdasarkan pada dasar teori-teori yang telah dipelajari sesuai analisis dan perancangan sistem pada sistem yang telah dilakukan.

BAB 6 Pengujian dan Analisis

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

BAB 7 Penutup

Bab ini berisi kesimpulan atas penelitian yang telah dilakukan, dan memberikan saran untuk pengembangan sistem lebih lanjut serta berguna untuk mengetahui apakah terdapat kesalahan pada implementasi dari hasil penelitian.

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab ini berisikan penjelasan dasar teori dari berbagai sumber pustaka (literatur) untuk menunjuang penelitian yang diusulkan. Kajian yang terdapat pada bab ini juga membahas penelitian yang sudah ada sebelumnya yang memiliki keterkaitan dengan penelitian yang akan dilakukan oleh penulis.

2.1 Tinjauan Pustaka

Tinjauan pustaka memuat tentang keterkaitan penelitian sebelumnya yang telah ada tentang penerapan metode pada *quadcopter* dalam melakukan pendaratan yang memiliki keterkaitan dengan penelitian yang akan dilakukan saat ini. Dengan refrensi penelitian dari sebelumnya maka akan berguna untuk membenahi kekurangan yang ada dan bisa menghasilkan penyempurnaan terhadap penelitian yang akan dibuat.

Penelitian yang dilakukan oleh Ginkle, dkk (2015) adalah penelitian yang bertujuan untuk mendaratkan *AR drone* secara otomatis dengan *image processing* yang mendeteksi sebuah pola yang berbentuk huruf H sebagai objek landasan. Pada penelitian tersebut menggunakan beberapa metode yaitu konversi RGB *(Red Green Blue)* ke HSV *(Hue Saturation Value)* untuk mendeteksi objek landasan dan pendeteksian kontur pada objek menggunakan metode *thresholding* lalu jika objek berbentuk huruf H terdeteksi, maka akan ditandai dengan batasan pinggiran dengan bentuk lingkaran sehingga memudahkan untuk mencari titik tengah sebagai landasan pendaratan. Namun pada hasil pengujiannya, memiliki masalah dalam melakukan deteksi objek sebagai penanda pendaratan *quadcopter* dan juga ada kesalahan dalam teknis pada bagian *hardware* yang membuat *quadcopter* tidak dapat terbang dengan stabil.

Penelitian lainnya yang dilakukan oleh Dang, dkk (2013) adalah penelitian berbasis pengolahan citra yang merupakan penelitian serupa untuk sistem pendaratan otomatis. Dengan sistem *tracking object* yang didukung dengan hanya satu metode pengolahan citra saja, yaitu segmentasi warna yang berupa konversi dari RGB ke HSV dan masih belum mendapatkan hasil yang maksimal karena ada kendala saat *tracking object* berlangsung. Ada *noise* yang muncul pada *background* ketika objek landasan terdeteksi, hal ini membuat *quadcopter* dalam situasi ini ketika lokasi objek berada di luar *area* pendeteksian, *quadcopter* menjadi melayang-layang di sekitar posisi di mana objek terdeteksi. Ada juga masalah dibagian *hardware* yang membuat *quadcopter* tidak bisa bekerja sesuai dengan tujuan yang ingin dicapai.

Maka, pada penelitian kali ini juga menerapkan hal yang sama dalam hal mendarat secara otomatis berbasis pengolahan citra. Namun, pada pendeteksian objek menggunakan beberapa metode yaitu *canny edge detection* sebagai pendukung pendeteksian tepi objek dan *douglas peucker* yang mengklasifikasikan bentuk dengan menghitung masing-masing tepi pada objek sehingga dapat dikenali saat *quadcopter* akan melakukan pendaratan.

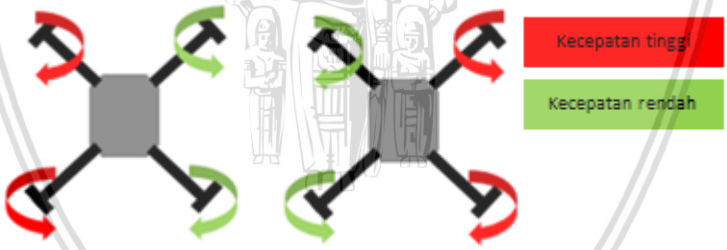
2.2 Dasar Teori

2.2.1 *Quadcopter*

Quadcopter adalah pesawat tanpa awak yang memiliki 4 buah rotor dan baling-baling untuk keempat sisinya dan dapat bergerak ke berbagai arah. Motor-motor yang dimiliki oleh *quadcopter* dapat diatur kecepatannya agar dapat mengubah pergerakan. Kelebihan yang dimiliki oleh *quadcopter* seperti dapat melakukan pergerakan ke segala arah memudahkan dalam melakukan *tracking* pada suatu objek melalui kamera yang dimiliki oleh *quadcopter (*Gaol, 2017).

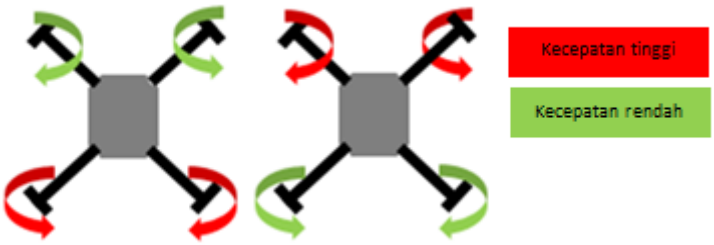
Pesawat tanpa awak ini merupakan terobosan teknologi yang memiiki pengaruh yang signifikan dalam berbagai keperluan. Segala misi yang memiliki resiko tinggi bagi pilot dapat digantikan dengan menggunakan *quadcopter* khususnya dibidang militer. Dalam melakukan penerbangan *quadcopter* memiliki beberapa gerakan dasar yang dipengaruhi oleh kecepatan dari masing-masing rotornya, gerakan tersebut yaitu *roll, pitch* dan *yaw* dan *throttle*.

1. Gerakan *Roll* memungkinkan pergerakan manuver ke kiri atau kanan dengan cara meningkatkan kecepatan dari rotor bagian kiri jika ingin menghasilkan gerakan ke kanan dan sebaliknya jika ingin menggerakan ke arah kiri, maka kecepatan putar dari baling-baling sebelah kanan ditingkatkan.



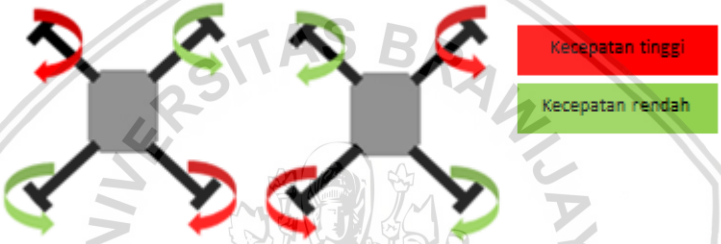
Gambar 2.1 Gerakan Roll *Quadcopter*

2. Gerakan *Pitch* memiliki prinsip yang serupa dengan pergerakan roll. Pada pergerakan *roll* menghasilkan gerakan manuver ke kiri dan kanan dari pengaturan kecepatan baling-baling pada bagian kiri dan kanan. Sedangkan pada *pitch* , kecepatan baling-baling yang diatur adalah pada bagian depan dan belakang. Sehingga akan menghasilkan pergerakan manuver maju ke depan jika kecepatan dari baling-baling belakang ditingkatkan dan sebaliknya, akan dihasilkan pergerakan mundur ke belakang jika kecepatan rotor depan meningkat.



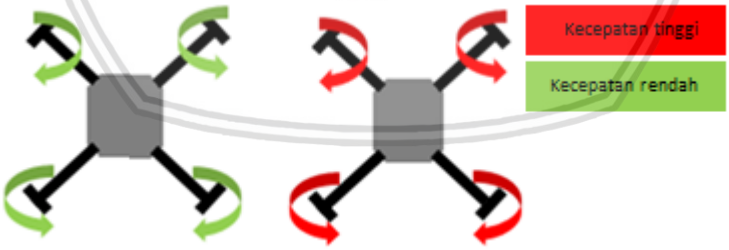
Gambar 2.2 Gerakan pitch *quadcopter*

3. *Yaw* adalah gerakan rotasi kanan dan kiri yang diperoleh dari pengurangan kecepatan putar secara bersamaan pada 2 baling-baling dengan posisi bersilangan dan meningkatkan kecepatan putar 2 baling-baling yang memiliki arah putar yang berlawanan dengan kedua baling-baling sebelumnya.



Gambar 2.3 Gerakan *yaw quadcopter*

4. Gerakan *throttle* memiliki pergerakan sepanjanng sumbu z yang merupakan pengaruh dari kecepatan rotor yang berputar secara bersamaan. *Quadcopter* dapat melakukan akselerasi naik dan turun untuk mengatur ketinggian sesuai dengan kecepatan rotor.



Gambar 2.4 Gerakan throttle

2.2.2 Pengolahan Citra Digital

Pengolahan citra digital merupakan teknik manipulasi citra secara digital yang khususnya menggunakan komputer menjadi citra lain yang sesuai untuk digunakan dalam aplikasi tertentu. Operasi pengolahan citra digital umumnya dilakukan dengan tujuan memperbaiki kualitas suatu gambar sehingga dapat dengan mudah dikenali oleh mata manusia dan mengolah informasi yang ada pada suatu gambar untuk kebutuhan identifikasi objek secara otomatis (Murinto,

2009). Berbagai aplikasi pengolahan citra telah banyak di implementasikan untuk berbagai kepentingan baik itu dalam bidang kesehatan maupun dalam bidang robotika. Agar robot dapat melihat objek atau lingkungan sekitarnya diperlukan kamera sebagai indra penglihatan yang berfungsi untuk menangkap gambar yang diproses menjadi citra digital yang dapat diproses melalui komputer.

Pada penelitian ini dirancang sebuah sistem pendaratan pada *quadcopter* yang menangkap objek melalui sensor kamera, lalu dapat melakukan proses pengenalan suatu objek sesuai dengan bentuk dan dapat melakukan pergerakan secara otomatis ketika mendeteksi objek saat akan mendarat pada landasan yang diinginkan. Peneliti akan melakukan pendeteksian bentuk dengan jarak yang telah ditentukan untuk mekanisme pendaratan sekaligus menganalisis akurasi pendaratan menggunakan pengolahan citra dengan kecepatan terbaik yang diperoleh pada pengujian sebelumnya dan ketinggian yang berbeda-beda.

2.2.3 *Color Space*

2.2.3.1 *RGB*

Dalam lingkup komputerisasi, ruang warna RGB telah sering digunakan untuk menampilkan berbagai informasi pada layar monitor karena tidak membutuhkan proses transformasi. Alasan ini juga yang menyebabkan penerapan ruang warna RGB semakin banyak dimanfaatkan sebagai warna dasar untuk beragam aplikasi (Swedia & Cahyani, 2015).

Model dari ruang warna RGB merupakan model warna yang didasari oleh terangnya cahaya primer yang terdiri dari *red, green, blue.* Jika tidak ada cahaya yang masuk pada suatu ruangan, maka telah dipastikan bahwa ruangan tersebut gelap. Hal itu disebabkan karena tidak terdapat signal gelombang dari cahaya yang terserap oleh mata. Dalam situasi ini, ruang warna RGB bernilai (0,0,0). Apalbila di ruangan tersebut akan ditambahkan cahaya berwarna merah, maka lingkup pada ruangan akan berubah menjadi warna merah misalnya nilai RGB tersebut (255,0,0), dan seluruh benda yang berada pada ruangan hanya akan terlihat berwarna merah begitu juga sebaliknya jika diganti dengan warna biru atau hijau (Swedia, 2010).

2.2.3.2 RGB *ke Grayscale*

Pada *image processing* proses awal yang biasanya dilakukan adalah mengkonversi suatu citra yang berwarna ke dalam bentuk citra *grayscale,* hal ini bertujuan untuk mengubah model citra tersebut menjadi sederhana. Ada tiga jenis layer yang terdapat pada citra berwarna yaitu R-layer, G-layer, dan B-layer. Sehingga jika akan melanjutkan ke proses-proses berikutnya harus memperhatikan layer-layer tersebut (Shanti, 2011).

Tiga layer yang telah disebutkan merupakan warna dasar dari citra yang digunakan sebagai parameter untuk pengubahan ke citra *grayscale*. Untuk mengubah suatu gambar menjadi citra *grayscale* dibutuhkan pengambilan seluruh *pixel* pada citra warna dari gambar. Selanjutnya informasi dari warna

dasarnya yaitu *red*, *green*, *blue* akan dilakukan perhitungan untuk mendapatkan nilai *luminous*. Setelah nilai *luminous* dari warna diperoleh, maka gambar yang berwarna akan berubah menjadi citra *grayscale* karena *pixel*-nya telah diset dengan nilai *luminous* yang telah didapatkan sebelumnya. Dan nilai-nilai untuk mendapatkan citra *grayscale* didefinisikan seperti persamaan pada 2.1 (Shanti, 2011).

$$L = 0.2989\ R\ + 0.5870\ G\ + 0.1140\ B \tag{2.1}$$

2.2.4 Deteksi Tepi (*Canny Edge Detection*)

Tepi atau sisi dari sebuah obyek adalah daerah di mana terdapat perubahan intensitas warna yang cukup tinggi. Proses deteksi tepi *(edge detection)* akan melakukan konversi terhadap daerah ini menjadi dua macam nilai yaitu intensitas warna rendah atau tinggi, contoh bernilai nol atau satu. Deteksi tepi akan menghasilkan nilai tinggi apabila ditemukan tepi dan nilai rendah jika sebaliknya (Lusiana, 2013). Pelacakan suatu tepi merupakan proses pengolahan citra yang bertujuan agar mendapatkan perubahan intensitas yang bervariasi pada citra (Prasetyo, 2011; Gonzales, 2002). Pada penelitian ini digunakan algoritma *canny* untuk mendeteksi tepian objek berbentuk segitiga, kotak, dan segilima.

Deteksi tepi *Canny* ditemukan oleh Marr dan Hildreth yang meneliti pemodelan persepsi visual manusia. Ada beberapa kriteria pendeteksi tepian paling optimum yang dapat dipenuhi oleh *canny* , seperti kemampuanya dalam mendeteksi dengan baik. Kemampuan yang dimaksud adalah untuk meletakkan dan menandai semua tepi yang ada sesuai dengan pemilihan parameter-parameter yang dilakukan. Sekaligus juga memberikan fleksibilitas yang sangat tinggi dalam hal menentukan ketebalan tepi sesuai yang diinginkan (Winarno, 2011).

Dalam pendeteksian tepi menggunakan *canny edge detection* terdapat beberapa proses yang terjadi yaitu sebagai berikut.

1. Penghalusan gambar untuk menghilangkan *noise* yang terdapat pada gambar objek pada penelitian ini menggunakan *averaging* dan *Gaussian blur* sebagai teknik yang mengatasi masalah tersebut. *Averaging* berfungsi untuk mendapatkan rata-rata dari semua *pixel* pada gambar pada area kernel dan mengganti element pusatnya untuk menciptakan tampilan yang halus. Ukuran kernel yang digunakan yaitu 3x3 seperti pada persamaan 2.2.

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.2}$$

Pada *Filter Gaussian* digunakan sebuah matriks ukuran 5x5 untuk mengatasi *noise* yang ada pada gambar objek landasan *quadcopter* sehingga tepian didapatkan dengan mudah. Persamaan untuk kernel *gaussian* yang digunakan adalah sebagai berikut.

$$H_{ij} = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{(i-(k+1))^2+(j-(k+1))^2}{2\pi\sigma^2}\right); 1 \leq\ i,j \leq (2k+1) \tag{2.3}$$

Berikut ini adalah contoh matriks 5x5 untuk *Filter Gaussian* yang digunakan untuk menghilangkan *noise* yang berada pada tepian dengan nilai $\sigma = 1.4$.

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \quad (2.4)$$

2. Setelah *noise* dihilangkan, maka langkah selanjutnya mencari nilai intensistas *gradient* pada gambar untuk mendapatkan hasil kekuatan pada setiap tepi yang diperoleh dari *gradient* vertikal dan horizontal. Pada tahap ini digunakan persamaan Sobel untuk mencari masing-masing *gradient*.

$$Gx = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad (2.5)$$

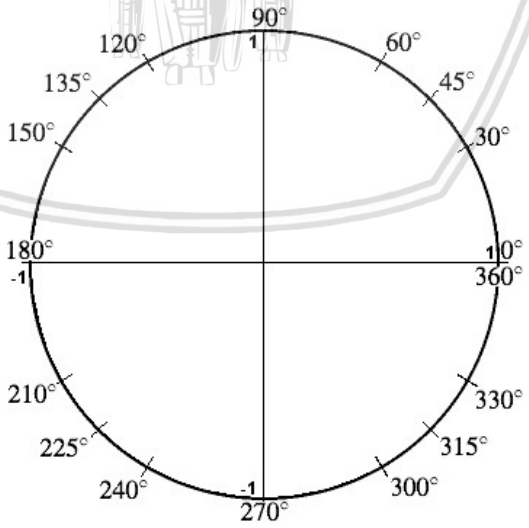
$$Gy = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \quad (2.6)$$

$$G = \sqrt{Gx^2 + Gy^2} \quad (2.7)$$

3. Agar pendeteksian tepi lebih fleksibel maka dilakukan pencarian direksi untuk mendapatkan *angle* pada tepian yang berfungsi untuk menemukan arah tepian selanjutnya dengan menggunakan rumus sebagai berikut.

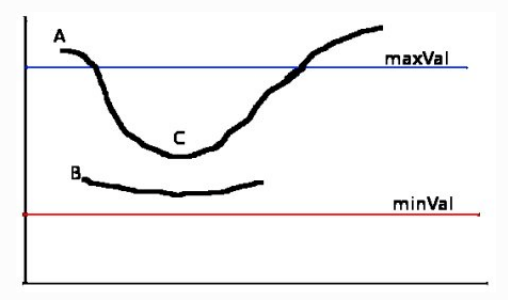
$$\theta = \arctan\left(\frac{Gy}{Gx}\right) \quad (2.8)$$

Aturan sudut untuk direksi tepian pada *canny edge detection* dapat dilihat pada Gambar 2.5.



Gambar 2.5 Arah tepi berdasarkan nilai sudut pada *canny*

4. Tahap akhir yaitu *Hysteresis Thresholding* yang menggunakan dua *threshold* yaitu *lower* dan *upper*. Berikut ini adalah beberapa kondisi untuk mendapatkan nilai tepian pada tahap akhir *canny edge detection* seperti yang terlihat pada Gambar 2.6.



Gambar 2.6 *Hysteresis Thresholding*

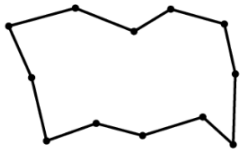
- Jika *gradient* pada *pixel* lebih besar daripada *upper threshold*, maka *pixel* tersebut merupakan tepian seperti yang terlihat pada garis A.
- Jika *gradient* pada *pixel* diantara kedua nilai *threshold*, maka *pixel* tersebut akan diterima jika berhubungan dengan *pixel* yang melebihi dari nilai *upper threshold* seperti yang terlihat pada garis C dan jika nilai *pixel* tidak terhubung dengan *pixel* yang melebihi dari *upper threshold* maka akan dibuang dari gambar seperti yang terlihat pada garis B.
- Jika *gradient* pada *pixel* lebih kecil daripada *lower threshold*, maka *pixel* tersebut dibuang dari gambar atau menjadi hitam.

2.2.5 *Douglas Peucker*

Algoritma *Douglas-Peucker* merupakan algoritma yang berfungsi untuk mengoptimalkan proses pengolahan citra. Algoritma ini sering digunakan pada saat penentuan jalur pada sebuah peta yang memiliki fitur *zoom*. Jika diperkecil, maka jalur yang akan digambar lebih kecil karena yang akan di ambil hanya titik-titik tertentu dan dianggap penting (Stefano, 2016).

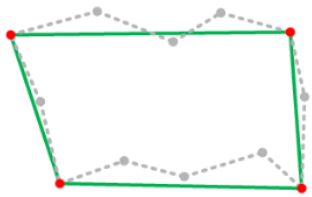
Titik-titik tersebut akan disimpan sebagai perwakilan dari jalur pada gambar. Berdasarkan uraian dari penjelasan tersebut, algoritma *douglas peucker* memilki proses awal dengan menyimpan titik pertama dan titik terakhir pada suatu jalur. Selanjutnya akan dilakukan penghitungan pada setiap titik dalam daftar perhitungan jarak yang tegak lurus mulai dari titik awal sampai titik tujuan (Stefano, 2016).

Seperti yang terlihat pada Gambar 2.7 merupakan tampilan awal ketika belum mengalami proses dengan metode *douglas peucker*. Pada gambar tersebut terdapat sejumlah titik point yang banyak dan membentuk pola yang rancu karena objek belum mencapai bentuk yang sesuai.



Gambar 2.7 Titik objek yang masih memiliki bentuk yang tidak beraturan

Setelah sejumlah titik point telah mengalami proses melalui *douglas peucker*, maka jumlah titik-titik tersebut akan berkurang dan sejumlah garis yang merupakan jarak dari tiap titik akan membentuk pola yang dapat dikenali sesuai pada Gambar 2.8.



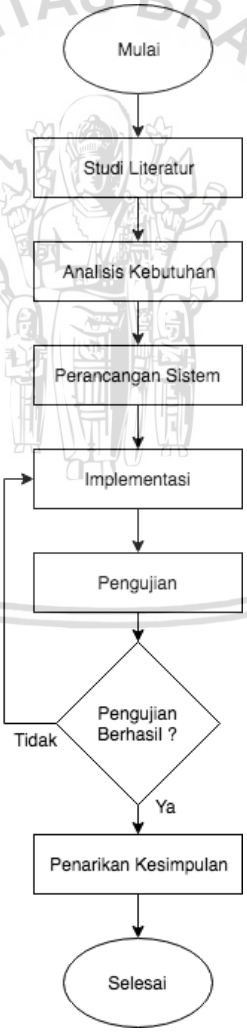
Gambar 2.8 Titik objek yang telah dilakukan pencarian algoritma *douglas peucker*

Dengan adanya penghitungan menggunakan algoritma *douglas peucker* bentuk objek dari landasan *quadcopter* dapat dikenali dan memungkinkan untuk dilakukannya pendaratan secara otomatis.

BAB 3 METODOLOGI

3.1 Metode Penelitian

Sebelum melaksanakan penelitian, maka akan ditentukan terlebih dahulu metode yang digunakan pada sistem. Hal ini akan menunjang dalam pelaksanaan penelitian. Metode yang digunakan pada penelitian ini akan disesuaikan dengan tujuan yang ingin dicapai dari penelitian. Pertama dilakukan pengumpulan berbagai dasar teori yang terkait dengan penelitian ini dan dijadikan sebagai studi literatur. Kemudian dilakukan analisis kebutuhan yang menjelaskan apa saja yang menjadi kebutuhan dalam sistem. Setelah itu dilakukan perancangan dan implementasi sistem yang didalamnya akan menjelaskan bagaimana sistem ini dirancang dan bentuk implementasinya serta akan dilakukan tahap pengujian dengan dua hasil yaitu jika berhasil maka dapat dilakukan penarikan kesimpulan dan jika pengujian tidak berhasil maka akan kembali ke tahap implementasi seperti yang terlihat pada Gambar 3.1.



Gambar 3.1 Alur Metodologi Penelitian

3.2 Studi Literatur

Pada bagian ini akan dijelaskan beberapa dasar teori yang terkait dan mendukung penelitian Sistem Pendaratan Otomatis *Quadcopter* Dengan Pengolahan Citra Menggunakan Metode *Douglas Peucker.* Dasar teori yang digunakan sebagai acuan dalam bahan studi adalah sebagai berikut.

1. RGB ke *Grayscale*
2. *Canny Edge Detection*
3. *Douglas Peucker*

3.3 Analisis Kebutuhan

Pada tahap ini akan dijelaskan berbagai kebutuhan yang diperlukan dalam membangun sistem. Kebutuhan yang ikut terlibat yaitu kebutuhan pengguna, kebutuhan sistem, serta kebutuhan fungsional dan non-fungsional. Kebutuhan pengguna nantinya akan menjelaskan tentang apa saja yang diperlukan agar dapat menghubungkan antara pengguna dengan sistem. Selanjutnya pada kebutuhan sistem akan dibahas perangkat yang dibutuhkan untuk membangun sistem yang terdiri atas dua jenis yaitu perangkat keras dan perangkat lunak. Kebutuhan perangkat keras akan membahas perangkat keras apa saja yang diperlukan secara spesifik, sedangkan pada kebutuhan perangkat lunak akan dijelaskan apa saja perangkat lunak yang diperlukan agar dapat mendukung performa sistem.

Kemudian pada kebutuhan fungsional akan dijelaskan bagaimana kesesuaian *input* yang diberikan terhadap kinerja sistem. Kebutuhan yang dimaksud adalah *quadcopter* dapat bergerak dan mendarat secara otomatis ketika mendeteksi objek landasan berwarna hitam dengan bentuk segitiga, kotak dan segilima .

Selanjutnya pada kebutuhan non-fungsional akan dibahas tentang kebutuhan pengguna, lingkungan operasi, asumsi dan ketergantungan, serta batasan perancangan dan implementasi.

3.4 Perancangan Sistem Dan Implementasi

Tahapan pada bab ini bertujuan untuk merancang sistem agar kebutuhan fungsional dapat terpenuhi. Perancangan pada sistem ini akan dibagi menjadi perancangan komunikasi sistem, perancangan deteksi objek dan perancangan pergerakan *quadcopter*.

Kemudian setelah tahap perancangan sistem selesai, akan dilanjutkan ke bagian proses realisai sistem dalam bentuk implementasi. Implementasi yang akan dilakukan berdasarkan perancangan sistem yang telah dibuat sebelumnya dengan beberapa jenis tahapan. Tahap awal yaitu implementasi komunikasi yang bertujuan untuk mengkoneksikan antara perangkat keras dengan ROS agar program dapat dijalankan. *Source code* yang dibuat berfungsi agar *quadcopter* dapat mengenali objek landasan berbentuk segitiga, kotak dan segilima sehingga dapat mendarat secara otomatis ketika objek telah terdeteksi. Program tersebut

dapat dijalankan secara langsung pada *quadcopter* untuk menguji kinerja *quadcopter* berdasarkan *source code* yang telah dibuat.

3.5 Pengujian dan Analisis

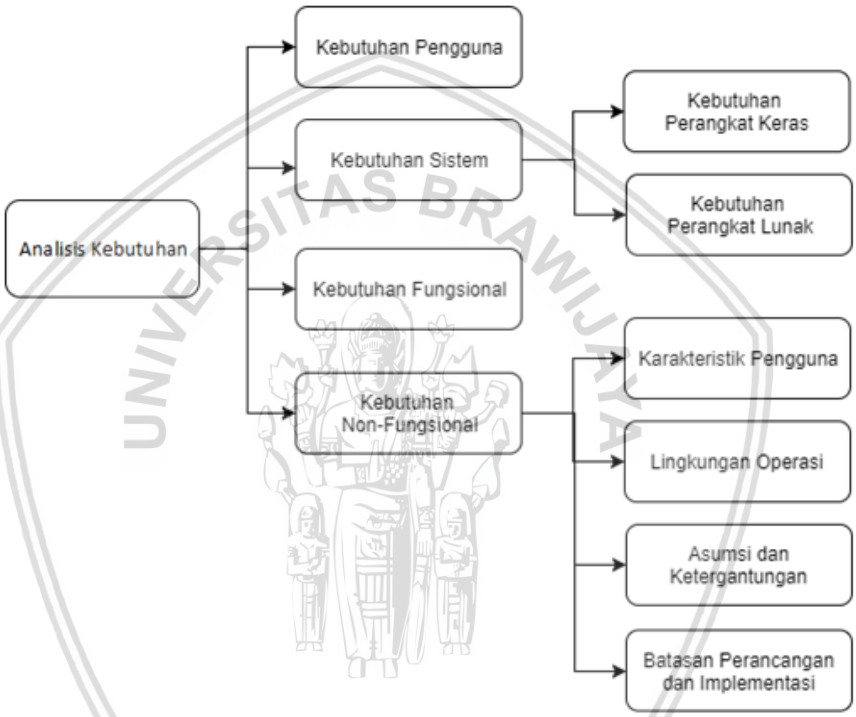
Tujuan dari pengujian dan analisis adalah mengetahui kesesuaian dari kinerja sistem secara keseluruhan dan juga performanya dengan perancangan yang telah dibuat. Hal yang akan dilakukan sebagai sekenario dari pengujian yaitu pengujian ketinggian, pengujian ketepatan dari gerakan *quadcopter*, pengujian *delay* sistem dan pengujian ketepatan *landing*.

3.6 Kesimpulan dan Saran

Pengambilan kesimpulan dan saran dilakukan setelah semua tahapan analisis kebutuhan, perancangan, implementasi, dan pengujian sistem selesai dilakukan. Kesimpulan diambil dari hasil pengujian dan analisis terhadap sistem yang dibangun. Tahap terakhir dari penulisan adalah saran yang dimaksudkan untuk memperbaiki kesalaham-kesalahan yang terjadi dan untuk menyempurnakan penulisan serta memberikan pertimbangan atas pengembangan sistem lebih lanjut.

BAB 4 ANALISIS KEBUTUHAN

Analisis kebutuhan bertujuan untuk mengetahui apa saja kebutuhan yang diperlukan sistem pada penelitian ini. Pada bagian analisis kebutuhan akan dijabarkan empat macam kebutuhan sistem, diantaranya perangkat keras, perangkat lunak, kebutuhan fungsional dan kebutuhan non-fungsional. Kemudian dari kebutuhan non-fungsional tersebut akan dijabarkan lagi menjadi empat karakteristik yang terdiri dari karakteristik pengguna, lingkungan operasi, asumsi dan ketergantungan, serta batasan perancangan dan implementasi seperti yang terlihat pada Gambar 4.1.



Gambar 4.1 Diagram Analisis Kebutuhan

4.1 Kebutuhan Pengguna

Kebutuhan pengguna merupakan suatu kebutuhan yang harus ada agar memudahkan pengguna dalam melakukan monitoring pada sistem dengan optimal. Kebutuhan antarmuka bagi pengguna pada sistem yakni dengan menggunakan *window* yang terdapat pada linux yang berfungsi menampilkan informasi yang telah diproses oleh sistem. Informasi yang ditampilkan dari sistem berupa objek dari bentuk segitiga, kotak dan segilima, serta respon dari pergerakan otomatis *quadcopter* ketika mendeteksi objek. Setiap objek memiliki ukuran yang berbeda-beda diantaranya, objek segitiga dengan tinggi 18 cm, alas 18 cm, sisi miring 25 cm, objek kotak dengan ukuran sisinya 18 cm dan objek segilima dengan ukuran 9 cm untuk semua sisinya.

4.2 Kebutuhan Sistem

Pada kebutuhan sistem, terdapat dua macam tahap analisis yang terdiri dari kebutuhan perangkat keras dan kebutuhan perangkat lunak.

4.2.1 Kebutuhan Perangkat Keras

Kebutuhan perangkat keras yang dibutuhkan oleh sistem ini adalah:

1. *Parrot AR.Drone 2.0*

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

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

   Spesifikasi yang terdapat pada *Parrot AR.Drone 2.0* ditunjukkan pada Tabel 4.1.

Tabel 4.1 Spesifikasi *Parrot Ar.Drone 2.0*

No	Parameter	Nilai
1.	Dimensi	517x517 milimeter dengan *blade protector*, 451x451 tanpa *blade protector*
2.	Kamera	HD 720 *pixel*(Video),JPEG(*Photo*)
3.	*Processor*	ARM Cortex 1GHz 32 bit, 1 GB RAM
4.	Sistem Operasi	Linux
5.	Video DSP	800 MHz
6.	Aplikasi	AR Freeflight 2.0 (Android)
7.	Konektivitas	*Wi-fi* (Jarak maksimal 50 meter)
8.	Baterai	LiPo (1.100 mAh / 11,1 volt)
9.	Durasi Baterai	12 menit

Sumber: Parrot (2011)

Berdasarkan spesifikasi pada Tabel 4.1 maka peneliti memilih *Parrot AR Drone* 2.0 untuk digunakan pada penelitian yang diajukan karena memiliki reolusi kamera yaitu 720 *pixel* yang sudah memenuhi standarisasi yang dibutuhkan oleh sistem pada penelitian ini dan bersifat *open source*.

4.2.2 Kebutuhan Perangkat Lunak

Kebutuhan perangkat lunak yang diperlukan dalam pembuatan sistem ini adalah sebagai berikut.

1. Sistem Operasi *Linux Ubuntu* versi 14.04
   Ubuntu adalah sistem operasi yang bersifat *open source* dan merupakan salah satu dari distribusi linux yang berbasis debian. Ubuntu biasanya digunakan berbagai kepentingan seperti penggunaan pribadi, namun ubuntu juga sangat populer untuk digunakan dalam *network servers*. Pada penilitian ini akan menggunakan *Ubuntu* versi 14.04 yang merupakan versi paling stabil dalam penggunaan yang akan dibutuhkan karena dapat terhubung dengan ROS tanpa kendala.
2. OpenCV
   OpenCV *(Open Source Computer Vision)* adalah sebuah pengolahan citra yang bersifat *open source* dan merupakan *machine learning library* yang didukung berbagai fungsi pemrograman untuk pengolahan citra secara *real-time*. Selain itu, OpenCV dapat digunakan secara gratis untuk penggunaan akademis maupun komersial dengan menggunakan lisensi produk BSD. OpenCV dapat digunakan dengan berbagai bahasa pemrograman seperti *C++, C, Python* dan *Java* (*Android*) dan mendukung sistem operasi *Windows, Linux, Android, iOS* dan *Mac OS*.
3. *ROS*
   ROS (*Robot Operating Sytem*) merupakan *framework* yang berfungsi untuk membuat berbagai macam program robot, menhubungkan pengguna dengan *quadcopter* dan bersifat fleksibel serta mudah untuk dioperasikan. Di dalam ROS terdapat berbagai jenis *library*, *driver*, *tool* yang memiliki tujuan untuk menyelesaikan semacam tugas pembuatan robot yang sistemnya kompleks dalam *platform* robot. Selain itu, ROS adalah *framework* yang juga bersifat *opensource* dan mendukung pengembangan kompleks, tetapi sistem masih berbasis modular dalam komputasi terdistribusi. Hal yang penting dalam kinerja pada *framework* ditulis dengan menggunakan bahasa pemrograman *C++.*
4. *C++*
   C++ adalah bahasa pemrograman yang bersifat *object oriented programming* yang dapat menyelesaikan berbagai masalah. Dalam menyelesaikan masalah, bahasa pemrograman *C++* memiliki beberapa langkah. Langkah awal yang dilakukan yaitu dengan menginisialisasi *class-class* yang akan dibuat dan merupakan abstraksi dari *object-object* fisik. Pada class tersebut terdapat kondisis *object,* kemampuan dan anggota-anggotanya dari *object*-nya.
5. *CVBridge*
   CvBridge merupakan *library* yang berperan dalam menghubungkan *Robot Operating System* dengan OpenCV.

4.3 Kebutuhan Fungsional

Pada kebutuhan fungsional akan diuraikan bagaimana sistem yang dibuat akan mendapatkan *output* yang sesuai dengan keinginan. Adapun bagian dari kebutuhan fungsional pada sistem yang harus terpenuhi adalah sebagai berikut.

1. Sistem dapat mendeteksi objek segitiga, kotak dan segilima dengan kecepatan dan ketinggian yang berbeda-beda.
2. Sistem dapat menampilkan pendeteksian objek segitiga, kotak dan segilima hingga fase mendarat secara otomatis.

4.4 Kebutuhan Non-Fungsional

Bab ini akan dibahas tentang apa saja yang akan menjadi batasan dalam kebutuhan non-fungsional terhadap kebutuhan pada perancangan sistem. Yang termasuk dalam lingkup kebutuhan non-fungsional dari sistem pada penelitian ini adalah sebagai berikut.

4.4.1 Karakteristik Pengguna

Karakteristik pengguna adalah diperuntukan untuk pengendali *quadcopter*. Dengan adanya sistem seperti ini diharapkan akan mempermudah dalam melakukan pendaratan karena adanya mode otomatis yang dirancang sehingga kesalahan selama *landing* dapat diminimalkan untuk mencegah hal-hal yang tidak diinginkan.

4.4.2 Lingkungan Operasi

Persyaratan lingkungan operasi untuk menjalankan sistem ini adalah sebagai berikut.

1. Objek yang digunakan berbentuk segitiga, kotak dan segilima dengan warna hitam.
2. Membutuhkan *area* dengan luas minimal 10x10 meter agar *quadcopter* bisa terbang dengan optimal.

4.4.3 Asumsi dan Ketergantungan

1. *Quadcopter* dapat mengenali bentuk ketika objek telah terdeteksi dan mencari titik tengahnya sekaligus melakukan pendaratan secara otomatis.
2. Sistem akan berfungsi jika komputer telah bisa terhubung dengan *Wi-fi* yang dipancarkan *AR.Drone quadcoper.*
3. Proses pengiriman data dari kamera *quadcopter* ke komputer hanya akan berhasil jika sistem telah berjalan
4. Sistem hanya dapat dijalankan pada komputer yang telah memiliki *ROS* dan OpenCV.

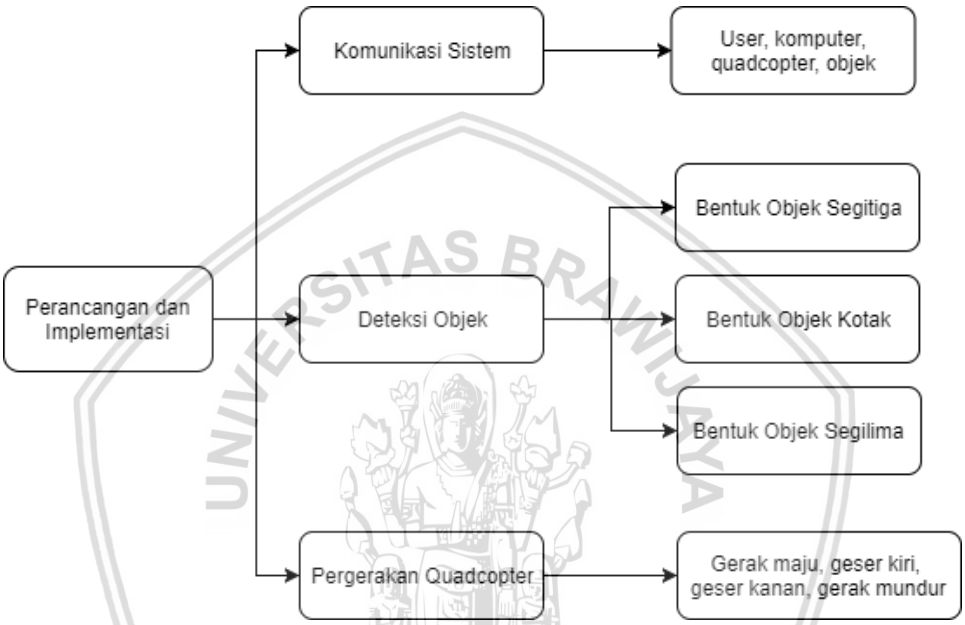
4.4.4 Batasan Perancangan dan Implementasi

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

1. Gerakan *quadcoper* hanya satu jalur sesuai dengan peletakan objek yang ditentukan.
2. Gerakan pada *quadcopter* yaitu gerakan *take off, landing, hover,* maju, mundur, geser kanan dan geser kiri.
3. *Quadcopter* bisa mendeteksi objek dan mendarat secara otomatis setelah perintah *take off* dan *input* navigasi dijalankan.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

Bab ini akan menjelaskan tentang proses perancangan dan implementasi dari sistem pendaratan otomatis menggunakan *quadcopter*. Dalam perancangan dan implementasi pada penelitian ini akan dilakukan beberapa tahapan seperti yang terdapat pada Gambar 5.1. Pertama dilakukan perancangan komunikasi pada sistem. Komunkasi pada sistem tersebut terdiri dari empat buah komponen yaitu *user*, komputer, *quadcopter*, dan objek berbentuk segitiga, kotak dan segilima.



Gambar 5.1 Tahapan Perancangan dan Implementasi

Setelah itu dilakukan perancangan terhadap pergerakan *quadcopter* dengan tujuan melakukan navigasi untuk dapat mencari objek landasan dalam satu jalur. Gerakan yang bisa dilakukan *quadcopter* ada 6 yaitu geser kiri, geser kanan, maju, mundur, *hover* dan *landing*. Tahap akhir yaitu perancangan dan implementasi deteksi objek agar pendaratan otomatis dapat dilakukan. Mode pendaratan otomatis yang dirancang hanya akan tercapai saat objek landasan terdeteksi. Untuk pendeteksian objek dibagi menjadi tiga jenis bentuk yaitu segitiga, kotak dan segilima.

Setelah tahap perancangan dan implementasi telah selesai dilakukan, maka akan menghasilkan hubungan yang akan membangun proses dari sistem secara keseluruhan. Hubungan tersebut sesuai dengan yang tertera pada Gambar 5.1.

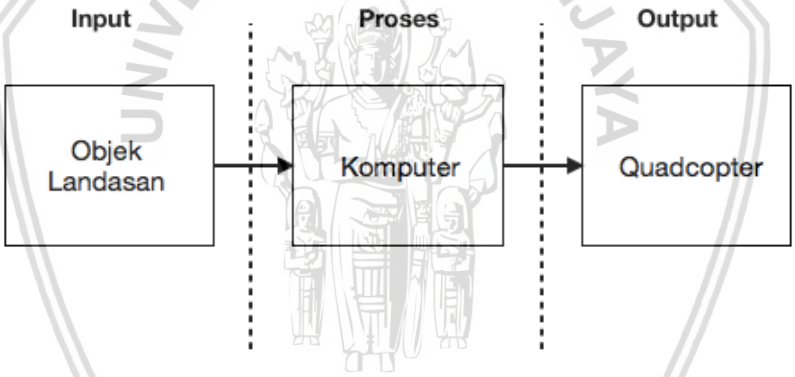
Langkah awal yang akan dilakukan berupa inisialisasi koneksi dari *quadcopter.* Selanjutnya akan dilakukan pembacaan sensor kamera *quadcopter*. Kemudian berlanjut ke proses pengubahan warna RGB ke *grayscale* untuk menjadikan objek berwarna abu-abu. Kemudian akan dilakukan proses *blur* pada objek untuk mengurangi *noise* dan berlanjut ke tahap deteksi tepi dengan menggunakan metode *canny edge detection*. Setelah tepian didapatkan maka

akan dilakukan proses pencarian nilai kontur agar tepi-tepi terdeteksi dengan optimal. Langkah berikutnya yaitu proses pengenalan bentuk objek melalui penghitungan setiap sisi menggunakan *douglass peucker*. Jika *quadcopter* belum dapat mendeteksi objek, maka akan kembali ke proses awal pengolahan citra hingga dapat mendeteksi bentuk objek landasan. Saat objek sudah terdeteksi, maka *quadcopter* akan melakukan pergerakan tanpa dikontrol manual untuk mencari titik tengah dari objek yang telah terdeteksi. Jika posisi *quadcopter* telah berhasil mencapai titik tengah objek, maka akan melakukan pendaratan secara otomatis.

5.1 Komunikasi Sistem

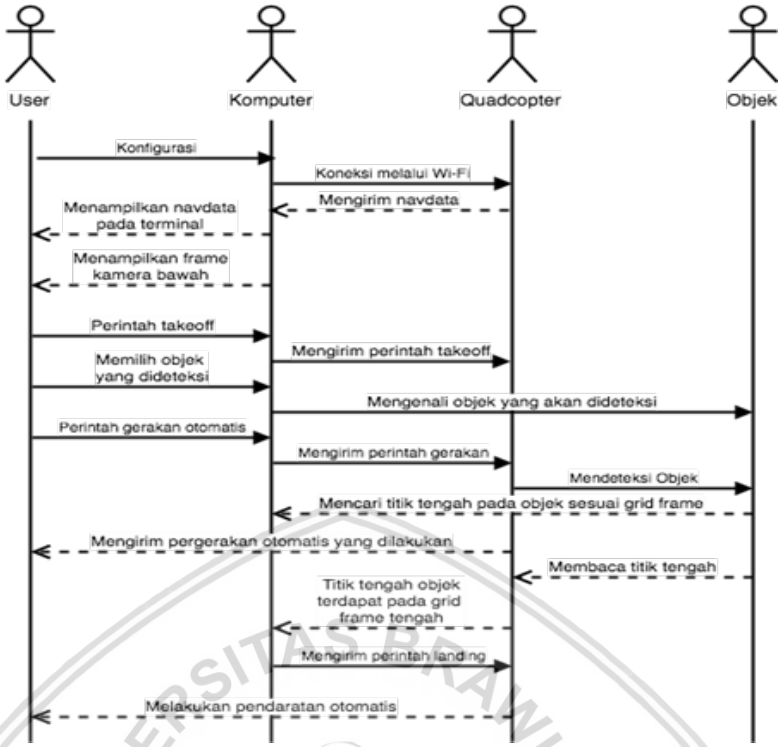
5.1.1 Perancangan Komunikasi Sistem

Pada perancangan sistem ini, untuk memudahkan pemahaman terhadap perancangan sistem secara keseluruhan maka dapat dijelaskan dalam bentuk diagram blok sistem. Diagram blok sistem pada penelitian ini akan menjelaskan prinsip kerja pendaratan secara otomatis pada *quadcopter* dalam mendeteksi objek yang menjadi target landasan seperti yang terlihat pada Gambar 5.2.



Gambar 5.2 Diagram blok komunikasi sistem

Pada perancangan ini, keseluruhan data yang diproses dijelaskan seperti pada Gambar 5.3. Pertama *user* mengkoneksikan komputer dengan *quadcopter* melalui *Wi-fi*. Kemudian melakukan konfigurasi pada komputer dengan menjalankan program yang telah dibuat. Selanjutnya *quadcopter* akan mengirim ke komputer berupa *navdata* yang terdiri atas data kamera dan juga data navigasi. Data yang telah dikirimkan akan ditampilkan pada layar komputer yang dapat dilihat langsung oleh *user*. Setelah itu *user* memberikan *input* dari komputer berupa perintah *take off* dan *quadcopter* akan mulai terbang kemudian bergerak dalam satu jalur dengan memberikan *input* untuk arah navigasi dengan tujuan mencari objek landasan. Selanjutnya *quadcopter* akan mendeteksi bentuk objek landasan tersebut dengan kamera bawah. Setelah data kamera berhasil didapatkan *quadcopter* akan mengirimkan *navdata* lagi ke komputer dan berikutnya *user* bisa melihat kinerja *quadcopter* yang mencapai mode otomatis dari mulai bergerak , mendeteksi landasan hingga mendarat.



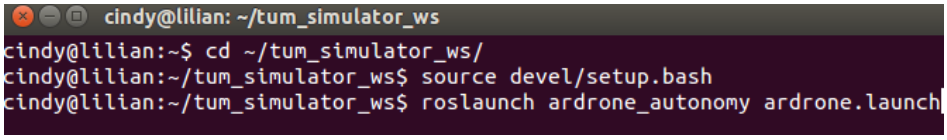
Gambar 5.3 Alur diagram pada komunikasi sistem

5.1.2 Implementasi Komunikasi Sistem

Bagian ini membahas bagaimana alur dari bentuk implementasi komunikasi sistem pada penelitian yang dilakukan. Untuk Implementasi komunikasi sistem ini ditunjukkan pada Gambar 5.4 sebagai langkah awal untuk mengkoneksikan komputer dengan *Wi-fi quadcopter* yang bernama "ArDrone_2".

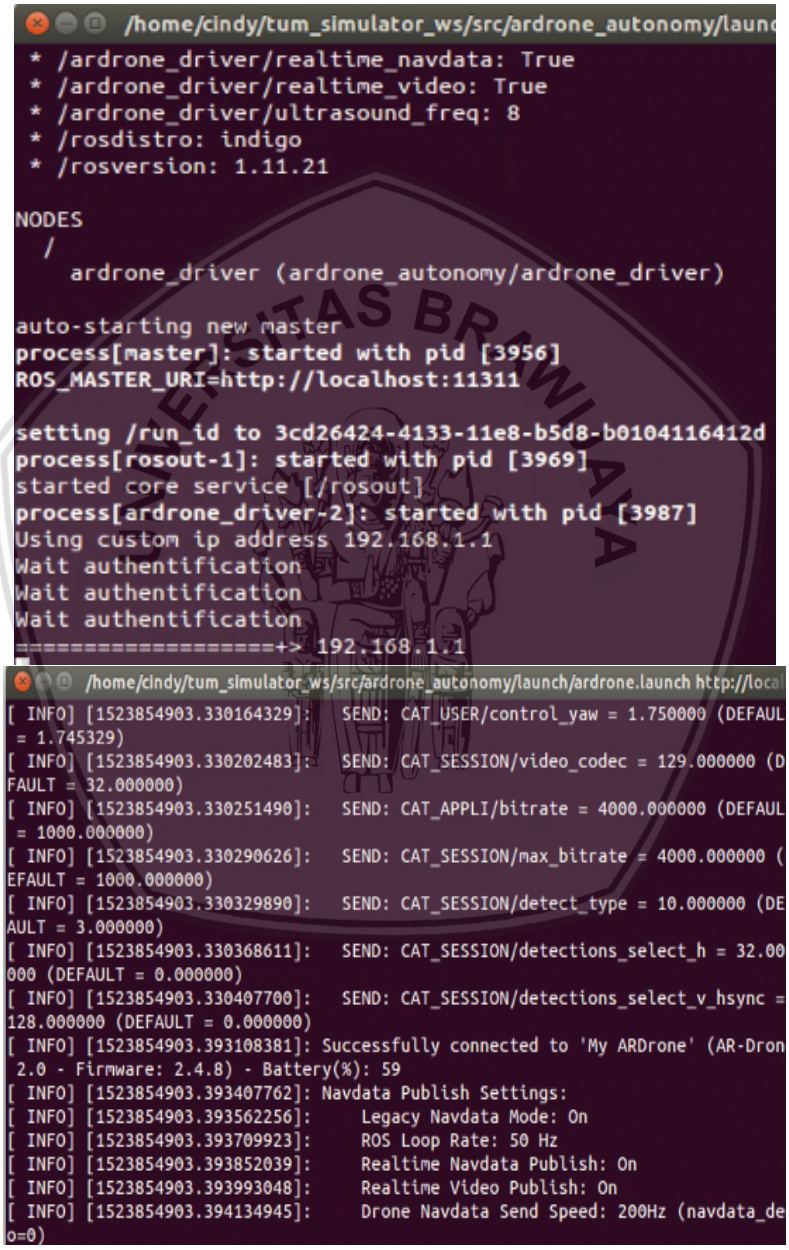
Gambar 5.4 Menghubungkan komputer dengan *Wi-fi quadcopter*

Kemudian buka terminal dan ketikan *syntax* seperti pada Gambar 5.5 yang berfungsi untuk menghubungkan IP *quadcopter* dan juga pengiriman *navdata* dari *quadcopter* ke komputer agar dapat diakses oleh *user*.



Gambar 5.5 Menghubungkan komputer dengan IP *quadcopter*

Tunggu beberapa saat hingga muncul pesan "connected" pada terminal sebagai tanda bahwa komputer telah terhubung dengan IP yang dimilki oleh *quadcopter* sesuai pada Gambar 5.6.



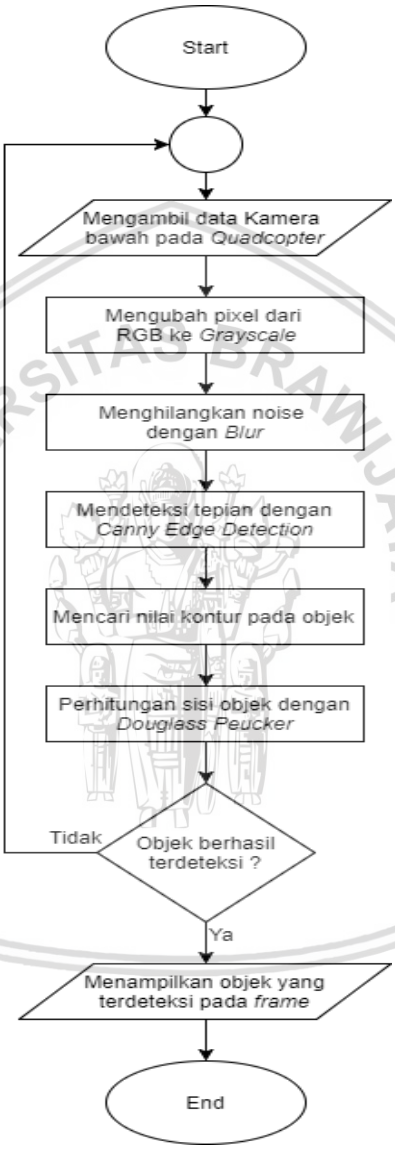
Gambar 5.6 Komputer telah berhasil terhubung dengan *quadcopter*

Jika komputer telah terkoneksi dengan IP *quadcopter ,* maka program akan dijalankan dan *quadcopter* akan bergerak sampai mendeteksi bentuk objek landasan menggunakan sensor dari kamera bawah.

5.2 Deteksi Objek

5.2.1 Perancangan Deteksi Objek

Dalam perancangan deteksi objek, yang akan diperlukan dan menjadi *input* dari sistem ini adalah data dari kamera pada *quadcopter*. Data tersebut diolah dengan menggunakan ROS dan OpenCV. Setelah data telah didapatkan, langkah berikutnya dilakukan beberapa proses seperti yang terdapat pada Gambar 5.7.



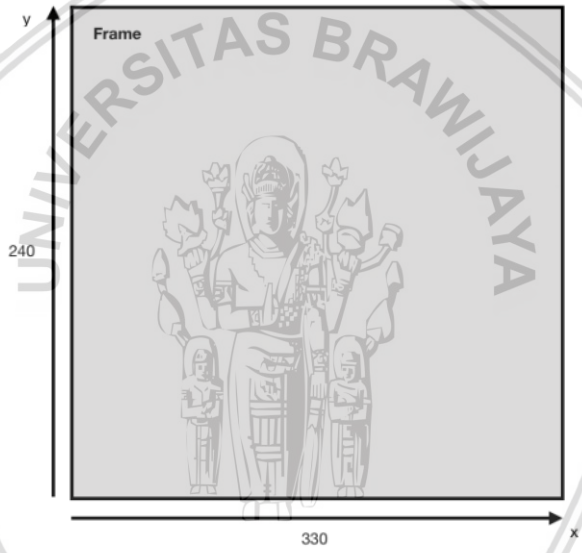
Gambar 5.7 Alur pendeteksian objek

Seperti yang terlihat pada Gambar 5.7 data dari kamera *quadcopter* diolah dengan beberapa metode pengolahan citra diantaranya konversi ruang warna RGB ke *grayscale* agar objek tidak sulit untuk terdeteksi karena akan menghasilkan warna abu-abu yang dilanjutkan dengan proses menghilangkan *noise* menggunakan *blur.* Setelah proses *blur* selesai, akan dilanjutkan dengan deteksi tepian objek menggunakan *canny edge detection*. Metode *Canny Edge Detection* dapat meminimalkan adanya kerancuan pada tepi objek yang akan

dideteksi sehingga memudahkan dalam pengolahan citra sebagai langkah awal untuk mengenali tepi dari objek yang dijadikan sebagai tempat mendarat. Tahap akhir yang dilakukan pada pendeteksian objek adalah pencarian nilai kontur yang sesuai pada intensitas tiap *pixel* yang bertetangga agar mendapatkan tepi-tepi dari objek yang akan dideteksi sehingga *quadcopter* bisa mengenali masing-masing bentuk objek dari tepian yang didapatkan serta dibantu dengan penghitungan sisi objek menggunakan metode *douglass peucker*. Setelah pendeteksian objek selesai dilakukan hasilnya akan ditampilkan pada sebuah *frame.*

5.2.1.1 Mengambil data kamera bawah *quadcopter*

Dalam mendeteksi objek maka dibuatlah sebuah *frame* yang diambil dari data kamera bawah *quadcopter*. Ukuran *frame* yang digunakan untuk pendeteksian objek landasan pada *quadcopter* yaitu 330x240. *Area* dari pendeteksian bisa dilihat pada Gambar 5.8.



Gambar 5.8 Area pendeteksian

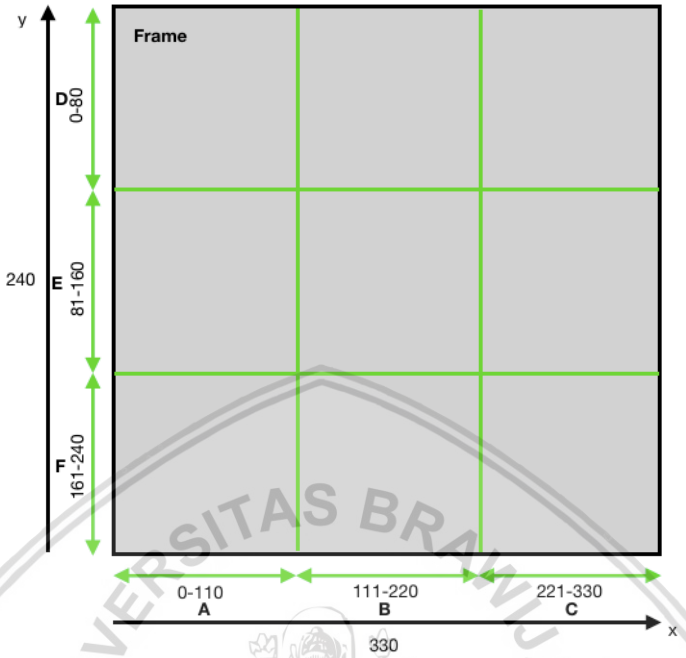
Frame ini akan digunakan untuk menampilkan hasil kamera dari *quadcopter* dan juga untuk fungsi pendeteksian. Dengan menggunakan ukuran *frame* 330x240 *pixel* dapat mempermudah dalam proses mendeteki objek dikarenakan *frame* akan dibagi lagi menjadi beberapa bagian. Dalam pembagian *frame* akan digunakan Persamaan 5.1 untuk pembagian pada sumbu x dan Persamaan 5.2 untuk pembagian pada sumbu y.

$$Pembagian\ sumbu\ x = \frac{sumbu\ x}{3} \tag{5.1}$$

$$Pembagian\ sumbu\ y = \frac{sumbu\ y}{3} \tag{5.2}$$

Berdasarkan persamaan 5.1 pada sumbu x akan dibagi menjadi 3 bagian, masing-masing bagian mempunyai ukuran 110 *pixel*. Sedangkan berdasarkan persamaan 5.2 pada sumbu y akan dibagi menjadi 3 bagian, masing-masing bagian mempunyai ukuran 80 *pixel*. Sehingga pada sumbu x dibagi dengan *range*

0-110 (A), *range* 111-220 (B), dan *range* 221-330 (C). Sedangkan pada sumbu y dibagi dengan *range* 0-80 (D), *range* 81-160 (E), dan *range* 161-240 (F). Untuk pembagian koordinat pada setiap *grid* dapat dilihat pada Gambar 5.9.



Gambar 5.9 Area pendeteksian dengan pembagian letak

5.2.1.2 Mengubah *pixel* dari RGB ke *Grayscale*

Langkah awal yang harus dilakukan adalah mengkonversi *frame* hasil dari data kamera bawah *quadcopter* yang masih berupa RGB ke *grayscale* agar objek yang berwarna tidak sulit untuk terdeteksi. Untuk melakukan konversi digunakan *library* pada opencv yaitu *cvtColor* yang memiliki parameter fungsi *CV_BGR2GRAY* dan *input frame "SRC Image"* yaitu data kamera bawah *quadcopter* yang masih memiliki nilai RGB dan hasil *output* berupa *frame* "*Grayscale Image*" yang merupakan hasil konversi ke *grayscale* seperti yang terlihat pada Gambar 5.10.

Gambar 5.10 Perancangan mengubah RGB ke *Grayscale*

Untuk mendapatkan nilai konversi *image* RGB ke *grayscale* digunakan rumus seperti pada persamaan 2.1. Pada Gambar 5.10 terlihat bahwa titik koordinat x adalah 0 dan y adalah 213 serta memiliki nilai *Red* 102, *Green* 105 dan *Blue* 96.

Dari gambar tersebut didapatkan nilai *luminosity* dengan hasil perhitungan sebagai berikut:

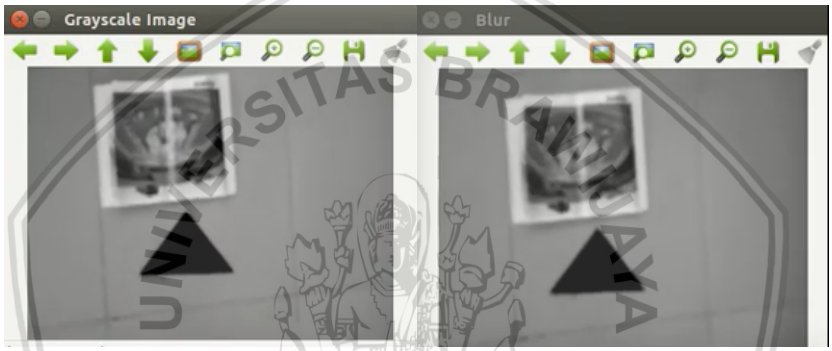
$$L = 0.2989 \text{ x } 102 + 0.5870 \text{ x } 105 + 0.1140 \text{ x } 96$$

$$L = 30{,}4878 + 61{,}635 + 10{,}944$$

$$L = 103{,}0668$$

5.2.1.3 Menghilangkan *noise* dengan *Blur*

Langkah selanjutnya adalah memproses *frame* hasil dari *grayscale* menjadi *blur* untuk menghilangkan *noise* sehingga objek mudah terdeteksi. Untuk melakukan konversi digunakan *library* pada opencv yaitu fungsi *blur* yang memiliki parameter *input frame "Grayscale Image"* yaitu *frame* yang merupakan hasil konversi dari *grayscale* dan hasil *output* berupa *frame* "*Blur"* yang merupakan hasil konversi dari *blur* seperti yang terlihat pada Gambar 5.11.



Gambar 5.11 Perancangan menghilangkan *noise* dengan *Blur*

Untuk mengkoversi nilai *blur* maka dilakukan pencarian nilai setiap *pixel* menggunakan persamaan rumus 2.2 dengan ukuran kernel 3x3 :

$$K = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Sebagai contoh, misalnya pada gambar awal yang didapatkan memiliki nilai matriks :

$$\begin{bmatrix}10 & 10 & 10 & 10\\ 10 & 10 & 10 & 10\\ 10 & 10 & 10 & 10\\ 10 & 10 & 10 & 10\\ 10 & 10 & 10 & 10\end{bmatrix}$$

$$K = \frac{1}{9}\begin{bmatrix}1x0 & 1x0 & 1x0\\ 1x0 & 1x10 & 1x10\\ 1x0 & 1x10 & 1x10\end{bmatrix}$$

$$K = \frac{1}{9}\begin{bmatrix}0 & 0 & 0\\ 0 & 10 & 10\\ 0 & 10 & 10\end{bmatrix}$$

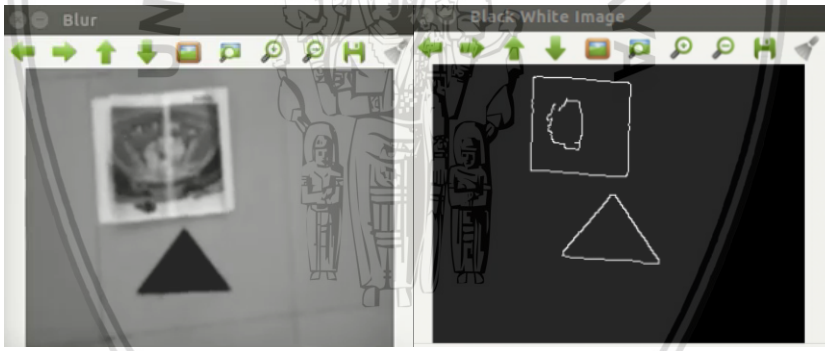
$$K = \frac{40}{9}$$
$$= 4{,}444$$

Perhitungan akan terus berulang untuk pencarian nilai lainnya pada setiap *pixel* gambar. Maka hasil akhir gambar yang telah dikonversi *blur* dengan menggunakan kernel 3x3 akan memiliki nilai matriks sebagai berikut:

$$\begin{bmatrix} 4 & 6 & 6 & 4 \\ 6 & 10 & 10 & 6 \\ 6 & 10 & 10 & 6 \\ 6 & 10 & 10 & 6 \\ 4 & 6 & 6 & 4 \end{bmatrix}$$

5.2.1.4 Mendeteksi tepian dengan *Canny Edge Detection*

Untuk dapat mendeteksi tepian pada objek yang akan dideteksi maka dilakukan pencarian tepian menggunakan *canny edge detection*. Langkah awalnya adalah dengan menggunakan *library* pada opencv yaitu fungsi *canny* dengan parameter *input* dari pendeteksian ini menggunakan *frame* yang telah dikonversi ke *Blur* dan *output frame* menggunakan "*Black White Image*" serta menggunakan nilai minimal dan maksimal untuk *threshold* seperti yang terlihat pada Gambar 5.12.



Gambar 5.12 Mendeteksi tepian dengan *Canny Edge Detection*

Untuk menghitung tepian dengan menggunakan *canny edge detection* ada beberapa tahap yaitu:

1. Mengkoversi menggunakan *Gauissan filter* pada setiap *pixel* dengan ukuran kernel 5x5 yang berfungsi untuk menghilangkan *noise* agar tepian semakin mudah didapatkan dengan menggunakan persamaan 2.2.

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Pixel berikut ini adalah hasil dari *blur* yang akan diteruskan ke proses *Gaussian*. Nilai yang akan dicari adalah *pixel* dengan nilai 10 yang berada pada baris 2 kolom 2.

$$\begin{bmatrix} 4 & 6 & 6 & 4 \\ 6 & 10 & 10 & 6 \\ 6 & 10 & 10 & 6 \\ 6 & 10 & 10 & 6 \\ 4 & 6 & 6 & 4 \end{bmatrix}$$

$$K = \frac{1}{159}\begin{bmatrix} 2x0 & 4x0 & 5x0 & 4x0 & 2x0 \\ 4x0 & 9x4 & 12x6 & 9x6 & 4x4 \\ 5x0 & 12x6 & 15x10 & 12x10 & 5x6 \\ 4x0 & 9x6 & 12x10 & 9x10 & 4x6 \\ 2x0 & 4x6 & 5x10 & 4x10 & 2x6 \end{bmatrix}$$

$$K = \frac{1}{159}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 36 & 72 & 54 & 16 \\ 0 & 72 & 150 & 120 & 30 \\ 0 & 54 & 120 & 90 & 24 \\ 0 & 24 & 50 & 40 & 12 \end{bmatrix}$$

$$K = \frac{964}{159}$$

$$K = 6{,}062$$

Dari perhitungan di atas akan menghasilkan nilai matriks dari *gaussian* yang menggantikan element *pixel* sebelumnya seperti berikut ini.

$$\begin{bmatrix} 4 & 6 & 6 & 4 \\ 6 & 6 & 10 & 6 \\ 6 & 10 & 10 & 6 \\ 6 & 10 & 10 & 6 \\ 4 & 6 & 6 & 4 \end{bmatrix}$$

2. Pencarian nilai intensistas *gradient* pada gambar dengan menggunakan persamaan *sobel* yaitu 2.5, 2.6 dan 2.7 :

   Maka nilai sobel yang didapat dengan menggunakan *pixel* 6 pada baris 2 kolom 2

$$Gx = \begin{bmatrix} -1x4 & 0x6 & +1x6 \\ -2x6 & 0x6 & +2x10 \\ -1x6 & 0x10 & +1x10 \end{bmatrix}$$

$$Gx = \begin{bmatrix} -4 & 0 & +6 \\ -12 & 0 & +20 \\ -6 & 0 & +10 \end{bmatrix}$$

$$Gx = -4 + 0 + 6 + (-12) + 0 + 20 + (-6) + 0 + 10$$

$$= 14$$

$$Gy = \begin{bmatrix} -1x4 & -2x6 & -1x6 \\ 0x6 & 0x6 & x0x10 \\ +1x6 & +2x10 & +1x10 \end{bmatrix}$$

$$Gy = \begin{bmatrix} -4 & -12 & -6 \\ 0 & 0 & 0 \\ +6 & +20 & +10 \end{bmatrix}$$

$$Gy = -4 + (-12 + (-6) + 0 + 0 + 0 + 6 + 20 + 10$$
$$= 14$$
$$G = \sqrt{14^2 + 14^2}$$
$$G = \sqrt{196 + 196}$$
$$G = 19.789$$

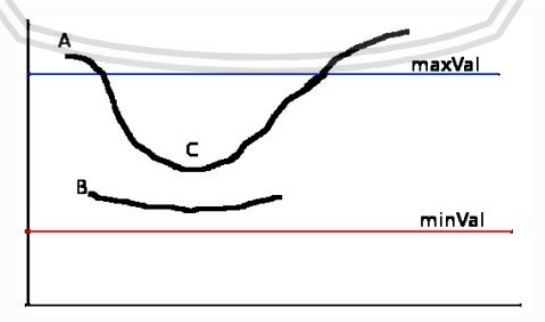
3. Mencari direksi untuk mendapatkan *angle* pada tepian, yang digunakan untuk arah untuk mencari tepian selanjutnya dengan menggunakan persamaan 2.8 maka *angle* yang didapatkan untuk direksinya adalah sebagai berikut.

$$\theta = \arctan\left(\frac{14}{14}\right)$$
$$\theta = \arctan(1)$$
$$\theta = 45°$$

Berikut ini adalah hasil direksi dari sudut $45°$ yang merupakan garis awal yang muncul pada tepian kemudian mengarah ke *pixel* yang berdekatan dan berfungsi untuk membentuk pola objek landasan *quadcopter* seperti bentuk segitiga, kotak dan segillima.

$$\begin{bmatrix} 4 & 6 & 6 & 4 \\ 6 & 19 & 10 & 6 \\ 6 & 10 & 10 & 6 \\ 6 & 10 & 10 & 6 \\ 4 & 6 & 6 & 4 \end{bmatrix}$$

4. Tahap akhir yaitu *Hysteresis Thresholding* yang menggunakan dua *threshold* yaitu *lower* dan *upper.* Berikut ini adalah beberapa kondisi untuk mendapatkan nilai tepian pada tahap akhir *canny edge detection* seperti yang terlihat pada Gambar 5.13.



Gambar 5.13 *Hysteresis Thresholding*

- Jika *gradient* pada *pixel* lebih besar daripada *upper threshold*, maka *pixel* tersebut merupakan tepian seperti yang terlihat pada garis A.
- Jika *gradient* pada *pixel* diantara kedua nilai *threshold*, maka *pixel* tersebut akan diterima jika berhubungan dengan *pixel* yang melebihi dari nilai *upper threshold* seperti yang terlihat pada garis C dan jika nilai *pixel*

tidak terhubung dengan *pixel* yang melebihi dari *upper threshold* maka akan dibuang dari gambar seperti yang terlihat pada garis B.

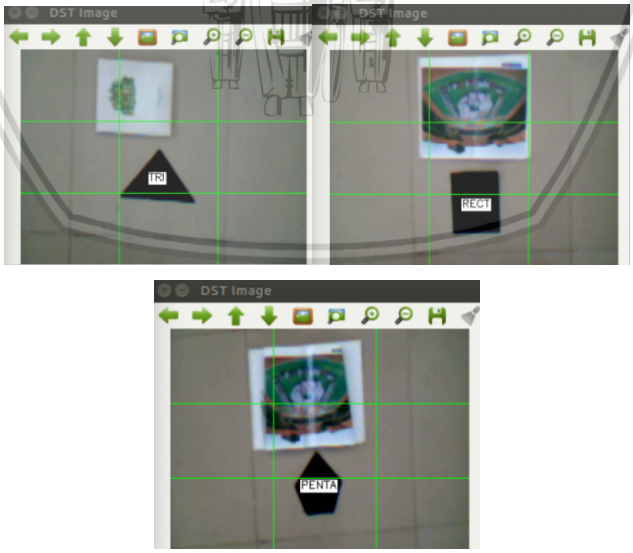
- Jika *gradient* pada *pixel* lebih kecil daripada *lower threshold*, maka *pixel* tersebut dibuang dari gambar atau menjadi hitam.

5.2.1.5 Mencari nilai kontur pada objek

Setelah mendapatkan tepian dengan menggunakan *canny edge detection* maka dilakukan pencarian nilai kontur pada objek yang akan dideteksi. Untuk mencari nilai kontur digunakan *library* pada opencv yaitu fungsi *findContours* dengan parameter *input frame "Black White Image"* yang merupakan hasil pendeteksian tepian dari *canny edge detection* dan *output* variabel dengan nama *countours* yang merupakan nilai dari kontur yang terdeteksi pada objek.

5.2.1.6 Mendeteksi sisi objek dengan *Douglass Pecker*

Untuk dapat mengetahui bentuk objek yang terdeteksi maka diperlukan pendeteksian dan perhitungan sisi pada objek dengan menggunakan *douglas peucker* yaitu perhitungan dari setiap tepi yang telah terdeteksi oleh *canny edge detection* sehingga *quadcopter* dapat mengenali objek yang terdeteksi. Untuk mengetahui bentuk objek digunakan fungsi *library* pada opencv yaitu *approxPolyDP* yang berfungsi menghitung sisi pada objek, jika sisi objek berjumlah 3 maka objek adalah segitiga, jika 4 maka objek adalah kotak, jika 5 maka objek adalah segilima dan jika yang lainnya maka tidak terdeteksi objek. Jika objek telah terdeteksi maka akan muncul label pada titik tengah koordinat objek sesuai nama objek yang terdeteksi dan ditampilkan pada *frame* "*DST Image*" seperti yang terlihat pada Gambar 5.14.



Gambar 5.14 Mendeteksi sisi objek dengan *Douglas Peucker*

Untuk dapat mengetahui bentuk objek yang terdeteksi oleh kamera bawah *quadcopter*, *douglas peucker* mencari setiap tepian garis yang diambil dari titik awal sampai titik ujung dari setiap objek seperti yang terlihat pada Gambar 5.15.

Gambar 5.15 Pencarian setiap titik garis dengan *Douglas Peucker*

5.2.2 Implementasi Deteksi Objek

Pendeteksian objek pada penelitian ini menggunakan metode *douglas peucker* yang awalnya akan melalui tahap konversi ruang warna RGB ke *grayscale* terlebih dahulu agar objek berwarna tidak sulit untuk terdeteksi yang dilanjutkan dengan proses menghilangkan *noise* menggunakan *blur.* Kemudian deteksi tepian dari masing-masing objek menggunakan *canny edge detection* yang berfungsi agar objek tidak mengalami kerancuan selama pendeteksian berlangsung. Langkah selanjutnya pencarian nilai kontur yang sesuai agar mendapatkan tepi-tepi dari objek yang akan dideteksi sehingga *quadcopter* bisa mengenali masing-masing bentuk objek dari tepian yang didapatkan serta dibantu dengan penghitungan sisi objek menggunakan metode *douglass peucker*.

5.2.2.1 Mengambil data kamera bawah *quadcopter*

Implementasi deteksi objek diawali dengan pengambilan data dari kamera *quadcopter*. Seperti pada *source code* baris ke 2 dilakukan pemanggilan data dari kamera *quadcopter* menggunakan ardrone/bottom/image_raw.

Kode Program 5.1 Implementasi kamera *quadcopter*

Kamera *Quadcopter*	
1	`image_transport::ImageTransport it(n);`
2	`image_transport::Subscriber image_sub =`
	`it.subscribe("/ardrone/bottom/image_raw",1,process);`

Setelah pengambilan data dari kamera bawah telah selesai kemudian dilakukan beberapa proses seperti berikut ini.

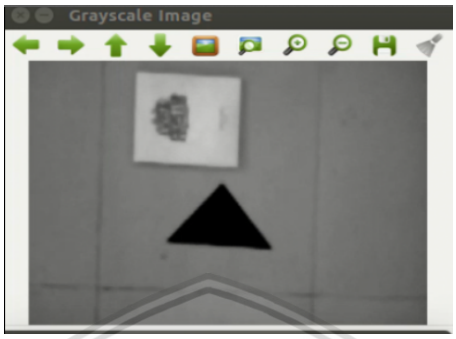
5.2.2.2 Mengubah *pixel* dari RGB ke *Grayscale*

Input dari pendeteksian ini menggunakan *frame* src dan *output*-nya menggunakan *gray*. Untuk mengkonversi RGB ke *grayscale* digunakan fungsi *CV_BGR2GRAY* seperti yang terlihat pada Kode Program 5.2.

Kode Program 5.2 Konversi RGB ke *Grayscale*

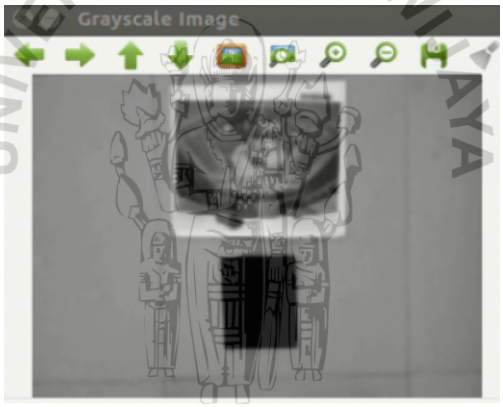
```
Mengubah pixel dari RGB ke Grayscale
1   cv::cvtColor(src, gray, CV_BGR2GRAY);
```

Implementasi dari pengubahan RGB ke *Grayscale* pada objek Segitiga dapat dilihat pada Gambar 5.16.



Gambar 5.16 Implementasi RGB ke *Grayscale* pada Segitiga

Implementasi dari pengubahan RGB ke *Grayscale* pada objek Segitiga dapat dilihat pada Gambar 5.17.



Gambar 5.17 Implementasi RGB ke *Grayscale* pada Kotak

Implementasi dari pengubahan RGB ke *Grayscale* pada objek Segitiga dapat dilihat pada Gambar 5.18.

Gambar 5.18 Implementasi RGB ke *Grayscale* pada Segilima

5.2.2.3 Menghilangkan *noise* dengan *Blur*

Input dari pendeteksian ini menggunakan *frame* yang telah dikonversi ke *Grayscale* yaitu *gray* dan *output frame* bw. Nilai kernel yang digunakan yaitu 3 seperti yang terlihat pada Kode Program 5.3.

Kode Program 5.3 Penghilangan *noise* dengan Blur

```
Menghilangkan noise dengan Blur
1   blur( gray, bw, Size(3,3) );
```

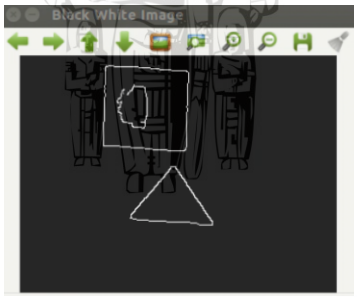
5.2.2.4 Mendeteksi tepian dengan *Canny Edge Detection*

Input dari pendeteksian ini menggunakan *frame* yang telah dikonversi ke *Grayscale* yaitu *gray* dan *output frame* menggunakan bw. Nilai minimal untuk *threshold* adalah 130 dan maksimal adalah 250 seperti terlihat pada Kode Program 5.4.

Kode Program 5.4 Deteksi tepian dengan *Canny Edge Detection*

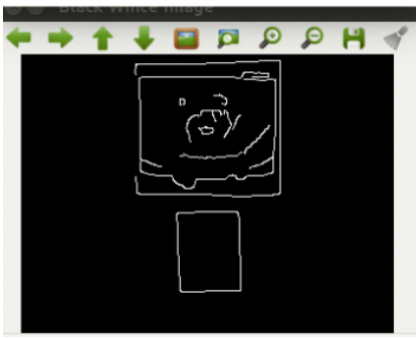
```
Mendeteksi tepian dengan Canny Edge Detection
1   cv::Canny(gray, bw, 130, 250, 3);
```

Implementasi *canny edge detection* untuk mendeteksi tepi pada objek dapat dilihat pada gambar-gambar berikut ini. Tepian garis yang terdeteksi oleh *canny edge detection* membentuk sebuah objek segitiga seperti pada Gambar 5.19.



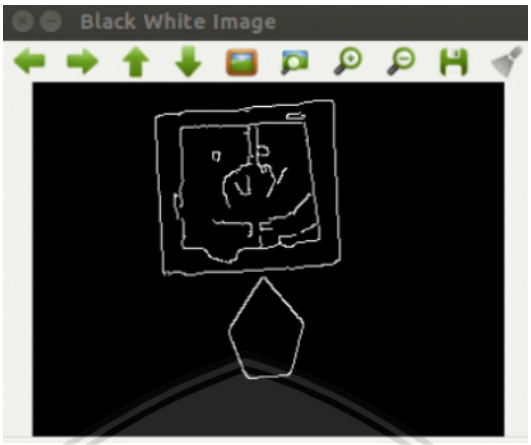
Gambar 5.19 Implementasi *Canny Edge Detection* pada Segitiga

Tepian garis yang terdeteksi oleh *canny edge detection* membentuk sebuah objek kotak seperti pada Gambar 5.20.



Gambar 5.20 Implementasi *Canny Edge Detection* pada Kotak

Tepian garis yang terdeteksi oleh *canny edge detection* membentuk sebuah objek segilima seperti yang terlihat pada Gambar 5.21.



Gambar 5.21 Implementasi *Canny Edge Detection* pada Segilima

5.2.2.5 Mencari nilai kontur pada objek

Input yang digunakan adalah yang telah diproses dengan *canny edge detection* yaitu *frame* bw dan menggunakan variabel *countours* sebagai *output* dari nilai kontur yang terdeteksi seperti yang terlihat pada Kode Program 5.5.

Kode Program 5.5 Mencari nilai kontur pada objek

```
Mencari nilai kontur pada objek
1   cv::findContours(bw.clone(), contours, CV_RETR_EXTERNAL
    CV_CHAIN_APPROX_SIMPLE);
```

5.2.2.6 Mendeteksi sisi objek dengan *Douglass Pecker*

Input yang digunakan adalah *countours* yaitu hasil dari nilai kontur yang telah terbaca pada objek dan *approx* sebagai *output* seperti yang terlihat pada Kode Program 5.6.

Kode Program 5.6 Deteksi sisi objek dengan *Douglass Pecker*

```
Mendeteksi sisi objek dengan Douglass Pecker
1   for (int i = 0; i < contours.size(); i++)
2   {
3
4   Double epsilon = arcLength(Mat(contours[i], true) * 0.02;
5
6   cv::approxPolyDP(cv::Mat(contours[i]), approx, epsilon,
7   true);
8
9   // Skip small or non-convex objects
10  if (std::fabs(cv::contourArea(contours[i])) < 100 ||
11  !cv::isContourConvex(approx))
12  continue;
```

Mendeteksi sisi objek dengan *Douglass Pecker*

```
13  // Number of vertices of polygonal curve
14  int vtc = approx.size();
15
16  // Use the number of vertices
17  // to determine the shape of the contour
18  if (vtc == 3 && nomerObjek == 3)
19  {
20  setLabel(dst, "TRI", contours[i]);    // Segitiga
21  }
22  else if (vtc >= 4 && nomerObjek == 4)
23  {
24  setLabel(dst, "RECT", contours[i]); //Kotak
25  }
26  else if (vtc >= 5 && nomerObjek == 5)
27  {
28  setLabel(dst, "PENTA", contours[i]); //Segilima
29  }
30  else
31  {
32  gotShape = false; //Objek tidak ditemukan
33  CoordShape.x = 999;
34  CoordShape.y = 999;
35  tipeObjek = "";
36  }
37  }
```

Untuk melakukan pencarian sisi objek digunakan fungsi *approxPolyDP* yang bertujuan menghitung sisi pada objek. Jika sisi objek berjumlah 3 maka objek adalah segitiga, jika 4 maka objek adalah kotak, jika 5 maka objek adalah segilima dan jika yang lainnya maka tidak terdeteksi objek.

Setelah mendeteksi objek yang dideteksi maka akan menampilkan label sesuai dengan objek yang dideteksi pada koordinat titik tengah objek tersebut. Agar mendapatkan titik tengah masing-masing dari bentuk landasan, maka dilakukan perhitungan terhadap titik tengah area kontur objek pada koordinat *x* yaitu dengan membagi lebar objek dan koordinat *y* dengan cara membagi tinggi objek. Titik tengah pada objek digunakan sebagai label berupa pemberian nama sesuai dengan objek yang terdeteksi dan sebagai acuan pergerakan otomatis yang akan dilakukan sesuai dengan posisi koordinat titik tengah seperti yang terlihat pada Kode Program 5.7.

Kode Program 5.7 Pemberian label dan pencarian titik koordinat tengah pada objek yang telah terdeteksi

Fungsi pemberian label dan pencarian titik koordinat tengah

```
1  void setLabel(cv::Mat& im, const std::string label,
2  std::vector<cv::Point>& contour)
```

Fungsi pemberian label dan pencarian titik koordinat tengah

```
3   {
4   int fontface = cv::FONT_HERSHEY_SIMPLEX;
5   double scale = 0.4;
6   int thickness = 1;
7   int baseline = 0;
8   cout << std::fabs(cv::contourArea(contour));
9   cv::Size text = cv::getTextSize(label, fontface, scale,
10  thickness, &baseline);
11  cv::Rect r = cv::boundingRect(contour);
12  gotShape = true;
13  cv::Point pt(r.x + ((r.width - text.width) / 2), r.y +
14  ((r.height + text.height) / 2));
15  CoordShape.x = r.x + (r.width/ 2);
16  CoordShape.y = r.y + (r.height/ 2);
17  tipeObjek = label;
18  cv::rectangle(im, pt + cv::Point(0, baseline), pt +
19  cv::Point(text.width, -text.height), CV_RGB(255,255,255),
20  CV_FILLED);
21  cv::putText(im, label, pt, fontface, scale, CV_RGB(0,0,0),
22  thickness, 8);
23  }
```

Sebelum *quadcopter* melakukan pendeteksan objek, *user* dapat memilih terlebih dahulu objek yang ingin dideteksi. Untuk memilih objek yang akan dideteksi *user* dapat memilih sesuai yang diinginkan. Jika *user* ingin mendeteksi segitiga maka *user* dapat memasukkan angka 3 pada terminal, jika *user* ingin mendeteksi kotak maka *user* dapat memasukkan angka 4 pada terminal dan jika *user* ingin mendeteksi segilima maka *user* dapat memasukkan angka 5 pada terminal seperti yang terlihat pada Kode Program 5.8.

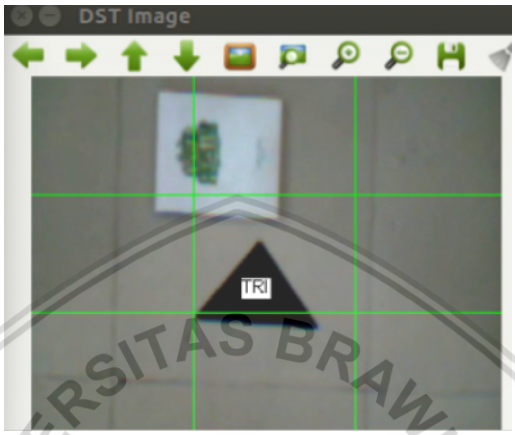
Kode Program 5.8 Pemilihan objek yang akan dideteksi

Fungsi pemilihan objek yang akan dideteksi

```
1   case '3':
2   cout<<"Objek segitiga akan dideteksi"<<endl;
3   nomerObjek = 3;
4   command = '~';
5   break;
6   case '4':
7   cout<<"Objek kotak akan dideteksi"<<endl;
8   nomerObjek = 4;
9   command = '~';
10  break;
11  case '5':
12  cout<<"Objek penta akan dideteksi";
13  nomerObjek = 5;
14  command = '~';
```

Fungsi pemilihan objek yang akan dideteksi	
15	break;

Jika objek berhasil terdeteksi, maka akan menampilkan label sesuai dengan nama objek tersebut tepat pada titik tengah objek. Pada Gambar 5.22 terlihat bahwa objek yang terdeteksi adalah segitiga, maka label yang ditampilkan adalah "TRI".

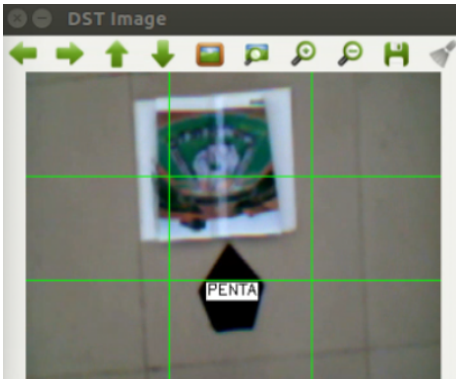


Gambar 5.22 Implementasi *Douglass Pecker* pada objek segitiga

Pada Gambar 5.23 terlihat bahwa objek yang terdeteksi adalah kotak, maka label yang ditampilkan adalah "RECT".

Gambar 5.23 Implementasi *Douglass Pecker* pada objek kotak

Pada Gambar 5.24 terlihat bahwa objek yang terdeteksi adalah segilima, maka label yang ditampilkan adalah "PENTA".

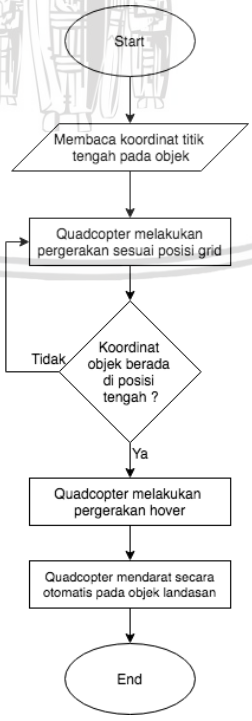


Gambar 5.24 Implementasi *Douglass Pecker* pada objek segilima

5.3 Pergerakan *Quadcopter*

5.3.1 Perancangan Pergerakan *Quadcopter*

Pada perancangan pergerakan *quadcopter* akan digunakan pembacaan pendeteksian objek landasan yang berfungsi sebagai *input* pada sistem. Kemudian *quadcopter* akan melakukan pergerakan secara otomatis sesuai dengan posisi objek yang terdeteksi. Pergerakan otomatis ini berlangsung hingga titik tengah objek berada di *area grid* tengah pada *frame* kamera *quadcopter.* Jika *quadcopter* berhasil melakukan pergerakan otomatis sampai dengan pendeteksian objek yang berada di *area grid* tengah pada *frame,* maka akan melakukan pendaratan secara otomatis. Berikut ini adalah pergerakan otomatis yang akan dihasilkan oleh *quadcopter* agar bisa mendeteksi objek dan mendarat secara otomatis seperti yang terlihat pada Gambar 5.25.



Gambar 5.25 Diagram Alir Pergerakan Quadcopter

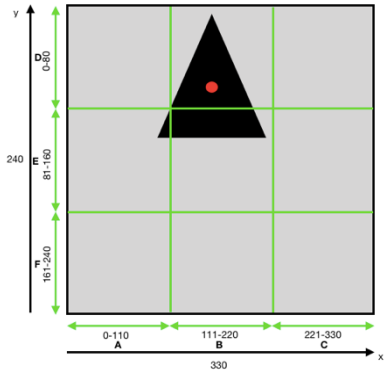
Pada Tabel 5.1 terdapat penjelasan untuk masing-masing pergerakan sesuai koordinat dari *frame* kamera bawah *quadcopter*.

Tabel 5.1 Pergerakan *quadcopter* sesuai pada koordinat *frame*

***Frame* Kamera**	**Gerakan *Quadcopter***
Pada *frame* BD yaitu sumbu x *area* 111-220 dan sumbu y dengan *area* 0-80.	Gerak Maju
Pada *frame* BF yaitu sumbu x area 111-220 dan sumbu y dengan area 161-240.	Gerak Mundur
Pada *frame* AF yaitu sumbu x *area* 0-110 dan sumbu y dengan *area* 161-240. Pada *frame* AE yaitu sumbu x area 0-110 dan sumbu y dengan area 81-160. Pada *frame* AD yaitu sumbu x area 0-110 dan sumbu y dengan area 0-80.	Geser Kiri
Pada *frame* CF yaitu sumbu x area 221-330 dan sumbu y dengan area 161-240. Pada *frame* CE yaitu sumbu x area 221-330 dan sumbu y dengan area 81-160. Pada *frame* CD yaitu sumbu x area 221-330 dan sumbu y dengan area 0-80.	Geser kanan
Pada *frame* BE yaitu sumbu x *area* 111-220 dan sumbu y dengan *area* 81-160.	*Landing*
Jika pada *frame* tidak menemukan objek.	*Hover*

1. Gerak maju

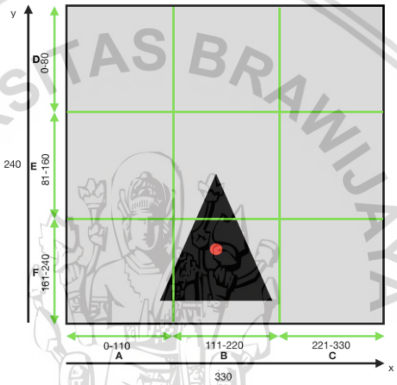
   Q*uadcopter* akan bergerak maju ketika objek yang terdeteksi berada pada bagian *frame* BD yaitu sumbu x *area* 111-220 dan sumbu y dengan *area* 0-80. Letak objek saat gerak maju dapat dilihat pada Gambar 5.26.



Gambar 5.26 Area pendeteksian gerak maju

2. Gerak mundur

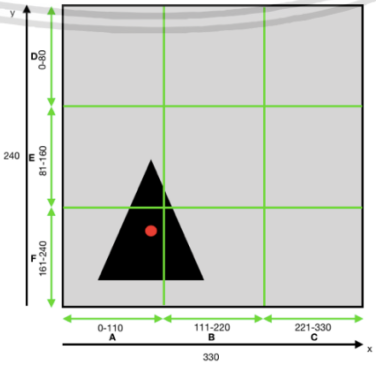
   Q*uadcopter* akan bergerak mundur ketika objek yang terdeteksi berada pada bagian *frame* BF yaitu sumbu x *area* 111-220 dan sumbu y dengan *area* 161-240. Letak objek saat gerak mundur dapat dilihat pada Gambar 5.27.



Gambar 5.27 Area pendeteksian gerak mundur

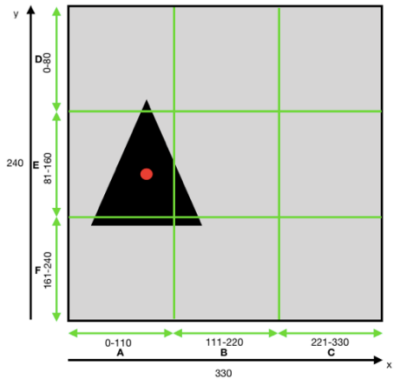
3. Geser Kiri

   Q*uadcopter* akan bergeser ke kiri ketika objek yang terdeteksi berada pada bagian *frame* AF yaitu sumbu x *area* 0-110 dan sumbu y dengan *area* 161-240. Letak objek saat gerak geser kiri dapat dilihat seperti pada Gambar 5.28.



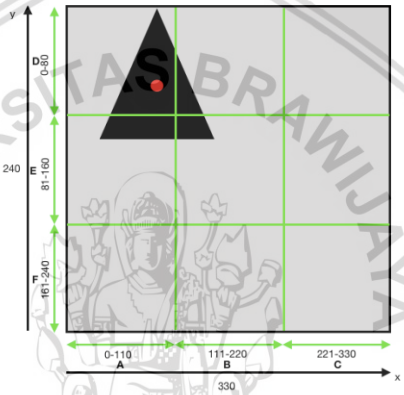
Gambar 5.28 Area pendeteksian geser kiri pada kotak *frame* AF

Q*uadcopter* akan bergeser ke kiri ketika objek yang terdeteksi berada pada bagian *frame* AE yaitu sumbu x *area* 0-110 dan sumbu y dengan *area* 81-160. Letak objek saat gerak geser kiri dapat dilihat seperti pada Gambar 5.29.



Gambar 5.29 Area pendeteksian geser kiri pada kotak *frame* AE

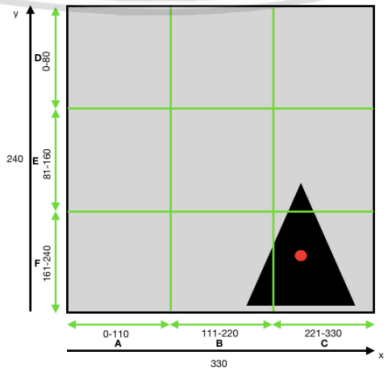
Q*uadcopter* akan bergeser ke kiri ketika objek yang terdeteksi berada pada bagian *frame* AD yaitu sumbu x *area* 0-110 dan sumbu y dengan *area* 0-80. Letak objek saat gerak geser kiri dapat dilihat seperti pada Gambar 5.30.



Gambar 5.30 Area pendeteksian geser kiri pada kotak *frame* AD

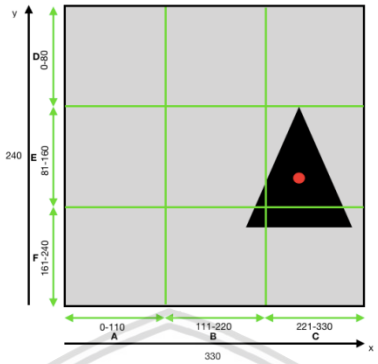
4. Geser Kanan

Q*uadcopter* akan bergeser ke kanan ketika objek yang terdeteksi berada pada bagian *frame* CF yaitu sumbu x *area* 221-330 dan sumbu y dengan *area* 161-240. Letak objek saat gerak geser kanan dapat dilihat pada Gambar 5.31.



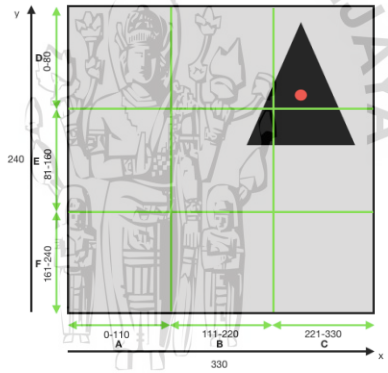
Gambar 5.31 Area pendeteksian geser kanan pada kotak *frame* CF

Q*uadcopter* akan bergeser ke kanan ketika objek yang terdeteksi berada pada bagian *frame* CE yaitu sumbu x *area* 221-330 dan sumbu y dengan *area* 81-160. Letak objek saat gerak geser kanan dapat dilihat pada Gambar 5.32.



Gambar 5.32 Area pendeteksian geser kanan pada kotak *frame* CE

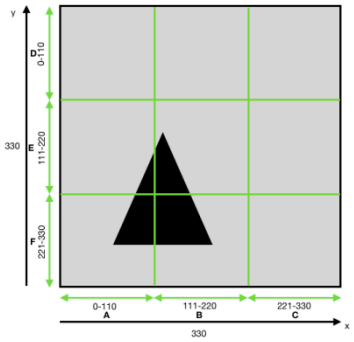
Q*uadcopter* akan bergeser ke kanan ketika objek yang terdeteksi berada pada bagian *frame* CD yaitu sumbu x *area* 221-330 dan sumbu y dengan *area* 0-80. Letak objek saat gerak geser kanan dapat dilihat pada Gambar 5.33.



Gambar 5.33 Area pendeteksian geser kanan pada kotak *frame* CD

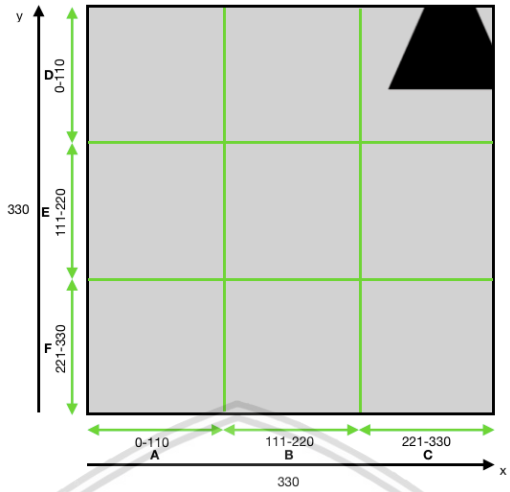
5. *Hover*

Q*uadcopter* akan *hover* atau tidak bergerak ketika belum menemukan titil tengah saat pendeteksian berlangsung. Letak objek saat *hover* dapat dilihat pada Gambar 5.34.



Gambar 5.34 Area pendeteksian *hover*

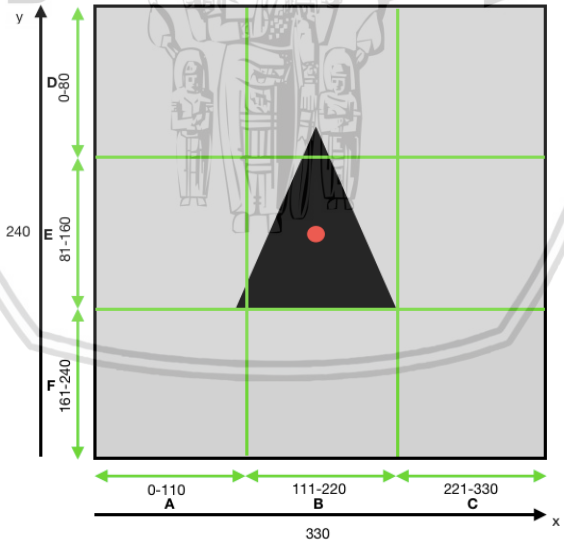
Kondisi *hover* lainnya dapat dilihat pada Gambar 5.35 ketika objek belum terdeteksi secara optimal.



Gambar 5.35 Area pendeteksian *hover*

6. *Landing*

Q*uadcopter* akan mendarat secara otomatis ketika objek yang terdeteksi berada pada bagian *frame* BE yaitu sumbu x *area* 111-220 dan sumbu y dengan *area* 81-160. Letak objek saat *landing* dapat dilihat pada Gambar 5.36.



Gambar 5.36 Area pendeteksian *Landing* pada kotak *frame* BE

5.3.2 Implementasi Pergerakan *Quadcopter*

Setelah perancangan pergerakan telah dibuat, langkah selanjutnya adalah pembuatan *source code* untuk setiap gerakan *quadcopter* dari mulai mencari objek hingga dapat melakukan pergerakan otomatis ketika mendeteksi objek sesuai dengan letak koordinat pada *area grid* dan dapat mendarat otomatis

dengan tepat. Adapun *source code* untuk masing-masing gerakan pada *quadcopter* adalah sebagai berikut.

1. Gerak Maju

   Inisialisasi gerak maju, *quadcopter* bergerak maju secara linear dengan memberikan nilai 0.02 pada sumbu x dan memberikan nilai 0 pada sumbu linear y dan z seperti yang terlihat pada Kode Program 5.9.

Kode Program 5.9 Inisialisasi gerak maju

```
Inisialisasi gerak maju
1  trackForward.linear.x=0.02;
2  trackForward.linear.y=0.0;
3  trackForward.linear.z=0.0;
4  trackForward.angular.z=0.0;
```

   Inisialisasi untuk melakukan fungsi Gerak Maju pada drone dengan melakukan *publish* untuk mengirimkan nilai trackForward yang telah diinisialisasi seperti yang terlihat pada Kode Program 5.10.

Kode Program 5.10 Fungsi gerak maju

```
Fungsi gerak maju
1  case '7':
2  pub_twist.publish(trackForward); //drone fly forward
3  command = '~';
4  break;
```

   Inisialisasi koordinat objek saat gerak maju dilakukan dengan membaca titik tengah objek yang terdeteksi. Jika koordinat titik tengah objek berada diantara area sumbu x 111-220 dan area sumbu y pada 0-80 maka akan melakukan pergerakan otomatis maju seperti yang terlihat pada Kode Program 5.11.

Kode Program 5.11 Koordinat objek saat gerak maju

```
Koordinat objek saat gerak maju
1  else if (CoordShape.x>110 && CoordShape.x<=220 &&
2  CoordShape.y >0 && CoordShape.y <=80 ) {
3  command = '7'; //maju ke depan jika diatas tengah
4  }
```

2. Gerak Mundur

   Inisialisasi gerak mundur, *quadcopter* bergerak maju secara linear dengan memberikan nilai -0.02 pada sumbu x dan memberikan nilai 0 pada sumbu linear y dan z seperti yang terlihat pada Kode Program 5.12.

Kode Program 5.12 Inisialisasi gerak mundur

```
Inisialisasi gerak mundur
1   trackBackward.linear.x=-0.02;
2   trackBackward.linear.y=0.0;
3   trackBackward.linear.z=0.0;
4   trackBackward.angular.z=0.0;
```

Inisialisasi untuk fungsi Gerak Mundur pada *quadcopter* dengan melakukan *publish* untuk mengirimkan nilai trackBackward yang telah diinisialisasi seperti yang terlihat pada Kode Program 5.13.

Kode Program 5.13 Fungsi gerak mundur

```
Fungsi gerak mundur
1   case '8':
2   pub_twist.publish(trackBackward); //drone fly backward
3   command = '~';
4   break;
```

Inisialisasi koordinat objek saat gerak mundur dilakukan dengan membaca titik tengah objek yang terdeteksi. Jika koordinat titik tengah objek berada diantara area sumbu x 111-220 dan area sumbu y pada 160-240 maka akan melakukan pergerakan otomatis mundur seperti yang terlihat pada Kode Program 5.14.

Kode Program 5.14 Koordinat objek saat gerak mundur

```
Koordinat objek saat gerak mundur
1   else if (CoordShape.x>110 && CoordShape.x<=220 &&
2   CoordShape.y >160 && CoordShape.y <=240 ) {
3   command = '8'; //maju ke depan jika diatas tengah
4   }
```

3. Geser Kiri

Inisialisasi geser kiri, *quadcopter* bergeser ke kiri secara linear dengan memberikan nilai 0.02 pada sumbu y dan memberikan nilai 0 pada sumbu linear x dan z seperti yang terlihat pada Kode Program 5.15.

Kode Program 5.15 Inisialisasi geser kiri

```
Inisialisasi gerak geser kiri
1   trackLeft.linear.x=0.0;
2   trackLeft.linear.y=0.02;
3   trackLeft.linear.z=0.0;
4   trackLeft.angular.z=0.0;
```

Inisialisasi untuk melakukan fungsi Geser Kiri pada drone dengan melakukan *publish* untuk mengirimkan nilai trackLeft yang telah diinisialisasi seperti yang terlihat pada Kode Program 5.16.

Kode Program 5.16 Fungsi geser kiri

```
Fungsi gerak geser kiri
1   case '3':
2   pub_twist.publish(trackLeft); //drone fly to left
3   command = '~';
4   break;
```

Inisialisasi koordinat objek saat gerak geser kiri dilakukan dengan membaca titik tengah objek yang terdeteksi. Jika koordinat titik tengah objek berada diantara area sumbu x 221-330 dan area sumbu y pada 0-240 maka akan melakukan pergerakan otomatis geser kiri seperti yang terlihat pada Kode Program 5.17.

Kode Program 5.17 Koordinat objek saat geser kiri

```
Koordinat objek saat gerak geser kiri
1   else if (CoordShape.x>220 && CoordShape.x<=330 &&
2   CoordShape.y >0 && CoordShape.y <=80 ) {
3         command = '3'; //geser kiri jika kotak kiri atas
4   } else if (CoordShape.x>220 && CoordShape.x<=330 &&
5   CoordShape.y >80 && CoordShape.y <=160 ) {
6         command = '3'; //geser kiri jika kotak kiri tengah
7   } else if (CoordShape.x>220 && CoordShape.x<=330 &&
8   CoordShape.y >160 && CoordShape.y <=240 ) {
9         command = '3'; //geser kiri jika kotak kiri bawah
10  }
```

4. Geser Kanan

Inisialisasi geser kanan, *quadcopter* bergeser ke kanan secara linear dengan memberikan nilai -0.02 pada sumbu y dan memberikan nilai 0 pada sumbu linear x dan z seperti yang terlihat pada Kode Program 5.18.

Kode Program 5.18 Inisialisasi geser kanan

```
Inisialisasi gerak geser kanan
1   trackRight.linear.x=0.0;
2   trackRight.linear.y=-0.02;
3   trackRight.linear.z=0.0;
4   trackRight.angular.z=0.0;
```

Inisialisasi untuk melakukan fungsi Geser Kanan pada drone dengan melakukan *publish* untuk mengirimkan nilai trackRight yang telah diinisialisasi seperti yang terlihat pada Kode Program 5.19.

Kode Program 5.19 Fungsi geser kanan

```
Fungsi gerak geser kanan
1   case '4':
2   pub_twist.publish(trackRight); //drone fly to right
3   command = '~';
4   break;
```

Inisialisasi koordinat objek saat geser kanan dilakukan dengan membaca titik tengah objek yang terdeteksi. Jika koordinat titik tengah objek berada diantara area sumbu x 0-110 dan area sumbu y pada 0-240 maka akan melakukan pergerakan otomatis geser kanan seperti yang terlihat pada Kode Program 5.20.

Kode Program 5.20 Koordinat objek saat geser kanan

```
Koordinat objek saat gerak geser kanan
1   if (CoordShape.x>0 && CoordShape.x<=110 && CoordShape.y >0
2   && CoordShape.y <=80 ) {
3         command = '4'; //geser kanan jika kotak kiri atas
4   } else if (CoordShape.x>0 && CoordShape.x<=110 &&
5   CoordShape.y >80 && CoordShape.y <160 ) {
6         command = '4'; //geser kanan jika kotak kiri atas
7   } else if (CoordShape.x>0 && CoordShape.x<=110 &&
8   CoordShape.y >160 && CoordShape.y <240 ) {
9         command = '4'; //geser kanan jika kotak kiri atas
10  }
```

5. *Hover*

Inisialisasi gerak hover, *quadcopter* tidak melakukan gerakan sehingga memberikan nilai 0 pada sumbu y, sumbu x dan sumbu z seperti yang terlihat pada Kode Program 5.21.

Kode Program 5.21 Inisialisasi hover

```
Inisialisasi gerak hover
1   hoverStop.linear.x=0.0;
2   hoverStop.linear.y=-0.0;
3   hoverStop.linear.z=0.0;
4   hoverStop.angular.z=0.0;
```

Inisialisasi untuk melakukan fungsi gerakan *Hover* pada drone dengan melakukan *publish* untuk mengirimkan nilai hoverStop yang telah diinisialisasi seperti yang terlihat pada Kode Program 5.22.

Kode Program 5.22 Fungsi hover

```
Fungsi gerak hover
1   case '5':
2   pub_twist.publish(hoverStop); //drone stop
3   command = '~';
4   break;
```

Jika titik tengah dari koordinat objek tidak terbaca maka akan melakukan gerakan hover untuk menghentikan *quadcopter* seperti yang terlihat pada Kode Program 5.23.

Kode Program 5.23 Kondisi saat hover

```
Koordinat objek saat gerak hover
1   else{
2           //Hover jika tidak menemui objek
3           char Areas[20];
4           sprintf(Areas,"Hover");
5           putText(dst,Areas,Point(10,40), FONT_HERSHEY_PLAIN,
6   1, Scalar(0,255,0),2);
7           command = '5';
8   }
```

6. *Landing*

Inisialisasi *source code landing*, dengan memanggil library pada Ardrone Autonomy yang diinisialisasi pada variabel pub_empty_land seperti yang terlihat pada Kode Program 5.24.

Kode Program 5.24 Inisialisasi *landing*

```
Inisialisasi landing
1   ros::Publisher pub_empty_land;
2   pub_empty_land    =
    node.advertise<std_msgs::Empty>("/ardrone/land", -1);
```

Inisialisasi untuk melakukan fungsi *Landing* pada *quadcopter* dengan melakukan *publish* untuk mengirimkan nilai hoverStop dan melakukan publish untuk memanggil fungsi pada library Ardrone Autonomy untuk *landing* seperti yang terlihat pada Kode Program 5.25.

Kode Program 5.25 Fungsi *landing*

```
Fungsi gerak landing
1   case '6':
2   pub_twist.publish(hoverStop); //drone is flat
3   pub_empty_land.publish(msg); //lands the drone
4   command = '~';
5   otomatis = false;
```

Fungsi gerak landing	
6	exit(0);

Inisialisasi koordinat objek saat *landing* dilakukan dengan membaca titik tengah objek yang terdeteksi. Jika koordinat titik tengah objek berada diantara area sumbu x 111-220 dan area sumbu y pada 80-160 yaitu berada posisi *frame* lokasi tengah maka akan melakukan *landing* dan akan menghentikan fungsi *tracking* objek seperti yang terlihat pada Kode Program 5.26.

Kode Program 5.26 Koordinat objek saat *landing*

Koordinat objek saat gerak *landing*	
1	else if (CoordShape.x>110 && CoordShape.x<=220 &&
2	CoordShape.y >80 && CoordShape.y <=160) {
3	char Areas[20];
4	sprintf(Areas,"Landing");
5	putText(dst,Areas,Point(10,40), FONT_HERSHEY_PLAIN,
6	1, Scalar(0,255,0),2);
7	command = '5';
8	isLanding = true;
9	isTrack = false;
10	}

Berikut ini terdapat gerakan secara manual yang dilakukan dengan menggunakan tombol pada *keyboard* untuk mengerakkan *quadcopter* :

1. *Take off*

   Inisialisasi *source code take off* manual, dengan memanggil library pada Ardrone Autonomy yang diinisialisasi pada variabel pub_empty_take off seperti yang terlihat pada Kode Program 5.27.

Kode Program 5.27 Inisialisasi *take off*

Inisialisasi *take off*	
1	ros::Publisher pub_empty_takeoff;
2	pub_empty_takeoff =
	node.advertise<std_msgs::Empty>("/ardrone/takeoff", 1);

Inisialisasi untuk melakukan fungsi *Take off* manual pada *quadcopter* dengan *user* meng*input* 't' pada terminal yang berfungsi untuk melakukan *publish* lalu mengirimkan nilai hoverStop dan melakukan publish untuk memanggil fungsi pada library Ardrone Autonomy untuk *Take Off* seperti yang terlihat pada Kode Program 5.28.

Kode Program 5.28 Fungsi *take off*

Fungsi take off	
1	case 't':
2	pub_empty_takeoff.publish(msg); //launches the drone
3	pub_twist.publish(hoverStop);

```
Fungsi take off
4         cout<<"Taking Off"<<endl;
5         command='~';
6         break;
```

2. *Landing*

Inisialisasi *source code landing* manual, dengan memanggil library pada Ardrone Autonomy yang diinisialisasi pada variabel pub_empty_land seperti yang terlihat pada Kode Program 5.29.

Kode Program 5.29 Inisialisasi *landing*

```
Inisialisasi Landing
1  ros::Publisher pub_empty_land;
2  pub_empty_land =
   node.advertise<std_msgs::Empty>("/ardrone/land", 1);
```

Inisialisasi untuk melakukan fungsi *Landing* manual pada *quadcopter* dengan *user* meng*input* 'l' pada terminal yang berfungsi untuk berfungsi untuk melakukan *publish* lalu mengirimkan nilai hoverStop dan melakukan *publish* untuk untuk memanggil fungsi pada library Ardrone Autonomy untuk *Landing* seperti yang terlihat pada Kode Program 5.30.

Kode Program 5.30 Fungsi *Landing*

```
Fungsi gerak geser kanan
1  case 'l':
2  pub_twist.publish(hoverStop); //drone is flat
3  pub_empty_land.publish(msg); //lands the drone
4  command = '~';
5  otomatis = false;
6  exit(0);
7  break;
```

3. Gerakan Maju

Inisialisasi gerak, *quadcopter* bergeser ke depan secara linear dengan memberikan nilai 0.02 pada sumbu x dan memberikan nilai 0 pada sumbu linear y dan z seperti yang terlihat pada Kode Program 5.31.

Kode Program 5.31 Inisialisasi Gerak Maju

```
Inisialisasi gerak maju
1  moveForward.linear.x=0.02;
2  moveForward.linear.y=0.0;
3  moveForward.linear.z=0.0;
4  moveForward.angular.z=0.0;
```

Inisialisasi untuk melakukan fungsi Gerakan Maju manual pada *quadcopter* dengan *user* meng*input* 'w' pada terminal yang berfungsi untuk melakukan *publish* untuk mengirimkan nilai moveForward yang telah diinisialisasi dan akan memberikan nilai "Maju" pada variabel serta memberi nilai *true* pada variabel isMoving seperti yang terlihat pada Kode Program 5.32.

Kode Program 5.32 Fungsi gerak maju

```
Fungsi gerak maju
1   case 'w':
2   cout<<"Gerakan Maju"<<endl;
3   otomatis = true;
4   gerakan = "Maju";
5   gotShape = false;
6   isTrack = false;
7   isMoving = true;
8   break;
```

4. Gerakan Mundur

Inisialisasi gerak, *quadcopter* bergeser ke depan secara linear dengan memberikan nilai -0.02 pada sumbu x dan memberikan nilai 0 pada sumbu linear y dan z seperti yang terlihat pada Kode Program 5.33.

Kode Program 5.33 Inisialisasi Gerak Mundur

```
Inisialisasi gerak mundur
1   moveBackward.linear.x=-0.02;
2   moveBackward.linear.y=0.0;
3   moveBackward.linear.z=0.0;
4   moveBackward.angular.z=0.0;
```

Inisialisasi untuk melakukan fungsi Gerakan Mundur manual pada *quadcopter* dengan *user* meng*input* 's' pada terminal yang berfungsi untuk melakukan *publish* untuk mengirimkan nilai moveBackward yang telah diinisialisasi dan akan memberikan nilai "Mundur" pada variabel serta memberi nilai *true* pada variabel isMoving seperti yang terlihat pada Kode Program 5.34.

Kode Program 5.34 Fungsi gerak mundur

```
Fungsi gerak mundur
1   case 's':
2   cout<<"Gerakan Mundur"<<endl;
3   otomatis = true;
4   gerakan = "Mundur";
5   gotShape = false;
6   isTrack = false;
7   isMoving = true;
```

Fungsi gerak mundur	
8	break;

5. Gerakan Geser Kiri

Inisialisasi gerak, *quadcopter* bergeser ke depan secara linear dengan memberikan nilai 0.02 pada sumbu y dan memberikan nilai 0 pada sumbu linear x dan z seperti yang terlihat pada Kode Program 5.35.

Kode Program 5.35 Inisialisasi Gerak Geser Kiri

Inisialisasi gerak geser kiri	
1	moveLeft.linear.x=0.0;
2	moveLeft.linear.y=0.02;
3	moveLeft.linear.z=0.0;
4	moveLeft.angular.z=0.0;

Inisialisasi untuk melakukan fungsi Geser Kiri manual pada drone dengan *user* meng*input* 'a' pada terminal yang berfungsi melakukan *publish* untuk mengirimkan nilai moveForward yang telah diinisialisasi dan akan memberikan nilai "Kiri" pada variabel serta memberi nilai *true* pada variabel isMoving seperti yang terlihat pada Kode Program 5.36.

Kode Program 5.36 Fungsi geser kiri

Fungsi gerak geser kiri	
1	case 'w':
2	cout<<"Gerakan Geser Kiri"<<endl;
3	otomatis = true;
4	gerakan = "Kiri";
5	gotShape = false;
6	isTrack = false;
7	isMoving = true;
8	break;

6. Gerakan Geser Kanan

Inisialisasi gerak, *quadcopter* bergeser ke depan secara linear dengan memberikan nilai -0.02 pada sumbu y dan memberikan nilai 0 pada sumbu linear x dan z seperti yang terlihat pada Kode Program 5.37.

Kode Program 5.37 Inisialisasi Geser Kanan

Inisialisasi gerak kanan	
1	moveRight.linear.x=0.0;
2	moveRight.linear.y=-0.02;
3	moveRight.linear.z=0.0;
4	moveRight.angular.z=0.0;

Inisialisasi untuk melakukan fungsi Geser Kanan manual pada *quadcopter* dengan *user* meng*input* 'd' pada terminal yang berfungsi melakukan *publish* untuk mengirimkan nilai moveRight yang telah diinisialisasi dan akan memberikan nilai "Kanan" pada variabel serta memberi nilai *true* pada variabel isMoving seperti yang terlihat pada Kode Program 5.38.

Kode Program 5.38 Fungsi geser kanan

```
Fungsi gerak geser kanan
1  case 'w':
2  cout<<"Gerakan Geser Kanan"<<endl;
3  otomatis = true;
4  gerakan = "Kanan";
5  gotShape = false;
6  isTrack = false;
7  isMoving = true;
8  break;
```

BAB 6 PENGUJIAN DAN ANALISIS

Pada bab pengujian dan analisis akan dibahas mengenai pengujian dari sistem yang telah dibuat serta analisisnya. Ada tiga pengujian yang akan dilakukan yaitu pengujian ketinggian untuk mendeteksi objek, pengujian pergerakan *quadcopter* dengan kecepatan dan ketinggian yang berbeda-beda dan ketepatan pendaratan di tengah objek landasan.

6.1 Pengujian Ketinggian

6.1.1 Tujuan Pengujian

Tujuan dilakukannya pengujian ini adalah untuk menguji pendeteksian objek dengan beberapa ketinggian dan untuk mengetahui apakah *quadcopter* dapat mendeteksi objek sebagaimana mestinya agar dapat mencapai mekanisme pendaratan otomatis dengan mengenali berbagai bentuk seperti segitiga, kotak dan segilima yang merupakan objek landasan.

6.1.2 Pelaksanaan Pengujian

Pengujian dilakukan dengan menerbangkan *quadcopter* dan mengatur nilai ketinggian yang berbeda untuk setiap pengujian. Agar dapat mengetahui nilai persentase keakuratan dengan ketinggian yang berbeda pada *quadcopter*, maka akan dilakukan pemantauan pada *frame* kamera bawah *quadcopter* untuk melihat setiap objek yang berhasil terdeteksi dengan ketinggian yang berbeda.

6.1.3 Prosedur Pengujian

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

1. Memasang objek berwarna hitam dengan bentuk segitiga, kotak dan segilima yang dibuat sebelumnya.
2. Menyiapkan *quadcopter*, memasang baterai *quadcopter* dan menunggu hingga *quadcopter* terkalibrasi. Setelah terdengar suara *beep* sebanyak empat kali maka *quadcopter* siap digunakan.
3. Menghubungkan *PC* dengan *hotspot Wi-fi* yang ada pada *quadcopter.*
4. Membuka *terminal* pada *ubuntu* dan masuk ke direktori dengan menggetik “cd tum_simulator_ws”, kemudian mengetik “source devel/setup.bash”. setelah itu mengetik “roslaunch ardrone_autonomy ardrone.launch” untuk menghubungkan *ROS* dengan *quadcopter*.
5. Membuka *terminal* baru lagi pada *ubuntu* dan masuk ke direktori dengan menggetik “cd tum_simulator_ws”. Langkah selanjutnya mengetik “rosservice call /ardrone/togglecam” untuk menggunakan kamera bawah *quadcopter*.

6. Membuka *terminal* lagi pada *ubuntu* dan masuk ke direktori dengan menggetik "cd tum_simulator_ws", kemudian mengetik "source devel/setup.bash". Kemudian mengetik "rosrun testopencv_node" untuk melakukan *run* pada program.
7. Memposisikan *quadcopter* tepat diatas objek landasan.
8. Mengatur nilai ketinggian sesuai dengan pengujian yang akan dilakukan.
9. Memilih bentuk objek yang akan dideteksi.
10. Mengamati *output* dari kamera *quadcopter* pada terminal untuk mengetahui apakah objek dapat terdeteksi dengan berbagai ketinggian.

6.1.4 Hasil Pengujian

Setelah semua pengujian dilakukan maka didapatkan nilai ketinggian minimal dan nilai ketinggian maksimal untuk mendeteksi objek seperti yang dijelaskan berikut ini.

1. Objek segitiga pada ketinggian 125 cm

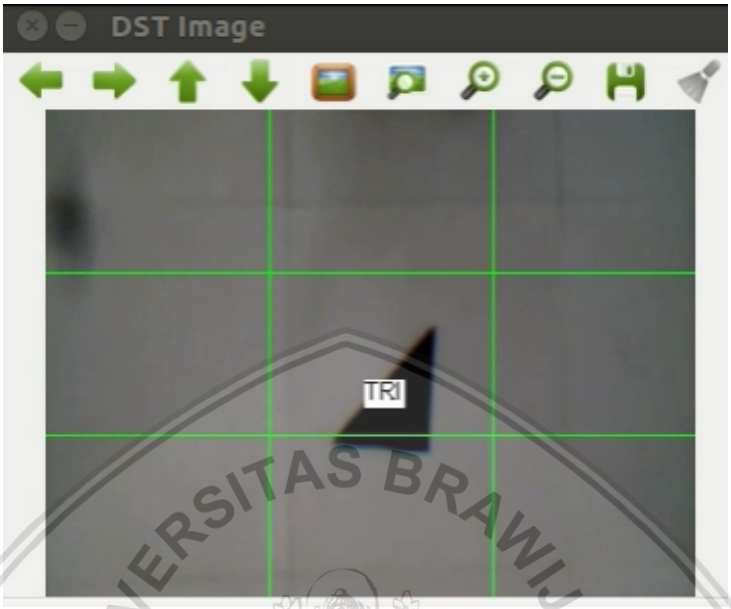
   Pada ketinggian 125 cm, *quadcopter* bisa mendeteksi objek dengan bentuk segitiga tanpa mengalami kendala dan setelah berhasil mendeteksi maka akan menampilkan label sesuai dengan jenis bentuk objek yang terdeteksi. Jika objek segitiga yang terdeteksi, maka tulisan label adalah "TRI" dan posisi label berada tepat di titik tengah objek seperti yang terlihat pada Gambar 6.1.

Gambar 6.1 Objek Segitiga pada Ketinggian 125 cm.

2. Objek segitiga pada ketinggian 150 cm

   Pada ketinggian 150 cm, *quadcopter* bisa mendeteksi objek dengan bentuk segitiga tanpa mengalami kendala dan setelah berhasil mendeteksi maka

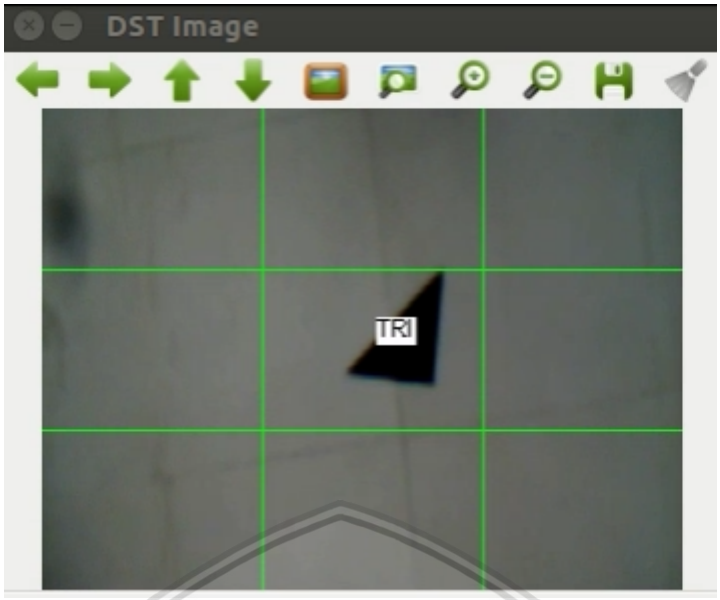
akan menampilkan label sesuai dengan jenis bentuk objek yang terdeteksi. Jika objek segitiga yang terdeteksi, maka tulisan label adalah “TRI” dan posisi label berada tepat di titik tengah objek seperti yang terlihat pada Gambar 6.2.



Gambar 6.2 Objek segitiga pada ketinggian 150 cm.

3. Objek segitiga pada ketinggian 175 cm

   Pada ketinggian 175 cm, *quadcopter* bisa mendeteksi objek dengan bentuk segitiga tanpa mengalami kendala dan setelah berhasil mendeteksi maka akan menampilkan label sesuai dengan jenis bentuk objek yang terdeteksi. Jika objek segitiga yang terdeteksi, maka tulisan label adalah “TRI” dan posisi label berada tepat di titik tengah objek seperti yang terlihat pada Gambar 6.3.



Gambar 6.3 Objek segitiga pada ketinggian 175 cm.

4. Objek segitiga pada ketinggian 200 cm

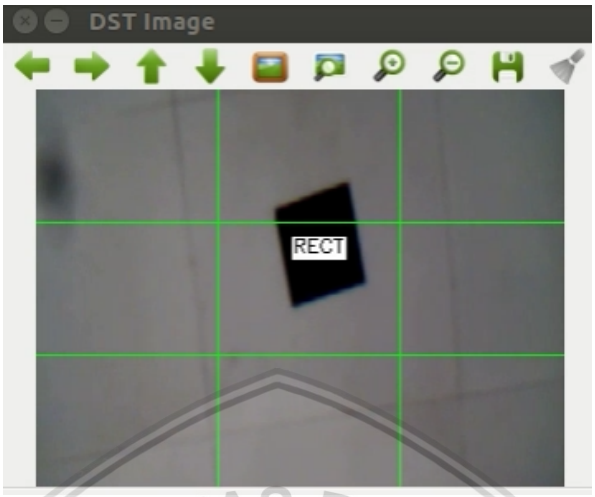
   Pada ketinggian 150 cm, *quadcopter* bisa mendeteksi objek dengan bentuk segitiga tanpa mengalami kendala dan setelah berhasil mendeteksi maka akan menampilkan label sesuai dengan jenis bentuk objek yang terdeteksi. Jika objek segitiga yang terdeteksi, maka tulisan label adalah "TRI" dan posisi label berada tepat di titik tengah objek seperti yang terlihat pada Gambar 6.4.

Gambar 6.4 Objek segitiga pada ketinggian 200 cm.

5. Objek kotak pada ketinggian 125 cm

   Pada ketinggian 125 cm, *quadcopter* bisa mendeteksi objek dengan bentuk kotak tanpa mengalami kendala dan setelah berhasil mendeteksi maka akan menampilkan label sesuai dengan jenis bentuk objek yang terdeteksi. Jika

objek kotak yang terdeteksi, maka tulisan label adalah “RECT” dan posisi label berada tepat di titik tengah objek seperti yang terlihat pada Gambar 6.5.



Gambar 6.5 Objek kotak pada Ketinggian 125 cm.

6. Objek kotak pada ketinggian 150 cm

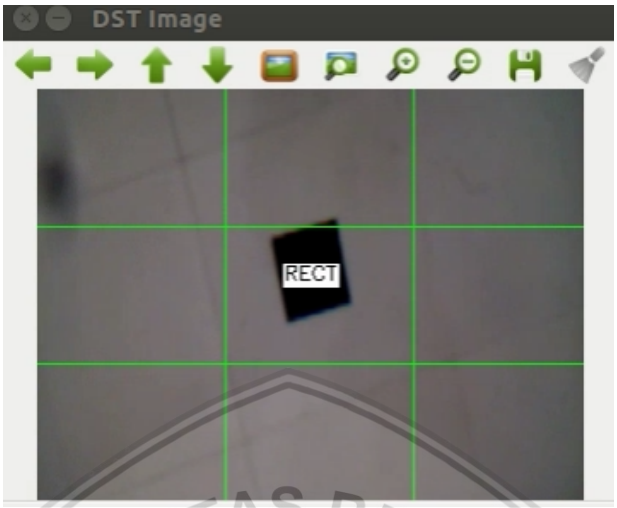
   Pada ketinggian 150 cm, *quadcopter* bisa mendeteksi objek dengan bentuk kotak tanpa mengalami kendala dan setelah berhasil mendeteksi maka akan menampilkan label sesuai dengan jenis bentuk objek yang terdeteksi. Jika objek kotak yang terdeteksi, maka tulisan label adalah “RECT” dan posisi label berada tepat di titik tengah objek seperti yang terlihat pada Gambar 6.6.

Gambar 6.6 Objek kotak pada ketinggian 150 cm.

7. Objek kotak pada ketinggian 175 cm

   Pada ketinggian 175 cm, *quadcopter* bisa mendeteksi objek dengan bentuk kotak tanpa mengalami kendala dan setelah berhasil mendeteksi maka akan menampilkan label sesuai dengan jenis bentuk objek yang terdeteksi. Jika

objek kotak yang terdeteksi, maka tulisan label adalah “RECT” dan posisi label berada tepat di titik tengah objek seperti yang terlihat pada Gambar 6.7.



Gambar 6.7 Objek kotak pada ketinggian 175 cm.

8. Objek kotak pada ketinggian 200 cm

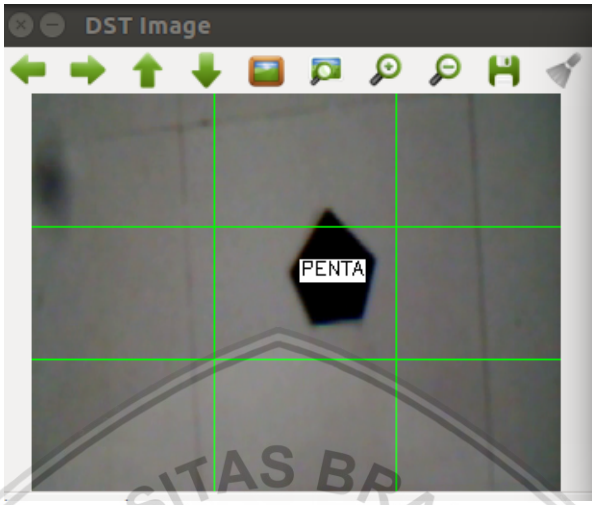
   Pada ketinggian 200 cm, *quadcopter* bisa mendeteksi objek dengan bentuk kotak tanpa mengalami kendala dan setelah berhasil mendeteksi maka akan menampilkan label sesuai dengan jenis bentuk objek yang terdeteksi. Jika objek kotak yang terdeteksi, maka tulisan label adalah “RECT” dan posisi label berada tepat di titik tengah objek seperti yang terlihat pada seperti yang terlihat pada Gambar 6.8.

Gambar 6.8 Objek kotak pada ketinggian 200 cm.

9. Objek segilima pada ketinggian 125 cm

   Pada ketinggian 125 cm, *quadcopter* bisa mendeteksi objek dengan bentuk segilima tanpa mengalami kendala dan setelah berhasil mendeteksi maka

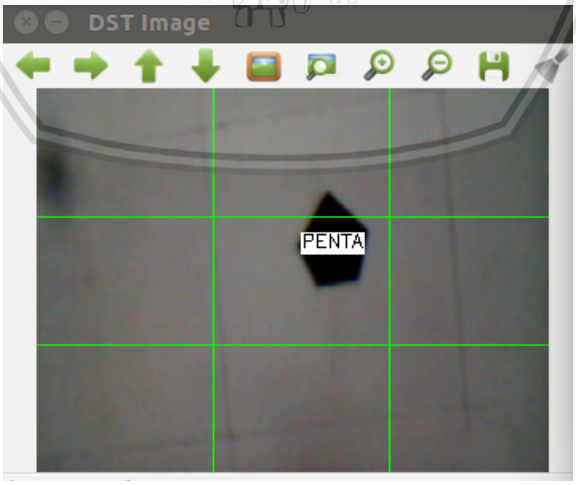
akan menampilkan label sesuai dengan jenis bentuk objek yang terdeteksi. Jika objek segilima yang terdeteksi, maka tulisan label adalah “PENTA” dan posisi label berada tepat di titik tengah objek seperti yang terlihat pada seperti yang terlihat pada Gambar 6.9.



Gambar 6.9 Objek Segitiga pada Ketinggian 125 cm.

10. Objek segitiga pada ketinggian 150 cm

    Pada ketinggian 150 cm, *quadcopter* bisa mendeteksi objek tanpa mengalami kendala dan setelah berhasil mendeteksi maka akan menampilkan label sesuai dengan jenis bentuk objek yang terdeteksi. Jika objek segilima yang terdeteksi, maka tulisan label adalah “PENTA” dan posisi label berada tepat di titik tengah objek seperti yang terlihat pada Gambar 6.10.

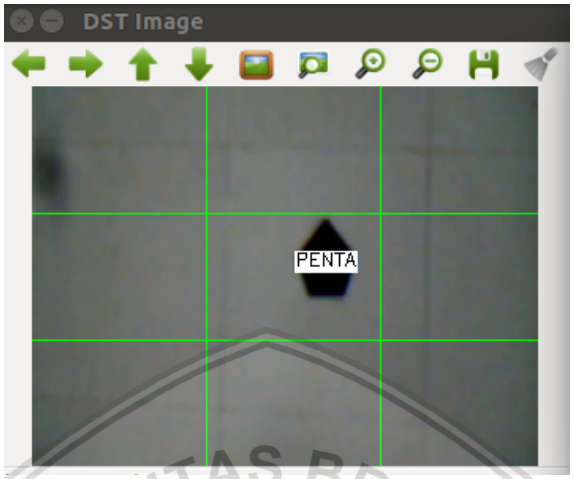


Gambar 6.10 Objek segitiga pada ketinggian 150 cm.

11. Objek segitiga pada ketinggian 175 cm

    Pada ketinggian 175 cm, *quadcopter* bisa mendeteksi objek tanpa mengalami kendala dan setelah berhasil mendeteksi maka akan

menampilkan label sesuai dengan jenis bentuk objek yang terdeteksi. Jika objek segilima yang terdeteksi, maka tulisan label adalah “PENTA” dan posisi label berada tepat di titik tengah objek seperti yang terlihat pada Gambar 6.11.

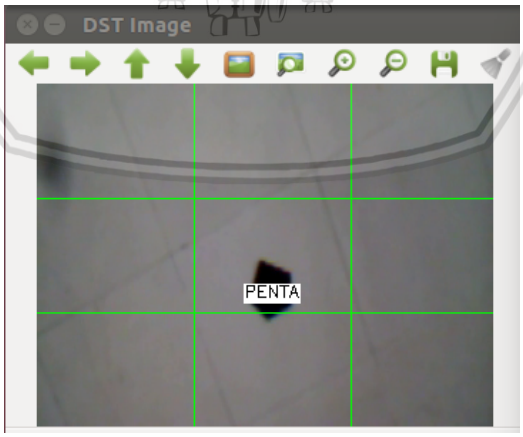


Gambar 6.11 Objek segitiga pada ketinggian 175 cm.

12. Objek segitiga pada ketinggian 200 cm

Pada ketinggian 150 cm, *quadcopter* bisa mendeteksi objek tanpa mengalami kendala dan setelah berhasil mendeteksi maka akan menampilkan label sesuai dengan jenis bentuk objek yang terdeteksi. Jika objek segilima yang terdeteksi, maka tulisan label adalah “PENTA” dan posisi

label berada tepat di titik tengah objek seperti yang terlihat pada Gambar 6.12.



Gambar 6.12 Objek segitiga pada ketinggian 200 cm.

6.1.5 Analisis Hasil Pengujian

Setelah pengujian terhadap ketinggian dilakukan dengan rentan nilai yang berbeda-beda sebanyak lima kali untuk masing-masing ketinggian, hasil yang didapatkan seperti yang terlihat pada Tabel 6.1. Keberhasilan pengujian diberi

tanda *checklist (√)* sedangkan pengujian yang tidak berhasil diberi tanda *cross (x).* Berdasarkan hasil tersebut terlihat bahwa *quadcopter* dapat mendeteksi objek secara otomatis dengan tiga bentuk objek yaitu segitiga, kotak, dan segilima pada rentan ketinggian mulai dari 125 cm sampai dengan ketinggian 200 cm.

Tabel 6.1 Tabel Hasil Pengujian Ketinggian Deteksi Objek

No	Objek	Ketinggian	Pengujian ke-					Hasil Deteksi
			1	2	3	4	5	
1	Segitiga	125	√	√	√	√	√	100%
		150	√	√	√	√	√	100%
		175	√	√	√	√	√	100%
		200	√	√	√	√	√	100%
2	Kotak	125	√	√	√	√	√	100%
		150	√	√	√	√	√	100%
		175	√	√	√	√	√	100%
		200	√	√	√	√	√	100%
3	Segilima	125	√	√	√	√	√	100%
		150	√	√	√	√	√	100%
		175	√	√	√	√	√	100%
		200	√	√	√	√	√	100%
TOTAL PERSENTASE HASIL DETEKSI OBJEK								100%

6.2 Pengujian Ketepatan Pergerakan dengan Kecepatan yang Berbeda

6.2.1 Tujuan Pengujian

Tujuan dilakukan pengujian ini untuk mengetahui apakah sistem sudah mencapai kesesuaian dengan perancangan yang telah dibuat dan *quadcopter* dapat memberikan *output* yang tepat dalam melakukan pergerakan otomatis untuk mendekati titik tengah pada objek yang terdeteksi sampai dengan *quadcopter* dapat mendarat otomatis.

6.2.2 Pelaksanaan Pengujian

Pelaksanaan pengujian ini yaitu dengan menerbangkan *quadcopter* dengan kecepatan yang berbeda. Untuk mengetahui setiap pergerakan *quadcopter* sesuai dengan objek yang terdeteksi, maka akan dilakukan pemantauan pada *frame* kamera bawah *quadcopter* untuk melihat setiap gerakan.

6.2.3 Prosedur Pengujian

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

1. Memasang objek berwarna hitam dengan bentuk segitiga, kotak dan segilima yang dibuat sebelumnya.
2. Menyiapkan *quadcopter*, memasang baterai *quadcopter* dan menunggu hingga *quadcopter* terkalibrasi. Setelah terdengar suara *beep* sebanyak empat kali maka *quadcopter* siap digunakan.
3. Menghubungkan *PC* dengan *hotspot Wi-fi* yang ada pada *quadcopter.*
4. Membuka *terminal* pada *ubuntu* dan masuk ke direktori dengan menggetik "cd tum_simulator_ws", kemudian mengetik "source devel/setup.bash". setelah itu mengetik "roslaunch ardrone_autonomy ardrone.launch" untuk menghubungkan *ROS* dengan *quadcopter*.
5. Membuka *terminal* baru lagi pada *ubuntu* dan masuk ke direktori dengan menggetik "cd tum_simulator_ws". Langkah selanjutnya mengetik "rosservice call /ardrone/togglecam" untuk menggunakan kamera bawah *quadcopter*.
6. Membuka *terminal* lagi pada *ubuntu* dan masuk ke direktori dengan menggetik "cd tum_simulator_ws", kemudian mengetik "source devel/setup.bash". Kemudian mengetik "rosrun testopencv_node" untuk melakukan *run* pada program.
7. Memposisikan *quadcopter* dekat dengan objek sesuai arah pergerakan yang akan dilakukan.
8. Menjalankan pergerakan *quadcopter* dengan perintah otomatis.
9. Mengamati *output* dari kamera *quadcopter* pada terminal untuk mengetahui apakah objek dapat terdeteksi dengan berbagai ketinggian.
10. Mengulangi prosedur ke 8 hingga lima kali percobaan dan mencatat hasil yang didapatkan.

6.2.4 Hasil Pengujian

Berikut ini adalah *output* yang dihasilkan dari pengujian yang telah dilakukan.

1. Gerakan maju

   Pada Gambar 6.13 terlihat titik tengah objek berada pada *frame* kamera bawah *quadcopter* dan posisi objek ada pada kotak *grid* tengah atas sehingga ketika terdeteksi, *quadcopter* akan melakukan pergerakan maju secara otomatis.

Gambar 6.13 Pengujian gerakan maju

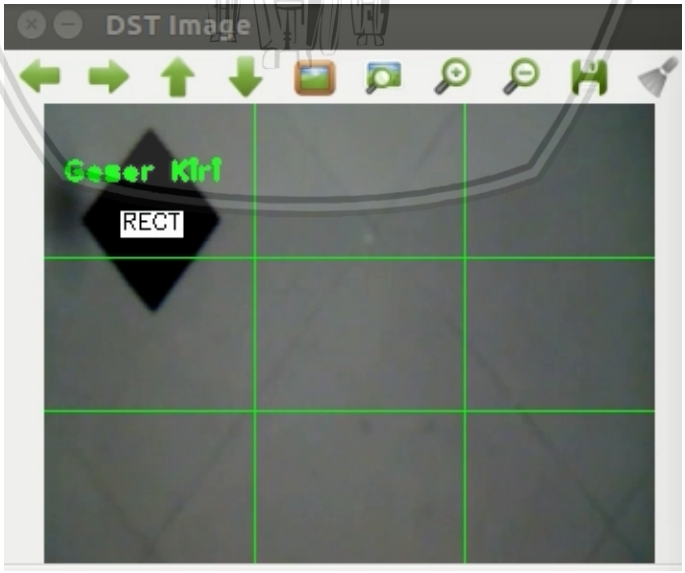
2. Gerakan mundur

   Pada Gambar 6.14 terlihat titik tengah objek berada pada *frame* kamera bawah *quadcopter* dan posisi objek ada pada kotak *grid* tengah bawah sehingga ketika terdeteksi, *quadcopter* akan melakukan pergerakan mundur secara otomatis.

Gambar 6.14 Pengujian gerakan mundur

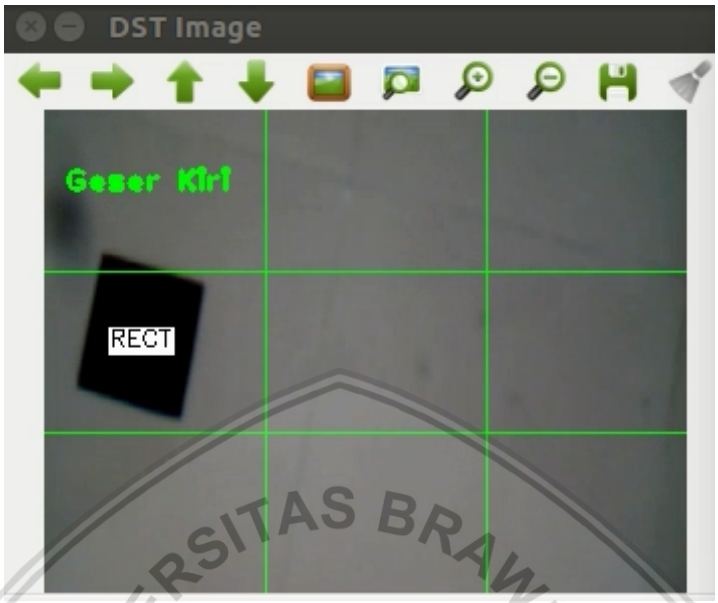
3. Gerakan ke kiri

   Pada Gambar 6.15, Gambar 6.16 dan Gambar 6.17 terlihat titik tengah objek berada pada *frame* kamera bawah *quadcopter* dan posisi objek ada pada kotak *grid* bagian kiri atas sehingga ketika terdeteksi, *quadcopter* akan melakukan pergerakan ke kiri secara otomatis.



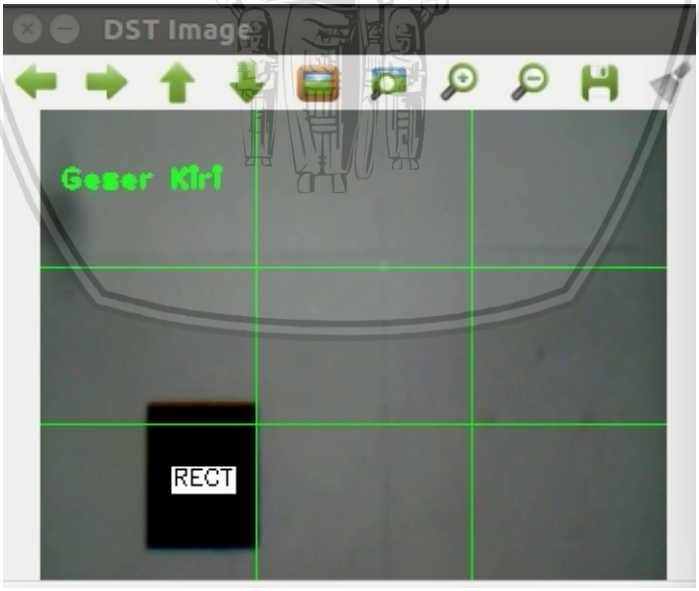
Gambar 6.15 Pengujian gerakan ke kiri

Jika posisi objek berada pada kotak *grid* bagian kiri tengah dan objek terdeteksi pada area tersebut, maka *quadcopter* akan melakukan pergerakan ke kiri secara otomatis.



Gambar 6.16 Pengujian gerakan ke kiri

Jika posisi objek berada pada kotak *grid* bagian kiri bawah dan objek terdeteksi pada area tersebut, maka *quadcopter* akan melakukan pergerakan ke kiri secara otomatis.

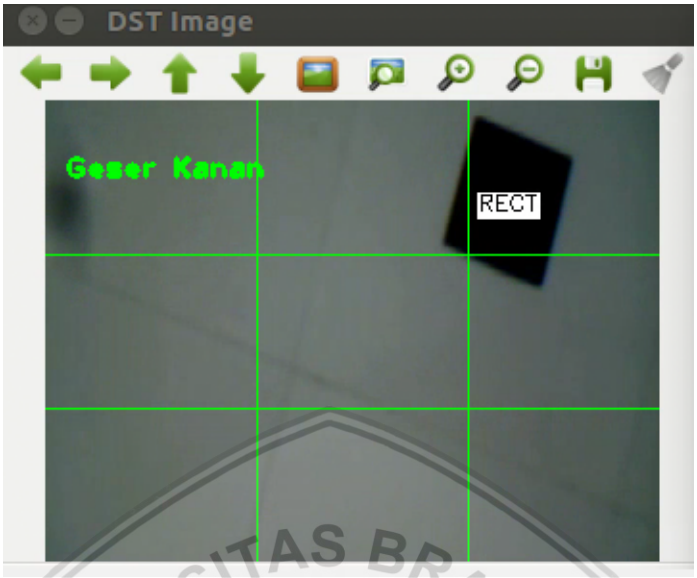


Gambar 6.17 Pengujian gerakan ke kiri

4. Gerakan ke kanan

   Pada Gambar 6.18, Gambar 6.19 dan Gambar 6.20 terlihat titik tengah objek berada pada *frame* kamera bawah *quadcopter* dan posisi objek ada pada

kotak *grid* bagian kanan sehingga ketika terdeteksi, *quadcopter* akan melakukan pergerakan ke kanan secara otomatis.

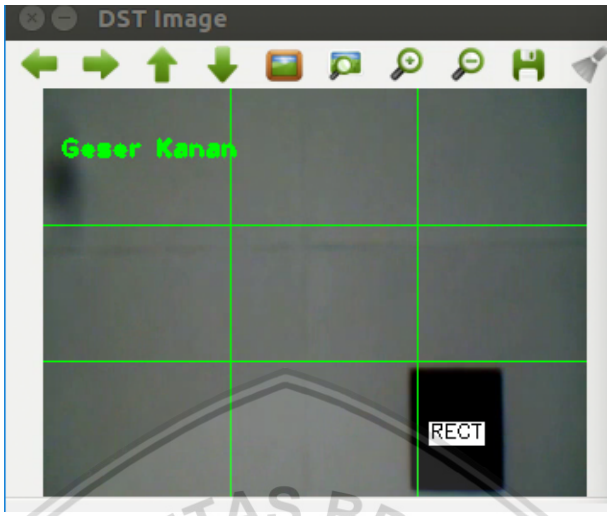


Gambar 6.18 Pengujian gerakan ke kanan

Jika posisi objek berada pada kotak *grid* bagian kanan atas dan objek terdeteksi pada area tersebut, maka *quadcopter* akan melakukan pergerakan ke kanan secara otomatis.

Gambar 6.19 Pengujian gerakan ke kanan

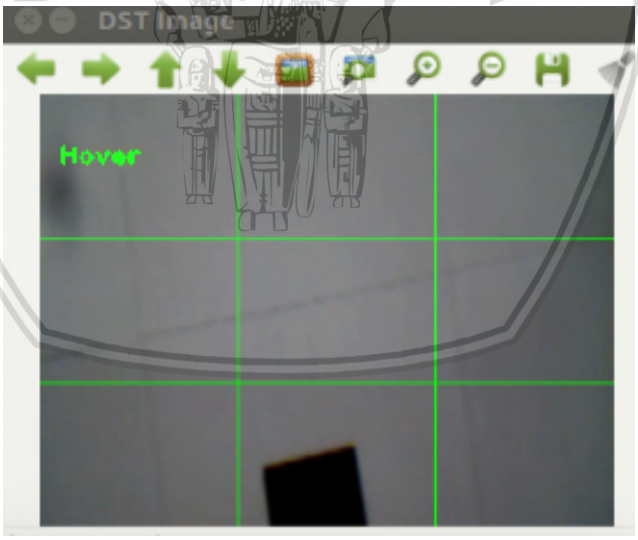
Jika posisi objek berada pada kotak *grid* bagian kanan bawah dan objek terdeteksi pada area tersebut, maka *quadcopter* akan melakukan pergerakan ke kanan secara otomatis.



Gambar 6.20 Pengujian gerakan ke kanan

5. Gerakan *hover*

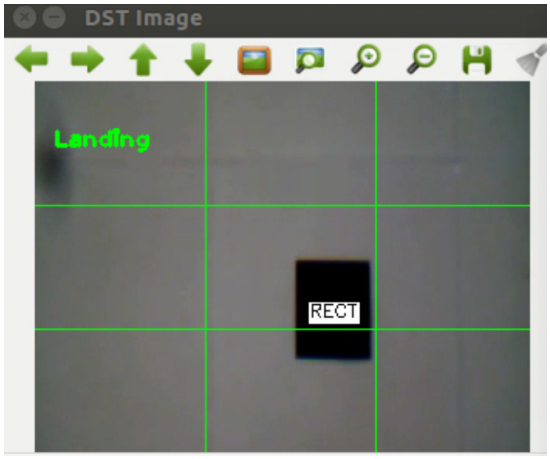
Pada Gambar 6.21 tidak terlihat label yang menandakan titik tengah objek pada *frame* kamera bawah *quadcopter* sehingga ketika tidak mendeteksi objek, maka *quadcopter* akan melakukan pergerakan *hover* secara otomatis.



Gambar 6.21 Pengujian gerakan hover

6. *Landing*

Pada Gambar 6.22 terlihat titik tengah objek berada pada *frame* kamera bawah *quadcopter* dan posisi objek ada pada kotak *grid* bagian tengah sehingga ketika terdeteksi, *quadcopter* akan melakukan *landing* secara otomatis.



Gambar 6.22 Pengujian *landing*

Setelah dilakukan percobaan untuk setiap navigasi sebanyak lima kali terhadap objek dengan kecepatan yang beragam yaitu 0,3 m/s, 0,4 m/s, 0,5 m/s, 0,6 m/s dan 0,7 m/s, didapatkan hasil yang dapat dilihat pada Tabel 6.1, Tabel 6.2, Tabel 6.3, Tabel 6.4 dan Hasil yang didapatkan dapat dilihat pada Tabel 6.5

Tanda *checklist* (√) merupakan simbol untuk pengujian yang berhasil, sedangkan tanda *cross* (×) merupakan simbol dari pengujian yang tidak berhasil. Untuk keseluruhan hasil dari gerakan otomatis pada *quacopter* dengan kecepatan 0,3 m/s telah mencapai keberhasilan tanpa ada kendala seperti yang terlihat pada Tabel 6.1.

Tabel 6.1 Hasil pengujian ketepatan gerakan pada kecepatan 0,3 m/s

Objek	Gerakan	Pengujian ke-				
		1	2	3	4	5
Segitiga	Gerak maju	√	√	√	√	√
	Gerak mundur	√	√	√	√	√
	Geser kanan	√	√	√	√	√
	Geser kiri	√	√	√	√	√
	Hover	√	√	√	√	√
	Landing	√	√	√	√	√
Kotak	Gerak maju	√	√	√	√	√
	Gerak mundur	√	√	√	√	√
	Geser kanan	√	√	√	√	√
	Geser kiri	√	√	√	√	√
	Hover	√	√	√	√	√
	Landing	√	√	√	√	√

Objek	Gerakan	Pengujian ke-				
		1	2	3	4	5
Segilima	Gerak maju	√	√	√	√	√
	Gerak mundur	√	√	√	√	√
	Geser kanan	√	√	√	√	√
	Geser kiri	√	√	√	√	√
	Hover	√	√	√	√	√
	Landing	√	√	√	√	√

Kemudian setelah melakukan kembali 5 percobaan yang sama dengan kecepatan 0,4 m/s, didapatkan hasil secara keseluruhan seperti pada Tabel 6.2. Dari percobaan tersebut, *quadcopter* dapat melakukan sebagian dari pergerakan otomatis hingga mencapai pendaratan.

Tabel 6.2 Hasil pengujian ketepatan gerakan pada kecepatan 0,4 m/s

Objek	Gerakan	Pengujian ke-				
		1	2	3	4	5
Segitiga	Gerak maju	√	x	√	√	√
	Gerak mundur	x	√	√	√	√
	Geser kanan	√	√	x	√	√
	Geser kiri	√	√	√	√	√
	Hover	√	√	√	√	√
	Landing	x	√	√	√	√
Kotak	Gerak maju	√	x	√	√	√
	Gerak mundur	x	√	√	√	√
	Geser kanan	√	√	√	√	√
	Geser kiri	√	√	√	√	√
	Hover	√	√	x	√	√
	Landing	√	x	√	√	√
Segilima	Gerak maju	√	√	√	x	√
	Gerak mundur	√	x	x	√	√
	Geser kanan	√	√	√	x	√
	Geser kiri	√	√	√	√	x

Objek	Gerakan	Pengujian ke-				
		1	2	3	4	5
	Hover	√	√	√	√	√
	Landing	√	x	√	√	√

Percobaan yang sama kembali dilakukan sebanyak 5 kali juga seperti sebelumnya dengan kecepatan 0,5 m/s. Hasil yang didapatkan seperti yang terlihat pada Tabel 6.3, *quadcopter* mengalami beberapa kegagalan pada gerakan yang diujikan dari mulai melakukan pendeteksian melalui pergerakan otomatis hingga mendarat tepat pada objek yang dituju.

Tabel 6.3 Pengujian ketepatan gerakan dengan kecepatan 0,5 m/s

Objek	Gerakan	Pengujian ke-				
		1	2	3	4	5
Segitiga	**Gerak maju**	√	√	√	x	√
	Gerak mundur	√	√	x	√	x
	Geser kanan	√	√	x	√	√
	Geser kiri	√	√	√	√	x
	Hover	x	√	√	√	√
	Landing	√	x	√	x	√
Kotak	**Gerak maju**	√	√	√	x	√
	Gerak mundur	√	x	x	√	√
	Geser kanan	√	x	√	√	√
	Geser kiri	√	√	√	x	√
	Hover	√	√	√	√	x
	Landing	x	√	√	√	√
Segilima	**Gerak maju**	x	√	x	√	√
	Gerak mundur	√	x	√	√	√
	Geser kanan	√	x	√	x	√
	Geser kiri	x	√	√	x	√
	Hover	√	√	x	√	√
	Landing	√	x	√	√	√

Pada kecepatan 0,6 m/s yang dilakukan sebanyak 5 kali percobaan pada setiap objek, *quadcopter* terbang dengan cukup cepat sehingga ada beberapa gerakan yang berhasil dilakukan untuk menuju ke arah objek dan ada juga gerakan yang tidak berhasil dilakukan selama pendeteksian objek berlangsung. Hasil yang didapatkan dapat dilihat pada Tabel 6.4.

Tabel 6.4 Pengujian ketepatan gerakan dengan kecepatan 0,6 m/s

Objek	Gerakan	Pengujian ke-				
		1	2	3	4	5
Segitiga	Gerak maju	x	√	√	√	x
	Gerak mundur	√	x	√	x	x
	Geser kanan	x	√	x	√	x
	Geser kiri	√	x	x	√	√
	Hover	√	√	x	x	√
	Landing	x	x	√	x	x
Kotak	Gerak maju	√	x	√	x	√
	Gerak mundur	x	√	√	x	√
	Geser kanan	√	x	√	√	x
	Geser kiri	x	√	x	x	√
	Hover	√	x	√	x	x
	Landing	x	√	x	x	√
Segilima	Gerak maju	x	√	x	√	x
	Gerak mundur	√	x	√	x	x
	Geser kanan	x	√	√	x	√
	Geser kiri	x	√	x	√	√
	Hover	√	x	√	√	x
	Landing	x	√	x	x	√

Pada kecepatan 0,7 m/s yang dilakukan sebanyak 5 kali percobaan pada setiap objek, *quadcopter* mulai mengalami peningkatan kecepatan yang signifikan dari percobaan sebelumnya. Pergerakan otomatis tidak ada yang berhasil dilakukan dibandingkan kecepatan pada percobaan-percobaan sebelumnya dan tidak bisa mendeteksi objek. Hasil yang didapatkan dapat dilihat pada Tabel 6.5

Tabel 6.5 Pengujian ketepatan gerakan pada kecepatan 0,7 m/s

Objek	Gerakan	Pengujian ke-				
		1	2	3	4	5
Segitiga	Gerak maju	x	x	x	x	x
	Gerak mundur	x	x	x	x	x
	Geser kanan	x	x	x	x	x
	Geser kiri	x	x	x	x	x
	Hover	x	x	x	x	x
	Landing	x	x	x	x	x
Kotak	Gerak maju	x	x	x	x	x
	Gerak mundur	x	x	x	x	x
	Geser kanan	x	x	x	x	x
	Geser kiri	x	x	x	x	x
	Hover	x	x	x	x	x
	Landing	x	x	x	x	x
Segilima	Gerak maju	x	x	x	x	x
	Gerak mundur	x	x	x	x	x
	Geser kanan	x	x	x	x	x
	Geser kiri	x	x	x	x	x
	Hover	x	x	x	x	x
	Landing	x	x	x	x	x

6.2.5 Analisis Hasil Pengujian

Berdasarkan pengujian yang dilakukan terkait ketepatan pergerakan pada *quadcopter* dengan kecepatan yang berbeda yaitu 0,3 m/s, 0,4 m/s, 0,5 m/s, 0,6 m/s dan 0,7 m/s didapatkan beberapa hasil. Pada kecepatan 0,3 m/s *quadcopter* berhasil melakukan seluruh pergerakan otomatis untuk keseluruhan *grid* pada *frame* kamera bawah *quadcopter* dengan benar untuk mendeteksi objek landasan untuk mendarat. Pada kecepatan 0,4 m/s dan 0,5 m/s mengalami kegagalan dalam pergerakan otomatis sedangkan dengan kecepatan 0,6 m/s *quadcopter* mengalami banyak kegagalan sehingga hanya dapat melakukan sebagian dari pergerakan otomatis dan pada kecepatan 0,7 m/s *quadcopter* tidak dapat mendeteksi objek ataupun bergerak secara otomatis karena *quadcopter* terlalu cepat dalam bernavigasi sehingga objek tidak terdeteksi dengan optimal dan objek tersebut dilewati. Maka, hasil persentase dari ketepatan gerakan yang

dilakukan pada sistem *quadcopter* dapat dihitung dengan rumus pada persamaan 6.1

$$Persentase\ ketepatan = \frac{jumlah\ gerakan\ benar}{jumlah\ pengujian}\ x\ 100\% \quad (6.1)$$

Pada kecepatan 0,3 m/s, *quadcopter* memperoleh hasil ketepatan gerakan dengan persentase keberhasilan 100%. Pergerakan otomatis yang berhasil dilakukan adalah gerakan untuk setiap *grid* pada *frame* kamera bawah saat mendeteksi objek. Hasil dari persentase ketepatan pergerakan otomatis tersebut dapat dilihat pada Tabel 6.6

Tabel 6.6 Analisis ketepatan gerakan dengan kecepatan 0,3 m/s

Objek	Gerakan	Jumlah pengujian	Jumlah gerakan benar	Persentase ketepatan
Kotak	Gerak maju	5	5	100%
	Gerak mundur	5	5	100%
	Geser kanan	5	5	100%
	Geser Kiri	5	5	100%
	Hover	5	5	100%
	Landing	5	5	100%
Segitiga	Gerak maju	5	5	100%
	Gerak mundur	5	5	100%
	Geser kanan	5	5	100%
	Geser Kiri	5	5	100%
	Hover	5	5	100%
	Landing	5	5	100%
Segilima	Gerak maju	5	5	100%
	Gerak mundur	5	5	100%
	Geser kanan	5	5	100%
	Geser Kiri	5	5	100%
Segilima	*Hover*	5	5	100%

Objek	Gerakan	Jumlah pengujian	Jumlah gerakan benar	Persentase ketepatan
	Landing	5	5	100%
RATA-RATA KETEPATAN GERAKAN				100%

Kemudian pada kecepatan 0,4 m/s, *quadcopter* memperoleh hasil ketepatan gerakan dengan persentase keberhasilan 85,5%. Pergerakan otomatis yang berhasil dilakukan adalah sebagian dari gerakan untuk setiap *grid* pada *frame* kamera bawah *quadcopter* saat mendeteksi objek. Hasil dari persentase ketepatan pergerakan otomatis tersebut dapat dilihat pada Tabel 6.7.

Tabel 6.7 Analisis ketepatan gerakan dengan kecepatan 0,4 m/s

Objek	Gerakan	Jumlah pengujian	Jumlah gerakan benar	Persentase ketepatan
Kotak	**Gerak maju**	5	4	80%
	Gerak mundur	5	4	80%
	Geser kanan	5	5	100%
	Geser Kiri	5	5	100%
	Hover	5	4	80%
	Landing	5	4	80%
Segitiga	**Gerak maju**	5	4	80%
	Gerak mundur	5	4	80%
	Geser kanan	5	4	80%
	Geser Kiri	5	5	100%
	Hover	5	5	100%
	Landing	5	4	80%
Segilima	**Gerak maju**	5	4	80%
	Gerak mundur	5	3	60%
	Geser kanan	5	4	80%

Objek	Gerakan	Jumlah pengujian	Jumlah gerakan benar	Persentase ketepatan
Segilima	Geser Kiri	5	4	80%
	Hover	5	5	100%
	Landing	5	4	80%
RATA-RATA KETEPATAN GERAKAN				84,45%

Pada kecepatan 0,5 m/s *quadcopter* memperoleh akurasi yang tidak sebesar pada kecepatan sebelumnya dengan persentase hasil ketepatan gerakan 73,3%. Pergerakan otomatis yang berhasil dilakukan hanya sebagian pada setiap *grid* pada *frame* kamera bawah saat mendeteksi objek. Hasil dari persentase ketepatan pergerakan otomatis tersebut dapat dilihat pada Tabel 6.8.

Tabel 6.8 Analisis ketepatan gerakan dengan kecepatan 0,5 m/s

Objek	Gerakan	Jumlah pengujian	Jumlah gerakan benar	Persentase ketepatan
Kotak	Gerak maju	5	4	80%
	Gerak mundur	5	3	60%
	Geser kanan	5	4	80%
	Geser Kiri	5	4	80%
	Hover	5	4	80%
	Landing	5	4	80%
Segitiga	Gerak maju	5	4	80%
	Gerak mundur	5	3	60%
	Geser kanan	5	4	80%
	Geser Kiri	5	4	80%
	Hover	5	4	80%
	Landing	5	3	60%
Segilima	Gerak maju	5	3	60%
	Gerak mundur	5	4	80%
	Geser kanan	5	3	60%

Objek	Gerakan	Jumlah pengujian	Jumlah gerakan benar	Persentase ketepatan
	Geser Kiri	5	3	60%
	Hover	5	4	80%
	Landing	5	4	80%
RATA-RATA KETEPATAN GERAKAN				73,3%

Pada kecepatan 0,6 m/s didapatkan hasil yang berbeda dengan kecepatan 0,3 m/s, 0,4 m/s, 0,5 m/s. Gerakan otomatis yang berhasil dilakukan hanya beberapa dengan persentase 48,8% seperti yang terlihat pada Tabel 6.8 . Hal ini karena *quadcopter cukup* cepat saat bernavigasi sehingga objek tidak dapat terdeteksi dengan baik saat pergerakan otomatis untuk seluruh *grid* berlangsung dan bahkan menyebabkan objek landasan dilewati oleh *quadcopter*. Persentase dari pergerakan otomatis yang dihasilkan tidak sebesar dengan kecepatan sebelumnya seperti yang terlihat pada Tabel 6.9.

Tabel 6.9 Analisis ketepatan gerakan dengan kecepatan 0,6 m/s

Objek	Gerakan	Jumlah pengujian	Jumlah gerakan benar	Persentase ketepatan
Segitiga	**Gerak maju**	5	3	60%
	Gerak mundur	5	3	60%
	Geser kanan	5	2	40%
	Geser Kiri	5	3	60%
	Hover	5	2	40%
	Landing	5	1	20%
Kotak	**Gerak maju**	5	3	60%
	Gerak mundur	5	3	60%
	Geser kanan	5	3	60%
	Geser Kiri	5	2	40%
	Hover	5	2	40%
	Landing	5	2	40%

Objek	Gerakan	Jumlah pengujian	Jumlah gerakan benar	Persentase ketepatan
Segilima	**Gerak maju**	5	2	40%
	Gerak mundur	5	2	40%
	Geser kanan	5	3	60%
	Geser Kiri	5	3	60%
	Hover	5	3	60%
	Landing	5	2	40%
RATA-RATA KETEPATAN GERAKAN				48,8%

Pada kecepatan 0,7 m/s seperti yang terdapat pada Tabel 6.10, *quadcopter* tidak optimal dalam melakukan pergerakan otomatis dan dalam mendeteksi objek. Tidak ada ketepatan gerakan yang berhasil pada kecepatan 0,7 m/s. Hal ini disebabkan *quadcopter* telah terbang dengan sangat cepat sehingga gerakan otomatis yang dihasilkan saat pendeteksian objek mengalami kendala hingga mengakibatkan *quadcopter* melewati objek. Persentase dari pergerakan otomatis yang dihasilkan dapat dilihat pada Tabel 6.10.

Tabel 6.10 Analisis ketepatan gerakan dengan kecepatan 0,7 m/s

Objek	Gerakan	Jumlah pengujian	Jumlah gerakan benar	Persentase ketepatan
Kotak	**Gerak maju**	5	0	0%
	Gerak mundur	5	0	0%
	Geser kanan	5	0	0%
	Geser Kiri	5	0	0%
	Hover	5	0	0%
	Landing	5	0	0%
Segitiga	**Gerak maju**	5	0	0%
	Gerak mundur	5	0	0%
	Geser kanan	5	0	0%

Objek	Gerakan	Jumlah pengujian	Jumlah gerakan benar	Persentase ketepatan
	Geser Kiri	5	0	0%
	Hover	5	0	0%
	Landing	5	0	0%
Segilima	**Gerak maju**	5	0	0%
	Gerak mundur	5	0	0%
	Geser kanan	5	0	0%
	Geser Kiri	5	0	0%
Segilima	***Hover***	5	0	0%
	Landing	5	0	0%
RATA-RATA KETEPATAN GERAKAN				0%

6.3 Pengujian *Delay* Pada Sistem

6.3.1 Tujuan Pengujian

Tujuan dari pengujian ini untuk mengetahui seberapa optimal kinerja sistem dalam mencapai pendaratan otomatis dengan melihat seberapa besar *delay* yang terjadi dari mulai mendeteksi dan bergerak hingga mencapai pendaratan otomatis.

6.3.2 Pelaksanaan Pengujian

Pelaksanaan dalam pengujian ini dilakukan dengan cara melihat selisih waktu yang akurat pada kondisi saat *quadcopter* mulai mendeteksi objek, melakukan pergerakan otomatis sesuai posisi objek pada *grid* hingga mendarat tepat pada objek landasan melalui pemantauan pada *frame* kamera bawah *quadcopter*.

6.3.3 Prosedur Pengujian

Pengujian akan dilakukan dengan prosedur sebagai berikut.

1. Memasang objek segitiga, kotak dan segi lima berwarna warna hitam yang telah dibuat.
2. Menyiapkan *quadcopter*, memasang baterai *quadcopter* dan menunggu hingga *quadcopter* terkalibrasi. Setelah terdengar suara *beep* sebanyak empat kali maka *quadcopter* siap digunakan.

3. Menghubungkan *PC* dengan *hotspot Wi-fi* yang ada pada *quadcopter.*
4. Membuka *terminal* pada *ubuntu* dan masuk ke direktori dengan menggetik "cd tum_simulator_ws", kemudian mengetik "source devel/setup.bash". setelah itu mengetik "roslaunch ardrone_autonomy ardrone.launch" untuk menghubungkan *ROS* dengan *quadcopter*.
5. Membuka *terminal* lagi pada *ubuntu* dan masuk ke direktori dengan menggetik "cd tum_simulator_ws", kemudian mengetik "rosservice call /ardrone/togglecam" untuk menggunakan kamera bawah *quadcopter*.
6. Membuka *terminal* lagi pada *ubuntu* dan masuk ke direktori dengan menggetik "cd tum_simulator_ws", kemudian mengetik "source devel/setup.bash". setelah itu mengetik "rosrun testopencv testopencv_node" untuk menjalankan program.
7. Mengatur jarak *quadcopter* dengan objek dan memposisikan sesuai dengan gerakan yang akan dilakukan.
8. Menjalankan perintah otomatis agar *quadcopter* dapat bergerak, mendeteksi dan mendarat pada objek.
9. Mengamati *output* pada *terminal* untuk mengetahui apakah gerakan *quadcopter* telah bisa mendeteksi objek dengan tepat dan mendarat tepat di tengah objek.

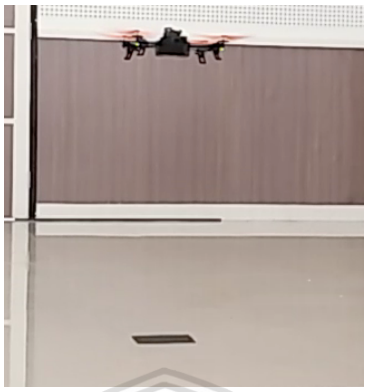
6.3.4 Hasil Pengujian

Setelah melakukan pengujian maka didapat hasil *delay* untuk masing-masing pergerakan *quadcopter* sebagai berikut.

1. Gerak Maju
   Sebelum melakukan *take off, quadcopter* diletakkan di belakang objek untuk melakukan gerak maju, ketika *quadcopter* telah melakukan *take off* maka *quadcopter* akan melakukan gerak maju melalui *input* pada terminal untuk mendekati objek. Jika *quadcopter* telah menemukan objek setelah melakukan gerak maju maka *quadcopter* akan melakukan pergerakan secara otomatis untuk mencari titik tengah dari objek yang terdeteksi sesuai dengan *grid* pada *frame* kamera bawah *quadcopter*. Jika *quadcopter* telah melakukan pergerakan otomatis maka secara otomatis juga akan melakukan pendaratan pada objek yang dijadikan landasan untuk mendarat. Posisi awal *quadcopter* dapat dilihat pada Gambar 6.23.

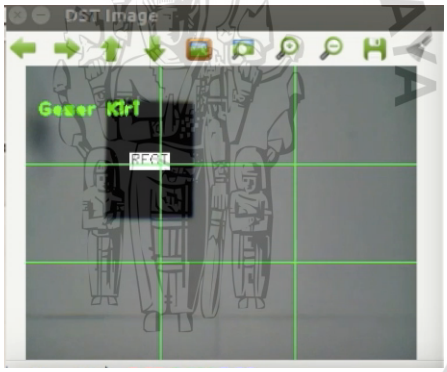
Gambar 6.23 Posisi awal *quadcopter* Gerakan Maju

Quadcopter melakukan pergerakan otomatis dalam satu jalur dengan memberikan input navigasi untuk melakukan gerakan maju agar dapat mendekati objek sampai dengan objek terdeteksi seperti pada Gambar 6.24.



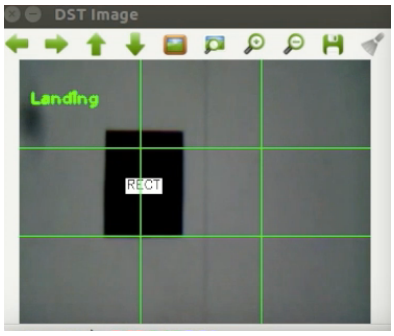
Gambar 6.24 Posisi *quadcopter* telah mendeteksi objek

Ketika *quadcopter* telah berhasil mendeteksi objek pada kamera bawah maka *quadcopter* melakukan pergerakan secara otomatis untuk mendekati titik tengah pada objek, pada pengujian ini objek berhasil terdeteksi lalu akan melakukan pergerakan otomatis geser ke kiri seperti pada Gambar 6.25.



Gambar 6.25 Pergerakan otomatis kamera bawah *quadcopter*

Setalah *quadcopter* berhasil melakukan pergerakan otomatis untuk mencari titik tengah objek, maka *quadcopter* akan melakukan pendaratan otomatis seperti pada Gambar 6.26.



Gambar 6.26 Pendaratan otomatis kamera bawah *quadcopter*

Pada pengujian gerakan maju *quadcopter* berhasil mendeteksi objek, melakukan pergerakan otomatis dan pendaratan otomatis dengan waktu sebanyak 1,68 detik, dengan melakukan pergerakan geser ke kiri selama 0,33 detik dan pendaratan otomatis selama 1,35 detik.

2. Gerak Mundur

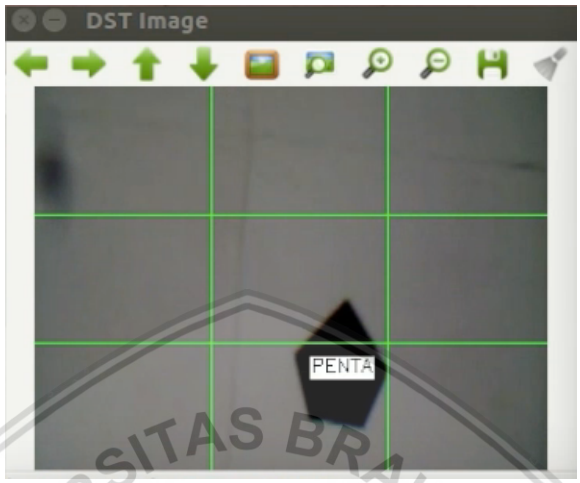
Sebelum melakukan *take off, quadcopter* diletakkan di depan objek untuk melakukan gerak mundur, ketika *quadcopter* telah melakukan *take off* maka *quadcopter* akan melakukan gerak mundur melalui *input* pada terminal untuk mendekati objek. Jika *quadcopter* telah menemukan objek setelah melakukan gerak mundur maka *quadcopter* akan melakukan pergerakan secara otomatis untuk mencari titik tengah dari objek yang terdeteksi sesuai dengan *grid* pada *frame* kamera bawah *quadcopter*. Jika *quadcopter* telah melakukan pergerakan otomatis maka secara otomatis juga akan melakukan pendaratan pada objek yang dijadikan landasan untuk mendarat. Posisi awal *quadcopter* dapat dilihat pada Gambar 6.27.

Gambar 6.27 Posisi awal *quadcopter* Gerakan Mundur

Quadcopter melakukan pergerakan otomatis dalam satu jalur dengan memberikan input navigasi untuk melakukan gerakan mundur agar dapat mendekati objek sampai dengan objek terdeteksi seperti pada Gambar 6.28.

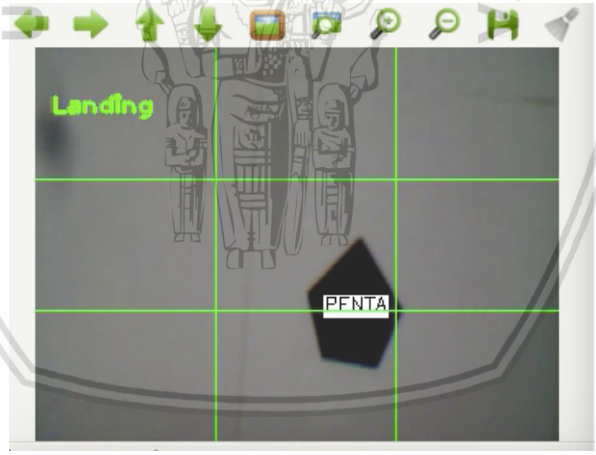
Gambar 6.28 Posisi *quadcopter* telah mendeteksi objek

Ketika *quadcopter* telah berhasil mendeteksi objek pada kamera bawah maka *quadcopter* melakukan pergerakan secara otomatis untuk mendekati titik tengah pada objek, pada pengujian ini objek berhasil terdeteksi lalu akan melakukan pergerakan otomatis mundur ke belakang seperti pada Gambar 6.25.



Gambar 6.29 Pergerakan otomatis kamera bawah *quadcopter*

Setalah *quadcopter* berhasil melakukan pergerakan otomatis untuk mencari titik tengah objek, maka *quadcopter* akan melakukan pendaratan otomatis seperti pada Gambar 6.26.



Gambar 6.30 Pendaratan otomatis kamera bawah *quadcopter*

Pada pengujian gerakan mundur *quadcopter* dapat berhasil mendeteksi objek, melakukan pergerakan otomatis dan pendaratan otomatis dengan waktu sebanyak 1,48 detik, dengan melakukan pergerakan mundur ke belakang selama 0,26 detik dan pendaratan otomatis selama 1,22 detik.

3. Gerak Geser Kanan

Sebelum melakukan *take off, quadcopter* diletakkan di sebelah kiri objek untuk melakukan pergerakan geser kanan, ketika *quadcopter* telah melakukan *take off* maka *quadcopter* akan melakukan pergerakan geser kanan melalui *input* pada terminal untuk mendekati objek. Jika *quadcopter*

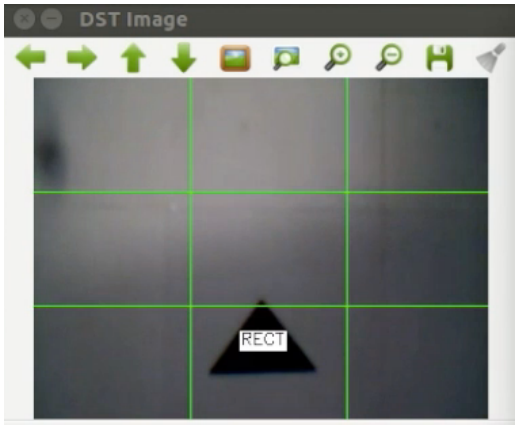
telah menemukan objek setelah melakukan pergerakan geser kanan maka *quadcopter* akan melakukan pergerakan secara otomatis untuk mencari titik tengah dari objek yang terdeteksi sesuai dengan *grid* pada *frame* kamera bawah *quadcopter*. Jika *quadcopter* telah melakukan pergerakan otomatis maka secara otomatis juga akan melakukan pendaratan pada objek yang dijadikan landasan untuk mendarat. Posisi awal *quadcopter* dapat dilihat pada Gambar 6.27.

Gambar 6.31 Posisi awal *quadcopter* Gerakan Geser Kanan

Quadcopter melakukan pergerakan otomatis dalam satu jalur dengan memberikan *input* navigasi untuk melakukan gerakan geser kanan untuk mendekati objek sampai dengan objek terdeteksi seperti pada Gambar 6.32.

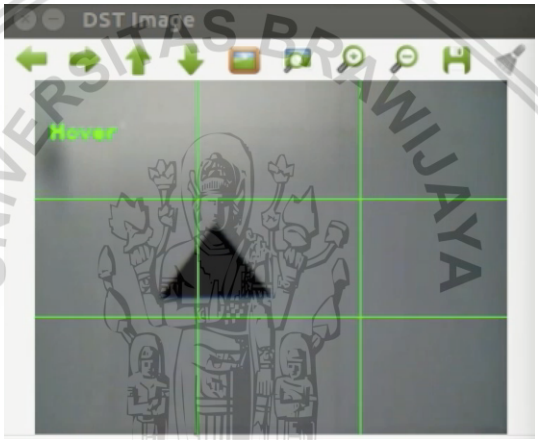
Gambar 6.32 Posisi *quadcopter* telah mendeteksi objek

Ketika *quadcopter* telah berhasil mendeteksi objek pada kamera bawah maka *quadcopter* melakukan pergerakan secara otomatis untuk mendekati titik tengah pada objek, pada pengujian ini objek berhasil terdeteksi lalu akan melakukan pergerakan otomatis mundur ke belakang seperti pada Gambar 6.33.



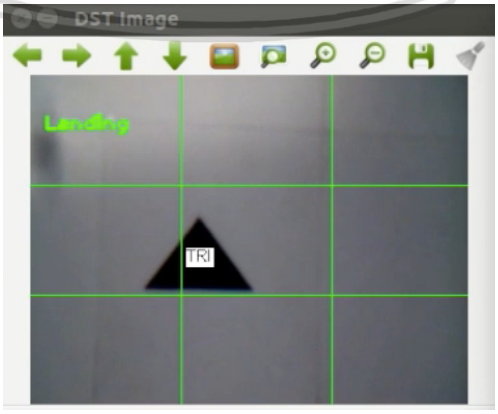
Gambar 6.33 Pergerakan otomatis kamera bawah *quadcopter*

Pada pergerakan otomatis pengujian ini *quadcopter* melakukan *hover* dikarenakan hasil kamera bawah *quadcopter* memiliki kualitas yang rendah untuk mendeteksi objek segitiga seperti pada Gambar 6.34.



Gambar 6.34 *Quadcopter* melakukan *hover* otomatis

Setalah *quadcopter* berhasil melakukan pergerakan otomatis untuk mencari titik tengah objek, maka *quadcopter* akan melakukan pendaratan otomatis seperti pada Gambar 6.35.



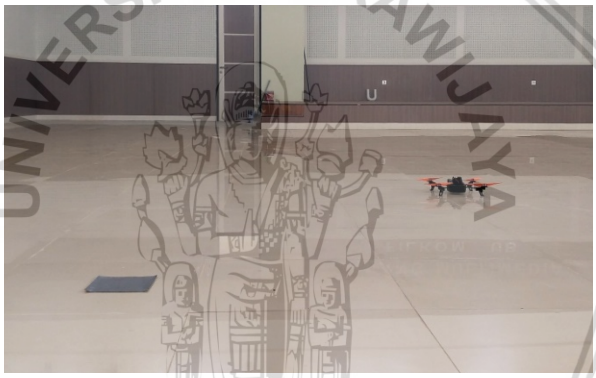
Gambar 6.35 Pendaratan otomatis kamera bawah *quadcopter*

Pada pengujian gerakan geser kanan *quadcopter* dapat berhasil mendeteksi objek, melakukan pergerakan otomatis dan pendaratan otomatis dengan

waktu sebanyak 7,6 detik, dengan melakukan pergerakan mundur ke belakang selama 0,28 detik, pergerakan hover selama 5,16 detik dan pendaratan otomatis selama 2,15 detik.

4. Gerak Geser Kiri

Sebelum melakukan *take off, quadcopter* diletakkan di sebelah kanan objek untuk melakukan pergerakan geser kiri, ketika *quadcopter* telah melakukan *take off* maka *quadcopter* akan melakukan pergerakan geser kiri melalui *input* pada terminal untuk mendekati objek. Jika *quadcopter* telah menemukan objek setelah melakukan pergerakan geser kiri maka *quadcopter* akan melakukan pergerakan secara otomatis untuk mencari titik tengah dari objek yang terdeteksi sesuai dengan *grid* pada *frame* kamera bawah *quadcopter*. Jika *quadcopter* telah melakukan pergerakan otomatis maka secara otomatis juga akan melakukan pendaratan pada objek yang dijadikan landasan untuk mendarat. Posisi awal *quadcopter* dapat dilihat pada Gambar 6.36.



Gambar 6.36 Posisi awal *quadcopter* Gerakan Geser Kiri

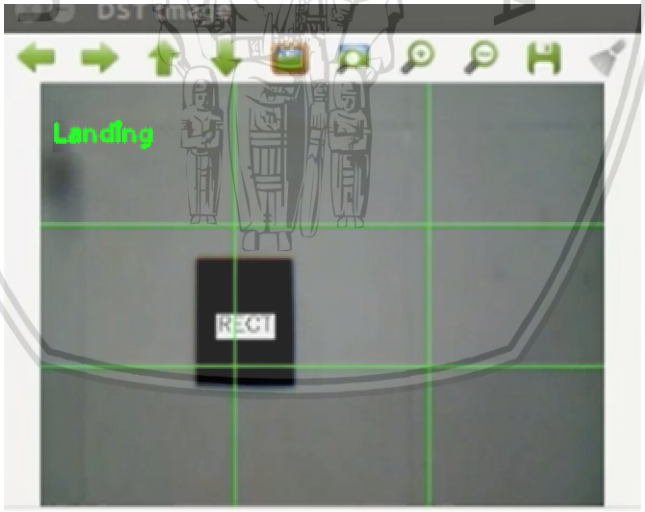
Quadcopter melakukan pergerakan otomatis dalam satu jalur dengan memberikan *input* navigasi untuk melakukan gerakan geser kiri untuk mendekati objek sampai dengan objek terdeteksi seperti pada Gambar 6.37.

Gambar 6.37 Posisi *quadcopter* telah mendeteksi objek

Ketika *quadcopter* telah berhasil mendeteksi objek pada kamera bawah maka *quadcopter* melakukan pergerakan secara otomatis untuk mendekati titik tengah pada objek, pada pengujian ini objek berhasil terdeteksi lalu akan melakukan pergerakan otomatis mundur ke belakang seperti pada Gambar 6.38.

Gambar 6.38 Pergerakan otomatis kamera bawah *quadcopter*

Setalah *quadcopter* berhasil melakukan pergerakan otomatis untuk mencari titik tengah objek, maka *quadcopter* akan melakukan pendaratan otomatis seperti pada Gambar 6.39.



Gambar 6.39 Pendaratan otomatis kamera bawah *quadcopter*

Pada pengujian gerakan geser kanan *quadcopter* dapat berhasil mendeteksi objek, melakukan pergerakan otomatis dan pendaratan otomatis dengan waktu sebanyak 2,23 detik, dengan melakukan pergerakan geser kiri selama 0,57 detik dan pendaratan otomatis selama 1,66 detik.

6.3.5 Analisis Hasil Pengujian

Setelah semua pengujian terhadap *delay* pendaratan otomatis yang dilakukan *quadcopter* pada objek segitiga yang terdeteksi dengan melakukan

berbagai gerakan untuk mendekati objek hingga dapat melakukan pendaratan secara otomatis, didapatkan hasil masing-masing *delay* seperti yang terlihat pada Tabel 6.11. Nilai rata-rata dari *delay* yang diperoleh adalah 3,52 detik berdasarkan perhitungan selisih waktu mulai dari *quadcopter* mendeteksi objek, melakukan gerakan otomatis sesuai *grid* hingga mendarat secara otomatis.

Tabel 6.11 Data *delay* pendaratan otomatis pada objek segitiga dengan ketinggian 125cm.

Objek	Gerakan	Hasil *Delay* (detik) Pengujian ke-					Rata-rata (detik)
		1	2	3	4	5	
Segitiga	Gerak Maju	1,53	6,86	1,89	1,54	6,42	3,64
	Gerak Mundur	1,35	2,64	1,52	1,36	1,9	1,75
	Geser Kiri	4,26	1,82	6,62	2,77	5,87	4,27
	Gerak Kanan	1,99	1,96	1,54	9,06	7,6	4,43
RATA-RATA DELAY							3,52

Setelah semua pengujian terhadap *delay* pendaratan otomatis yang dilakukan *quadcopter* pada objek kotak yang terdeteksi dengan melakukan berbagai gerakan untuk mendekati objek hingga dapat melakukan pendaratan secara otomatis, didapatkan hasil masing-masing *delay* seperti yang terlihat pada Tabel 6.12. Nilai rata-rata dari *delay* yang diperoleh adalah 2,53 detik berdasarkan perhitungan selisih waktu mulai dari *quadcopter* mendeteksi objek, melakukan gerakan otomatis sesuai *grid* hingga mendarat secara otomatis.

Tabel 6.12 Data *delay* pendaratan otomatis pada objek kotak dengan ketinggian 125cm.

Objek	Gerakan	Hasil *Delay* (detik) Pengujian ke-					Rata-rata (detik)
		1	2	3	4	5	
Kotak	Gerak Maju	1,74	1,57	1,68	1,84	2,03	1,77
	Gerak Mundur	1,62	1,26	1,38	1,83	1,98	1,61
	Geser Kiri	2,23	1,91	1,84	1,88	3,44	2,26
	Gerak Kanan	8,7	8,86	1,45	1,9	1,5	4,48
							2,53

Setelah semua pengujian terhadap *delay* pendaratan otomatis yang dilakukan *quadcopter* pada objek segilima yang terdeteksi dengan melakukan berbagai gerakan untuk mendekati objek hingga dapat melakukan pendaratan secara otomatis, didapatkan hasil masing-masing *delay* seperti yang terlihat pada Tabel 6.13. Nilai rata-rata dari *delay* yang diperoleh adalah 4,2 detik berdasarkan

perhitungan selisih waktu mulai dari *quadcopter* mendeteksi objek, melakukan gerakan otomatis sesuai *grid* hingga mendarat secara otomatis.

Tabel 6.13 Data *delay* pendaratan otomatis pada objek segilima dengan ketinggian 125cm.

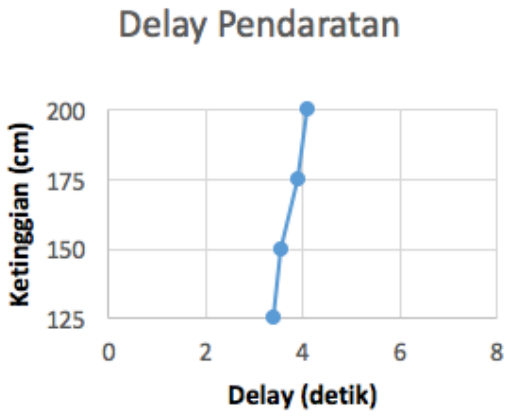
Objek	Gerakan	Hasil *Delay* (detik) Pengujian ke-					Rata-rata (detik)
		1	2	3	4	5	
Segilima	Gerak Maju	1,39	1,62	1,4	1,21	10,12	3,2
	Gerak Mundur	11,38	6,35	1,48	4,75	6,67	6,13
	Geser Kiri	9,75	2,07	1,56	12,86	1,44	5,54
	Gerak Kanan	1,7	3	1,54	1,77	1,7	1,94
RATA-RATA DELAY							4,2

Berdasarkan data *delay* yang tertera pada tabel seluruh objek landasan pendaratan *quadcopter* dengan menggunakan ketinggian dan objek yang berbeda, didapatkan hasil rata-rata waktu dengan ketinggian berbeda untuk setiap pergerakan hingga mencapai mendarat dalam bentuk grafik sebagai acuan untuk menganalisis efisiensi selama pendaratan berlangsung. Berikut ini adalah tampilan data *delay* dalam bentuk grafik untuk keseluruhan gerakan dengan ketinggian yang berbeda pada semua objek pendaratan seperti yang terlihat pada Tabel 6.14.

Tabel 6.14 Rata-rata *delay* dengan ketinggian yang berbeda pada objek pendaratan

Ketinggian (cm)	Delay pada Objek			Rata-rata (detik)
	Segitiga	Kotak	Segilima	
125	3,52	2,53	4,2	3,42
150	3,85	2,6	4,7	3,72
175	4,09	2,59	5,05	3,91
200	4,08	2,7	5,17	3,98

Sesuai pada data yang terdapat pada Tabel 6.14, terihat bahwa rata-rata keseluruhan *delay* pada semua objek landasan untuk mendarat menghasilkan nilai yang berbeda pada setiap ketinggian. Pada ketinggian 125 cm didapatkan rata-rata *delay* 3,42 detik, ketinggian 150 cm dengan nilai *delay* 3,72 detik, ketinggian 175 cm dengan nilai *delay* 3,91 detik dan ketinggian 200 cm dengan nilai *delay* 3,98 detik. Hasil *delay* tersebut membuktikan bahwa semakin besar nilai ketinggian maka waktu yang dibutuhkan untuk mencapai pendaratan otomatis juga semakin besar seperti yang terlihat pada Gambar 6.40.



Gambar 6.40 Grafik rata-rata *delay* dengan ketinggian yang berbeda pada objek pendaratan

6.4 Pengujian Ketepatan *Landing*

6.4.1 Tujuan Pengujian

Tujuan dilakukannya pengujian ini adalah untuk mengetahui seberapa tepat pendaratan secara otomatis pada *quadcopter* untuk tiga jenis objek yaitu segitiga, kotak dan segilima.

6.4.2 Pelaksanaan Pengujian

Pengujian yang dilakukan dengan cara menerbangkan *quadcopter* dan memonitoring melalui kamera bawah yang ditampilkan pada *frame* untuk melihat pergerakan otomatis dan pendeteksian objek untuk mengetahui ketepatan *quadcopter* saat akan melakukan pendaratan secara otomatis.

6.4.3 Prosedur Pengujian

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

1. Memasang objek berwarna hitam dengan bentuk segitiga, kotak dan segi lima yang dibuat sebelumnya.
2. Menyiapkan *quadcopter*, memasang baterai *quadcopter* dan menunggu hingga *quadcopter* terkalibrasi. Setelah terdengar suara *beep* sebanyak empat kali maka *quadcopter* siap digunakan.
3. Menghubungkan *PC* dengan *hotspot Wi-fi* yang ada pada *quadcopter.*
4. Membuka *terminal* pada *ubuntu* dan masuk ke direktori dengan menggetik “cd tum_simulator_ws”, kemudian mengetik “source devel/setup.bash”. setelah itu mengetik “roslaunch ardrone_autonomy ardrone.launch” untuk menghubungkan *ROS* dengan *quadcopter*.
5. Membuka *terminal* baru lagi pada *ubuntu* dan masuk ke direktori dengan menggetik “cd tum_simulator_ws”. Langkah selanjutnya mengetik

"rosservice call /ardrone/togglecam" untuk menggunakan kamera bawah *quadcopter*.

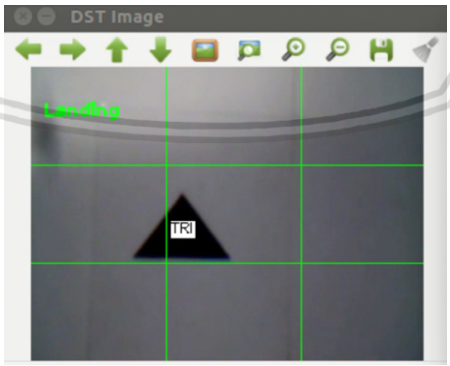
6. Membuka *terminal* lagi pada *ubuntu* dan masuk ke direktori dengan menggetik "cd tum_simulator_ws", kemudian mengetik "source devel/setup.bash". Kemudian mengetik "rosrun testopencv_node" untuk melakukan *run* pada program.
7. Mengatur jarak *quadcopter* dengan objek dan memposisikan sesuai dengan gerakan yang akan dilakukan.
8. Menjalankan pergerakan *quadcopter* dengan perintah otomatis sesuai dengan arah yang akan dituju seperti navigasi gerakan maju, gerakan mundur, gerakan ke kiri, dan gerakan ke kanan.
9. Mengamati *output* dari kamera *quadcopter* pada terminal untuk mengetahui apakah objek dapat terdeteksi, melakukan pergerakan otomatis untuk mencari titik tengah pada objek dan mendarat secara otomatis.

6.4.4 Hasil Pengujian

Setelah semua pengujian dilakukan maka didapatkan beberapa kondisi ketepatan saat *quadcopter* akan mendarat secara otomatis. Berikut ini akan dijelaskan hasil pendaratan yang diperoleh dengan tiga objek diantaranya segitiga, kotak dan segilima.

1. Objek Segitiga

   Pada pendeteksian objek segitiga, *quadcopter* dapat melakukan pergerakan secara otomatis untuk mencari titik tengah pada objek hingga dapat mendarat secara otomatis. Pada Gambar 6.41 terlihat bahwa objek yang terdeteksi pada *frame* kamera bawah *quadcopter* berada di posisi *grid* tengah.



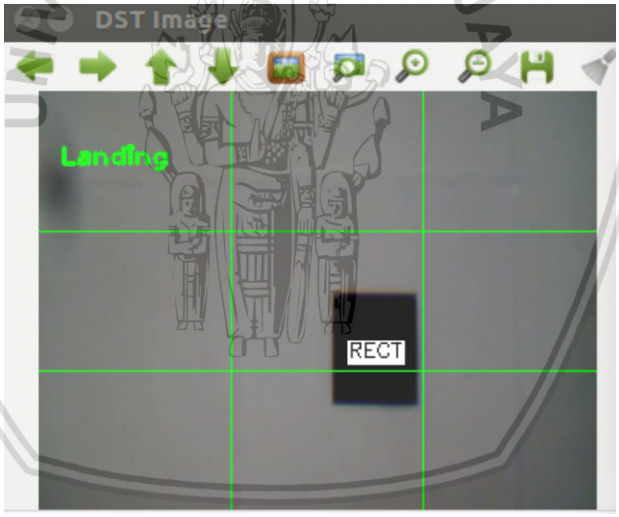
Gambar 6.41 Ketepatan mendarat pada objek segitiga

Setelah objek berada pada *grid* tengah maka *quadcopter* berhasil mendarat secara otomatis tepat pada objek segitiga dan hasil dari posisi ketepatan pendaratan tersebut dapat dilihat pada Gambar 6.42.

Gambar 6.42 Hasil pendaratan otomatis pada objek Segitiga

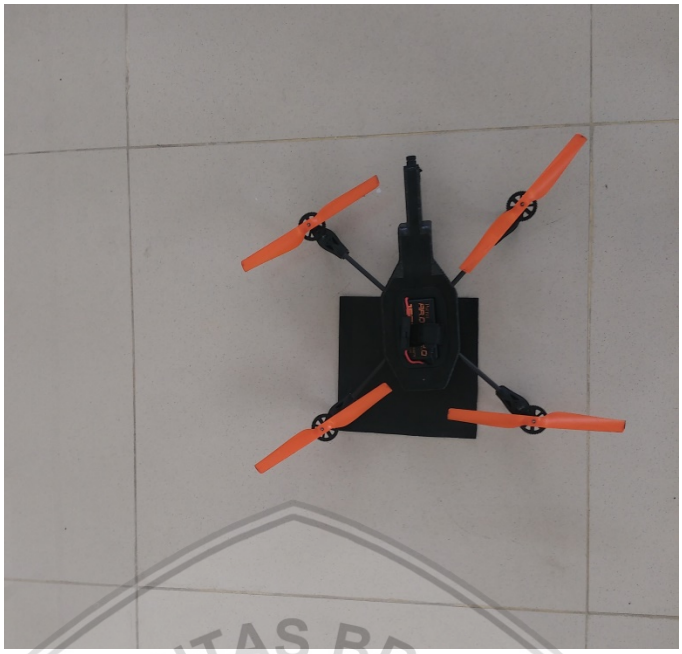
2. Objek Kotak

Pada pendeteksian objek kotak, *quadcopter* dapat melakukan pergerakan secara otomatis untuk mencari titik tengah pada objek hingga dapat mendarat secara otomatis. Pada Gambar 6.43 terlihat bahwa objek yang terdeteksi pada *frame* kamera bawah *quadcopter* berada di posisi *grid* tengah.



Gambar 6.43 Ketepatan mendarat pada objek kotak

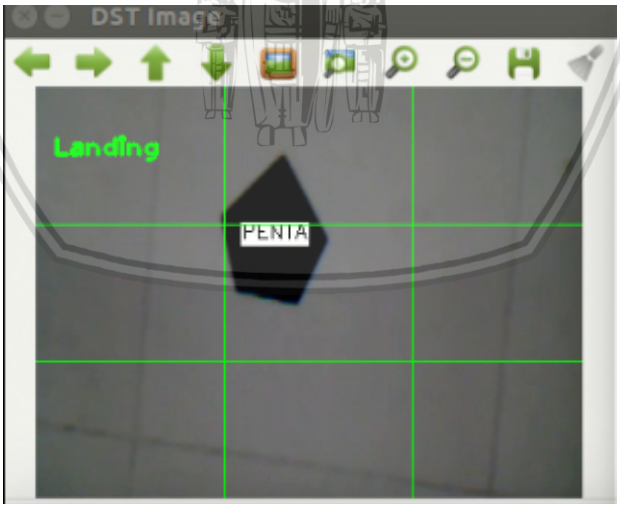
Setelah objek berada pada *grid* tengah maka *quadcopter* berhasil mendarat secara otomatis tepat pada objek kotak dan hasil dari posisi ketepatan pendaratan tersebut dapat dilihat pada Gambar 6.44.



Gambar 6.44 Hasil pendaratan otomatis pada objek kotak

3. Objek Segilima

Pada pendeteksian objek segilima, *quadcopter* dapat melakukan pergerakan secara otomatis untuk mencari titik tengah pada objek hingga dapat mendarat secara otomatis. Pada Gambar 6.45 terlihat bahwa objek yang terdeteksi pada *frame* kamera bawah *quadcopter* berada di posisi *grid* tengah.



Gambar 6.45 Ketepatan mendarat pada objek segilima

Setelah objek berada pada *grid* tengah maka *quadcopter* berhasil mendarat secara otomatis tepat pada objek segilima dan hasil dari posisi ketepatan pendaratan tersebut dapat dilihat pada Gambar 6.46.

Gambar 6.46 Hasil pendaratan otomatis pada objek segilima

6.4.5 Analisis Hasil Pengujian

Setelah pengujian terhadap ketepatan mendarat dilakukan sebanyak sepuluh kali percobaan untuk masing-masing objek landasan, hasil yang didapatkan seperti yang terlihat pada Tabel 6.14 Keberhasilan pengujian diberi tanda *checklist* (√) sedangkan pengujian yang tidak berhasil diberi tanda *cross* (x). Berdasarkan hasil tersebut terlihat bahwa *quadcopter* dapat mendeteksi objek landasan dengan melakukan pergerakan otomatis hingga mencapai pendaratan saat objek berada pada area *grid* tengah pada *frame* kamera bawah.

Tabel 6.15 Analisis hasil pengujian ketepatan pendaratan dengan ketinggian 125cm.

No	Objek	Pengujian ke-										Persentase Ketepatan Pendaratan
		1	2	3	4	5	6	7	8	9	10	
1	Segitiga	√	√	√	√	√	√	√	√	x	√	90%
2	Kotak	√	√	√	√	√	x	√	√	√	x	80%
3	Segilima	√	√	√	√	√	√	√	x	x	√	80%
RATA-RATA KETEPATAN PENDARATAN												83,3%

Pada pengujian ketepatan pendaratan otomatis pada Tabel 6.15 didapatkan hasil ketepatan pendaratan yang dapat mendarat secara tepat terhadap 3 objek dengan rata-rata persetanse senilai 83,3% dengan melakukan 10 kali percobaan untuk setiap objek dengan persentase tingkat keberhasilan pendaratan pada objek segitiga sebesar 90% dengan keberhasilan 9 kali dapat mendarat tepat di *area* tengah objek segitiga, objek kotak sebesar 80% dengan keberhasilan 8 kali dapat mendarat tepat di *area* tengah objek kotak dan objek segilima sebesar 80% dengan keberhasilan 8 kali dapat mendarat tepat di *area* tengah objek segilima.

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan berbagai analisis dari hasil yang diperoleh melalui pengujian yang dilakukan pada penelitian ini, maka didapatkan beberapa kesimpulan dari rumusan masalah yang ditentukan sebelumnya seperti yang telah dijabarkan sebagai berikut.

1. Untuk dapat melakukan pengenalan objek menggunakan metode *douglas peucker* langkah awal yang harus dilakukan adalah mengkonversi RGB ke *grayscale* agar objek yang berwarna tidak sulit terdeteksi yang dilanjutkan dengan proses menghilangkan *noise* menggunakan *blur*. Setelah itu deteksi tepian objek menggunakan *canny edge detection* yang dapat meminimalkan adanya kerancuan pada tepi objek yang akan dideteksi. Tahap akhir pada pendeteksian objek adalah pencarian nilai kontur yang sesuai pada objek yang akan dideteksi sehingga *quadcopter* bisa mengenali masing-masing bentuk objek dari tepian yang didapatkan serta dibantu dengan penghitungan sisi objek menggunakan metode *douglas peucker*.
2. Dalam pengujian ketepatan pergerakan otomatis pada *quadcopter* dengan kecepatan yang berbeda-beda, didapatkan hasil akurasi dengan beberapa persentase yaitu 100% pada kecepatan 0,3 m/s, kecepatan 0,4 m/s sebesar 85,5%, kecepatan 0,5 m/s sebesar 73,3%, kecepatan 0,6 m/s sebesar 48,8% dan kecepatan 0,7 m/s sebesar 0%. Sehingga kecepatan yang memiliki akurasi yang baik dalam mendeteksi objek yaitu 0,3m/s.
3. Agar *quadcopter* bisa mencapai titik tengah pada objek, maka pada penelitian ini membuat sebuah *grid* yang terbagi menjadi 9 dengan ukuran *frame* kamera bawah sebesar 330x240 *pixel*. Hal ini bertujuan agar *quadcopter* dapat bergerak secara otomatis sesuai dengan posisi *grid* berdasarkan objek yang terdeteksi. Jika objek berada di 3 *grid* paling kiri atau pada koordinat sumbu x area 0-110 dan sumbu y dengan area 0-240 maka *quadcopter* akan melakukan pergerakan geser kiri, jika objek berada di 3 *grid* paling kanan atau pada koordinat sumbu x area 221-330 dan sumbu y dengan area 0-240 maka *quadcopter* akan melakukan pergerakan geser kanan, jika objek berada di *grid* atas bagian tengah atau pada koordinat sumbu x area 111-220 dan sumbu y dengan *area* 0-80 *quadcopter* akan bergerak maju, jika objek berada di *grid* bawah bagian tengah atau pada koordinat sumbu x *area* 111-220 dan sumbu y dengan *area* 161-240 *quadcopter* akan bergerak mundur, jika objek tidak terdeteksi maka *quadcopter* akan melakukan gerakan *hover* dan jika objek berada tepat pada *grid* bagian tengah atau pada koordinat sumbu x *area* 111-220 dan sumbu y dengan *area* 81-160 maka *quadcopter* akan melakukan pendaratan otomatis.
4. Pada pengujian ketepatan pendaratan otomatis didapatkan hasil persentase terbaik dari rata-rata ketepatan pendaratan untuk 3 objek yaitu 83,3% pada

ketinggian 125cm dengan melakukan 10 kali percobaan untuk setiap objek. Masing-masing persentase tingkat keberhasilan pendaratan yang didapatkan yaitu 90% pada objek segitiga serta 80% pada objek kotak dan segilima dengan rata-rata *delay* terbaik senilai 3,42 detik.

7.2 Saran

Agar sistem dapat dikembangkan lebih lanjut dan dengan performa yang lebih baik lagi, maka diperlukan saran sebagai berikut.

1. Mengembangkan variasi dari gerakan *quadcopter* seperti gerakan spiral, zigzag atau gerakan unik lainnya untuk melakukan pencarian objek secara otomatis.
2. Sistem dapat dikembangkan dengan membuat *quadcopter* bisa mendeteksi objek dengan berbagai macam warna dan bisa mendeteksi warna secara khusus.
3. Diharapkan pada penelitian selanjutnya *quadcopter* dapat mendeteksi berbagai macam objek selain objek segitiga, kotak dan segilima.

DAFTAR PUSTAKA

Aprilian, E., 2017. Pengembangan Sistem Pendaratan Otomatis Pada Pesawat Tanpa Awak. Institut Teknologi Sepuluh Nopember.

Dang, T, C., Pham, T, H., Pham, B, T., Truong, V, N., 2013. *Vision Based Ground Object Tracking Using AR.Drone Quadrotor*. ICCAIS Main Track, pp. 146-150.

Gadda, S, J., Patil, D, R., 2013. *Quadcopter (Uavs) For Border Security With Gui System. International Journal of Research in Engineering and Technology Volume: 02, Issue: 12,* Desember 2013.

Gaol, L, C, A., 2017. Pendaratan Otomatis Quadcopter AR Drone Menggunakan Metode Linear Quadratic Regulator (LQR). Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer *Volume* 01, No.10, Oktober 2017 : 1028-1035.

Ginkel, V, R., Meerman, I., Mulder, T., 2013. *Autonomous Landing* of a *Quadcopter on a Predefined Marker*. Universiteit Van Amsterdam, Amsterdam, Netherlands.

Gonzalez, R. C., Woods, R.E., 2002. *Digital Image Processing second edition*, New Jersey: Prentice-Hall, Inc.

Hanafi, N., Rameli, M., Ak, R, E. Pendaratan Otomatis *Quadcopter* Pada Platform Yang Bergerak Menggunakan *Neuro Fuzzy. JAVA Journal of Electrical and Electronics Engineering Volume* 12, No.1, April 2014.

Hamdani, N, C., A.K., E, R., Iskandar, E., 2013. Perancangan *Autonomous Landing* pada *Quadcopter* Menggunakan *Behavior-Based Intelligent Fuzzy Control. JURNAL TEKNIK POMITS Volume 2,* No. 2, 2013.

Lusiana, V., 2013. Deteksi Tepi pada Citra Digital menggunakan Metode Kirsch dan Robinson. Jurnal Teknologi informasi DINAMIK *Volume* 18, No.2, Juli 2013 : 182-189.

Murinto, M., 2009. Analisis Perbandingan Metode *Intensity Filtering* Dengan Metode *Frequency Filtering* Sebagai Reduksi Noise Citra Digital.(Online). http://journal.uii.ac.id/index.php/Snati/article/view/1695/1477 Diakses 15 September 2011.

Prasetyo, E., 2011. Pengolahan Citra Digital dan Aplikasinya menggunakan Matlab. Yogyakarta: Penerbit Andi.

ROS. 2016. About ROS. (Online) http://www.ros.org/about-ros/ [Diakses pada 15 Februari 2018].

OpenCV. 2011. Opencv *documentation*. (Online) https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/*canny*_detector/*canny*_detector.html [Diakses pada 17 Februari 2018].

SA, P. (2016). *Parrot Official Website*. Diambil kembali dari Parrot AR Drone 2.0 Elite Edition: https://www.parrot.com/us/drones/parrot-ardrone-20-elite-edition#parrot-ardrone-20-elite-edition [Diakses pada 5 Februari 2018].

Shanti, N, C., 2011. Mengibah Citra Berwarna Menjadi Gray-Scale dan Citra Biner. Jurnal Teknologi informasi DINAMIK Volume 16, No.1, Januari 2011 : 14-19.

Stefano. 2016. Algoritma *Douglas Peucker*. (Online) https://piptools.net/algoritma-douglas-peucker/ [Diakses pada 13 Februari 2018].

Swedia, E. R., Cahyani, M., 2010. Algoritma Transformasi Ruang Warna. Depok: Indie Publishing.

Winarno, E., 2011. Aplikasi Deteksi Tepi pada Realtime Video Menggunakan Algoritma Canny Detection. Teknologi Informasi DINAMIK, 44-49.