PENGGUNAAN *FINITE STATE MACHINE* UNTUK NAVIGASI DAN KENDALI *AR.DRONE QUADCOPTER* MENGGUNAKAN *MICROSOFT KINECT*

SKRIPSI

KEMINATAN TEKNIK KOMPUTER

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Reza Tanjung Ahmad Fauzi
NIM: 135150301111028

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

PENGESAHAN

PENGGUNAAN *FINITE STATE MACHINE* UNTUK NAVIGASI DAN KENDALI *AR.DRONE QUADCOPTER* MENGGUNAKAN *MICROSOFT KINECT*

SKRIPSI

KEMINATAN TEKNIK KOMPUTER

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Reza Tanjung Ahmad Fauzi
NIM: 135150301111028

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

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

Mengetahui
Ketua Jurusan Teknik Informatika

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

PERNYATAAN ORISINALITAS

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

Apabila ternyata di dalam 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, 01 Agustus 2018

Reza Tanjung Ahmad Fauzi

NIM: 135150301111028

KATA PENGANTAR

Puji syukur kehadirat Allah SWT yang telah melimpahkan rahmat, taufik, serta hidayah-Nya, sehingga penulis dapat menyelesaikan laporan skripsi dengan judul "Penggunaan *Finite State Machine* untuk Navigasi dan Kendali *AR.Drone Quadcopter* menggunakan *Microsoft Kinect*". Penyusunan laporan skripsi ini bertujuan untuk memenuhi persyaratan kelulusan dalam menempuh ujian Sarjana pada Fakultas Ilmu Komputer, Universitas Brawijaya Malang.

Penulis menyadari bahwa dalam penelitian dan penulisan laporan skripsi ini tidak akan terwujud tanpa adanya dorongan semangat dan bantuan dari berbagai pihak yang ikut membantu. Maka dari itu, peneliti ingin menyampaikan ucapan terima kasih dan rasa hormat kepada:

1. Allah SWT yang telah memberikan rahmat, taufik serta hidayah-Nya sehingga laporan skripsi dapat terselesaikan dengan baik.
2. Kedua orang tua serta keluarga penulis yang ikut serta membantu dengan memberi dorongan semangat kepada penulis.
3. Bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D. selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya.
4. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D. selaku Ketua Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya Malang.
5. Bapak Sabriansyah Rizqika Akbar, S.T, M.Eng. selaku Ketua Program Studi Teknik Komputer Fakultas Ilmu Komputer Universitas Brawijaya Malang.
6. Bapak Gembong Edhi Setyawan, S.T, M.T. selaku Dosen pembimbing satu yang telah memberikan saran serta penjelasan dalam menyelesaikan skripsi.
7. Bapak Wijaya Kurniawan, S.T, M.T. selaku Dosen pembimbing dua yang telah memberikan saran dan penjelasan untuk menyelesaikan laporan penulis.
8. Teman-teman Teknik Komputer Angkatan 2013 yang telah membantu akan terselesaikannya laporan skripsi ini yang tidak dapat diucapkan satu persatu.
9. Grup *Quadcopter* yang tidak dapat disebutkan namanya satu persatu yang telah memberikan dukungan dan do'a.
10. Seluruh pihak yang tidak dapat diucapkan satu persatu, peneliti mengucapkan banyak terima kasih atas segala bentuk dukungan dan doa sehingga laporan skripsi ini dapat terselesaikan.

Peneliti menyadari bahwa tulisan ini masih jauh dari kata sempurna dan masih memiliki berbagai macam kekurangan. Oleh karena itu penulis mengharapkan kritik dan saran yang membangun, agar ke depannya penulis dapat menjadi lebih

baik lagi. Semoga isi Laporan Skripsi ini dapat memberi manfaat bagi perkembangan ilmu pengetahuan di kemudian hari.

Malang, 01 Agustus 2018

Reza Tanjung Ahmad Fauzi
rezakiller00@gmail.com

ABSTRAK

Sistem berbasis komputer yaitu robot telah menempati kedudukan yang meningkat di dalam kehidupan sehari-hari atau kehidupan seorang profesional. Peningkatan kapabilitas pada robot menyebabkan semakin bertambahnya tugas-tugas yang dikerjakan oleh robot di dalam kehidupan manusia dan bahkan robot-robot tersebut melakukan kolaborasi dengan manusia. Oleh sebab itu diperlukan adanya penjaminan bahwa robot-robot tersebut tidak melukai dirinya sendiri atau lingkungan disekitar mereka. Aspek yang paling penting dari robot-robot tersebut adalah keamanan mereka ketika sedang melakukan tugas pada lingkungan manusia. Perhatian pada keamanan telah dipertimbangkan dengan menghubungkan beberapa aspek pada *robotic embedded system* seperti penghindaran pada tabrakan, kegiatan interaksi yang aman dengan manusia, dan pendeteksi kesalahan. Untuk menghidari hal-hal berbahaya yang dapat ditimbulkan oleh robot-robot, maka pada penelitian ini digunakan sebuah model *abstract machine* yaitu *finite state machine*. Pada penelitian ini dilakukan implementasi dari sistem kendali navigasi robot berjenis *quadcopter* atau lebih umum dikenal sebagai *drone* menggunakan model *finite state machine* menggunakan *gesture* tubuh dari pengguna. Gerakan *gesture* tubuh dari pengguna dideteksi menggunakan sensor kamera *kinect*. Berdasarkan pada pengujian kombinasi gerakan yang dilakukan sebanyak empat belas kali dengan menggunakan metode *finite state machine* pada sistem ini, dipeloreh hasil pergerakan *quadcopter* yang dimana tidak terjadi tabrakan terhadap objek yang berada pada sekitar *quadcopter*. Selain itu didapatkan hasil nilai persentase ketepatan gerakan yang dilakukan oleh pengguna dalam pengendalian *quadcopter* sebesar 100 %. *Delay* yang dihasilkan sistem ini pada saat pengguna melakukan *pergerakan tubuh* hingga *quadcopter* melakukan pergerakkan sebesar 0,75 detik.

Kata kunci: *Finite state machine, Kinect, Navigasi, Quadcopter*

ABSTRACT

Computer-based systems that are robots have occupied an increasing position in everyday life or the life of a professional. Increased capabilities in robots cause the increasing number of tasks performed by robots in human life and even those robots collaborate with humans. Therefore it is necessary to guarantee that the robots do not hurt themselves or the environment around them. The most important aspect of these robots is their security while performing tasks in the human environment. Attention to security has been considered by linking several aspects of robotic embedded systems such as collision avoidance, safe human interaction, and error detection. To avoid dangerous things that can be caused by robots, then in this research used a model of abstract machine that is finite state machine. In this research, we perform implementation of quadcopter type robot navigation control system or more commonly known as drone using finite state machine model using gesture body from a user. The gesture movement of the body of the user is detected using a Kinect camera sensor. Based on the combination of motion testing that was carried out fourteen times using the finite state machine method on this system, the results of quadcopter movement were found which did not occur in collisions of objects around the quadcopter. In addition, the result of the percentage of movement accuracy performed by users in quadcopter control is 100%. Delay generated this system at the time of the user to move the body until the quadcopter to move is 0.75 seconds.

Keywords: *Finite state machine, Kinect, Navigation, Quadcopter*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar belakang

Sistem berbasis komputer yaitu robot telah menempati kedudukan yang meningkat di dalam kehidupan sehari-hari atau kehidupan seorang profesional. Peningkatan kapabilitas pada robot menyebabkan semakin bertambahnya tugas-tugas yang dikerjakan oleh robot di dalam kehidupan manusia dan bahkan robot-robot tersebut melakukan kolaborasi dengan manusia. Oleh sebab itu diperlukan adanya penjaminan bahwa robot-robot tersebut tidak melukai dirinya sendiri atau lingkungan disekitar mereka. Aspek yang paling penting dari robot-robot tersebut adalah keamanan mereka ketika sedang melakukan tugas pada lingkungan manusia (Kugler & Holzapfel, 2016).

Perhatian pada keamanan telah dipertimbangkan dengan menghubungkan beberapa aspek pada *robotic embedded system* seperti penghindaran pada tabrakan, kegiatan interaksi yang aman dengan manusia, dan pendeteksi kesalahan. Untuk menghidari hal-hal berbahaya yang dapat ditimbulkan oleh robot-robot, maka digunakan sebuah model komputasi matematika yaitu *finite state machine*. *Finite state machine* digunakan didalam model tingkah laku sistem yang banyak diaplikasian dibidang *engineering* dan *scientific*. *State* pada sebuah sistem didefinisikan sebagai sebuah kondisi sistem yang sedang terjadi pada waktu tertentu dalam satu waktu. *Finite state machine* merupakan sebuah sistem yang keluarannya tidak hanya tergantung pada masukan yang sedang terjadi, tetapi juga tergantung pada *state* yang sedang terjadi pada sistem tersebut. *State* pada sebuah sistem merupakan ringkasan dari keseluruhan sistem yang dibutuhkan untuk mengetahui tentang masukan sebelumnya dalam rangka untuk menghasilkan keluaran (Mihn, et al., 2017).

Salah satu robot yang dapat mengimplementasikan model *finite state machine* adalah *quadcopter*. Menurut Setyawan, Setiawan, dan Kurniawan (2015) *quadcopter* merupakan salah satu jenis dari *Unmanned Aerial Vehicle* yang memiliki empat lengan dengan baling – baling disetiap ujungnya. Pada umumnya untuk menerbangkan *quadcopter* dibutuhkan sebuah *remote control* dengan pemancar frekuensi radio atau *Wi-Fi*. Pada *quadcopter* dua baling-baling berputar searah jarum jam dan dua baling-baling lainnya akan berputar berlawanan dengan jarum jam. Hal inilah yang memungkinkan *quadcopter* untuk dapat terbang dengan stabil. *Quadcopter* memiliki kelebihan dalam hal mobilitas dan fleksibilitas dalam menjelajahi wilayah yang sempit. Selanjutnya pada penelitian Hadi et al. (2017), dilakukan pengendalian robot dengan jenis *Unmanned Aerial Vehicle* yaitu *AR.Drone quadcopter* dengan menggunakan *microsoft kinect*, Pengendalian *AR.Drone quadcopter* dilakukan dengan mengakuisisi data pergerakan dari pengguna dengan menggunakan *micrososft kinect* yang kemudian diolah didalam komputer dengan menggunakan bahasa pemograman *node.js*. Setelah data pergerakan tubuh dari pengguna diolah dengan node.js, maka komputer akan

memberikan instruksi kepada *AR.Drone quadcopter* berupa pergerakan navigasi yang dimana instruksi yang diberikan tergantung dari pergerakan tubuh yang dilakukan oleh pengguna dihadapan *microsoft kinect*. Namun terdapat kekurangan pada penelitian yang telah dilakukan oleh Hadi et al. (2017) tersebut. Pengendalian *quadcopter* dengan menggunakan *Kinect* dilakukan tanpa menggunakan sebuah kondisi yang berguna untuk meminimalkan kesalahan gerakan dari pengguna. Hal ini menyebabkan ketika pengguna melakukan pergerakan yang tidak dirancang oleh sistem, maka *quadcopter* tetap akan mengeksekusi instruksi dari pergerakan pengguna tersebut secara acak yang menyebabkan pengguna kehilangan kendali dalam melakukan kegiatan navigasi *quadcopter*.

Berdasarkan dari kekurangan pada penelitian yang telah dilakukan oleh Hadi et al. (2017) dalam hal menangani kesalahan gerakan yang dilakukan oleh pengguna, maka pada penelitian ini dilakukan implementasi pada sistem kendali navigasi robot berjenis *quadcopter* atau lebih umum dikenal sebagai *drone* dengan menggunakan metode *finite state machine* menggunakan *gesture* tubuh dari pengguna. Penggunaan metode *finite state machine* dapat menangani masalah yang kerap muncul pada robot *Unnamed Aerial Vehicle* yaitu *fault tolerance* (Vitor, et al., 2014). Sehingga penerapan model *finite state machine* dapat meminimalkan kesalahan gerakan dari pengguna. Gerakan *gesture* tubuh dari pengguna dideteksi menggunakan sensor kamera *kinect*. Pemilihan sensor *Kinect* ini dikarenakan menurut Zeng (2012) *Kinect* memiliki kamera RGB dan *infrared* yang bisa digunakan untuk menghasilkan *sceleton tracking* dengan proses yang sangat cepat. Selain itu pada penelitian Sanna et al. (2013) yang menyatakan bahwa sensor ini memiliki akurasi penyelesaian *task* secara akurat yaitu sebesar 100%. Hal ini membuat tingkat keakuratan dalam pendeteksian gerakan tubuh menjadi sangat baik. *Quadcopter* yang digunakan adalah *Parrot AR.Drone*. Pemilihan *Parrot AR.Drone* dikarenakan jenis ini merupakan salah satu *quadcopter* yang bersifat *open source* dengan harga terjangkau.

1.2 Rumusan masalah

Berdasarkan latar belakang di atas, penulis merumuskan permasalahan sebagai berikut:

1. Apa pengaruh penggunaan metode *finite state machine* terhadap navigasi dan pengendalian *AR.Drone quadcopter* dengan menggunakan sensor *kinect* ?
2. Bagaimana tingkat ketepatan gerakan yang dihasilkan oleh *kinect* sebagai sistem kendali *quadcopter* ?
3. Seberapa besar total selisih *delay* yang terjadi pada sistem navigasi dan kendali *AR.Drone quadcopter* dengan menggunakan sensor *kinect* antara sistem yang menggunakan metode *finite state machine* dengan sistem yang tidak menggunakan metode *finite state machine* ?

1.3 Tujuan

Berdasarkan latar belakang dan rumusan masalah yang didapat, maka tujuan skripsi ini adalah:

1. Mengetahui cara penggunaan *kinect* sebagai sistem kendali navigasi *quadcopter*.
2. Mengetahui tingkat ketepatan gerakan yang dihasilkan sistem kendali *quadcopter* dengan menggunakan *Kinect*.
3. Mengetahui pengaruh penggunaan metode *finite state machine* pada *quadcopter* dengan menggunakan *Kinect* dalam segi keamanan.
4. Mengetahui seberapa besar *delay* yang terjadi antara *input* dari *Kincet* dan pergerakan *quadcopter*.
5. Mengetahui seberapa besar total selisih *delay* yang terjadi pada sistem navigasi dan kendali *AR.Drone quadcopter* dengan menggunakan sensor *kinect* antara sistem yang menggunakan metode *finite state machine* dengan sistem yang tidak menggunakan metode *finite state machine*.

1.4 Manfaat

Manfaat dari penelitian ini antara lain sebagai berikut.

1. Membantu pengguna agar lebih mudah dalam mengendalikan *quadcopter* khususnya bagi pengguna pemula.
2. Dapat mengetahui dan memahami implementasi dari NUI dengan *Kinect* sebagai sistem kendali *Quadcopter.*
3. Dapat mengetahui dan memahami implementasi dari model *finite state machine* dengan *Kinect* sebagai sistem kendali *Quadcopter.*
4. Dapat mengembangkan penelitian-penelitian sebelumnya sehingga menghasilkan *output* penelitian yang lebih baik lagi.

1.5 Batasan Masalah

Batasan masalah pada penelitian ini antara lain sebagai berikut.

1. Model sistem yang digunakan adalah *finite state machine*
2. Jenis robot yang digunakan adalah *quadcopter*.
3. *Quadcopter* dioperasikan pada dalam ruangan.
4. Sensor kamera yang digunakan adalah *Kinect*.
5. Jarak pengguna dan *Kincet* maksimal sejauh 4 meter.
6. Saat melakukan pengendalian *quadcopter* didepan *Kinect*, tidak boleh ada orang lain di sekitar pengguna.

1.6 Sistematika Pembahasan

Berikut uraian singkat dari masing masing BAB yang terdapat pada penelitian ini

BAB I Pendahuluan

Bab ini berisi tentang latar belakang, Rumusan Masalah, Batasan Masalah, Tujuan Penulisan, Manfaat Penulisan, dan Sistematika Penulisan.

BAB II Landasan Kepustakaan

Bab ini menjeaskan dasar teori yang mendasari penulis untuk mengangkat tema yang dijadikan penelitian. Penjelasan tentang teknologi yang akan dijadikan objek penelitian juga dibahas pada bab ini.

BAB III Metodologi

Bab ini menjelaskan mengenai langkah-langkah kerja yang dilakukan dalam penelitian ini. Pada bab ini juga berisi tentang kesimpulan dari dasar teori yang telah dijadikan obyek peneliian.

BAB IV Rekayasa Kebutuhan

Bab ini menjelaskan secara rinci yang terkait meliputi deskripsi umum dari sistem, kebutuhan perangkat keras dan lunak, kebutuhan fungsional, dan batasan desain sistem.

BAB V Perancangan dan Implementasi

Bab ini menjelaskan perancangan sistem penelitian serta implementasi sistem penelitian berupa implementasi perangkat keras, implementasi perangkat lunak.

BAB VI Pengujian dan Analisis

Bab ini membahas beberapa langkah kerja dalam melaksanakan pengujian dan menganalisis pengujian yang telah dilakukan.

BAB VII Penutup

Bab ini membahas tentang kesimpulan dan saran yang diperoleh dari rumusan masalah penelitian setelah melakukan analisis dan pengujian.

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab ini akan dijelaskan tinjauan pustaka terhadap penelitian-penelitian yang sudah dilakukan sebelumnya sehingga dapat menjadi acuan pada penelitian ini, dan dasar teori terhadap perangkat perangkat keras dan lunak yang digunakan.

2.1 Tinjauan Pustaka

Penelitian tentang implementasi kendali *unmanned aerial vehicle* menggunakan gerakan tubuh pernah dilakukan oleh Hadi et al. (2017) yang membahas mengenai pengendalian *quadcopter* dengan menggunakan *Kinect*. Pada penelitian tersebut gerakan tubuh digunakan untuk mengendalian gerakan *quadcopter*. Gerakan tubuh akan diakuisisi oleh *kinect* dan diproses pada komputer. Kemudian data pergerakan tubuh tersebut diteruskan pada *quadcopter* melalui jaringan *Wi-Fi*. Hasil dari penelitian tersebut menunjukkan bahwa penggunaan *kinect* sebagai kontrol sangat baik dalam segi pendeteksi objek. Hal ini dibuktikan dengan persentase *task* yang *complete* sebesar 100%. Namun tidak ada sebuah *state* pada sistem tersebut yang menyebabkan adanya kemungkinan dari pengguna untuk melakukan kesalahan gerakan yang dapat menyebabkan hal-hal yang berbahaya bagi pengguna dan lingkungan sekitar.

Untuk mengurangi kemungkinan terjadinya kesalahan yang fatal dari pengguna, maka diterapkan model *finite state machine* pada *unmanned aerial vehicle*. Penelitian tentang implementasi model *finite state machine* pada robot pernah dilakukan oleh Vitor et al. (2014) yang membahas proses pengontrolan *Unmanned aerial vehicle* dengan menggunakan *finite state machine* yang menghasilkan sebuah *high level system* termasuk semua fitur-fitur yang dibutuhkan pada sebuah UAV control yang kompleks. UAV control kompleks digunakan untuk menyelesaikan tugas-tugas yang spesifik seperti *fast respond* untuk masukan *non-linear* dan toleransi terhadap kesalahan. Selanjutnya pada penelitan Mihn et al. (2017) dilakukan sebuah strategi pengendalian pada model robot *biped* termasuk *force direction control* dan pengendalian pergelangan kaki robot sederhana. Penelitian tersebut menerapkan model *State machine* pada pengontrolan pergelangan kaki robot dengan tujuan mengurangi *impulsive impact* pada robot ketika sedang berjalan. Pengurangan impuls impact dilakukan dengan pengendalian torsi pada pergelangan kaki robot dalam fase *heel strike*.

Berdasarkan penelitian-penelitian tersebut maka dapat disimpulkan bahwa banyak cara menerapkan model *finite state machine* pada robot dengan tujuan untuk meningkatkan kegunaan dan tingkat keamanan robot ketika berada pada lingkungan manusia. Namun tentunya masih ada kekurangan dalam penelitian-penelitian sebelumnya. Pada penelitian Vitor et al. (2014), model *finite state machine* tidak diimplementasikan pada bahasa pemograman tingkat tinggi dengan tujuan untuk memungkinkan pengoperasian didalam skenario *real time* menggunakan sebuah UAV yang sebenarnya. Lalu pada penelitian Mihn et al. (2017), penerapan model state machine masih diterapkan pada robot dengan

jenis *biped*. Kemudian pada penelitian yang dilakukan oleh Hadi et al. (2017), pengendalian *quadcopter* dengan menggunakan *Kinect* dilakukan tanpa menggunakan sebuah kondisi yang berguna untuk meminimalkan kesalahan gerakan dari pengguna. Hal ini menyebabkan ketika pengguna melakukan pergerakan yang tidak dirancang oleh sistem, maka *quadcopter* tetap akan mengeksekusi instruksi dari pergerakan pengguna tersebut secara acak yang menyebabkan pengguna kehilangan kendali dalam melakukan kegiatan navigasi *quadcopter*. Sehingga pada penelitian ini dilakukan implementasi model *finite state machine* yang diimplementasikan dengan bahasa pemograman tingkat tinggi pada sistem navigasi dan kendali *AR.Drone quadcopter* menggunakan *microsoft kinect*. Pengguna dapat menjalankan dan mengendalikan *quadcopter* dengan nyaman walaupun dalam lingkungan yang ramai dengan resiko kecelakaan yang kecil. Sistem yang dirancang juga memastikan *quadcopter* masih dalam keadaan yang aman ketika pengguna melakukan pergerakan yang tidak dirancang oleh sistem. Untuk *output* dari sistem yang dibuat langsung diimplementasikan pada *quadcopter* tanpa menggunakan simulasi.

2.2 Dasar Teori

Pada bagian ini dijelaskan mengenai apa saja teori dasar yang melandasi penelitian ini meliputi *Microsoft Kinect*, *quadcopter*, *finite state machine* serta AT *command*.

2.2.1 *Quadcopter*

Menurut Kusuma, Effendi, dan Iskandar (2012) *quadcopter* adalah sebuah konfigurasi empat buah motor pada sebuah kerangka berbentuk meyilang. Bahan yang sering digunakan sebagai kerangka untuk *quadcopter* merupakan bahan dasar fiber dikarenakan sifatnya yang ringan dan kuat. Pada masing-masing motor terpasang baling-baling untuk membuat aliran udara yang menghasilkan tekanan ke arah sehingga timbul gaya angkat pada *quadcopter*. Dua baling-baling tersebut berputar searah jarum jam dan dua baling-baling lainnya akan berputar berlawanan dengan jarum jam. Hal inilah yang memungkinkan *quadcopter* untuk dapat terbang dengan stabil. *Quadcopter* memiliki beberapa istilah dalam operasi gerakannya, antara lain:

1. *Roll* yaitu gerakan *quadcopter* ke arah kiri dan kanan. Gerakan ini dilakukan dengan melakukan perubahan kecepatan pada pasangan motor bagian kiri dan kanan. Sebagai contoh seperti pada Gambar 2.1, saat *quadcopter* ingin bergerak kekanan maka kecepatan pasangan motor sebelah kanan akan dikurangi dan kecepatan motor bagian kiri akan ditambah, begitu pula sebaliknya. Gerakan ini berjalan secara linear pada koordinat sumbu y dari *quadcopter*.

Gambar 2.1 Gerakan *Roll Quadcopter*

2. *Pitch* yaitu gerakan kedepan dan kebelakang. Gambar 2.2 mengilustrasikan bahwa gerakan ini dilakukan dengan melakukan perubahan kecepatan pada pasangan motor bagian depan dan belakang. Sebagai contoh saat ingin bergerak maju maka kecepatan pasangan motor depan akan dikurangi dan kecepatan motor bagian belakang akan ditambah, begitu pula sebaliknya. Gerakan ini berjalan secara linear pada koordinat sumbu x dari *quadcopter*.

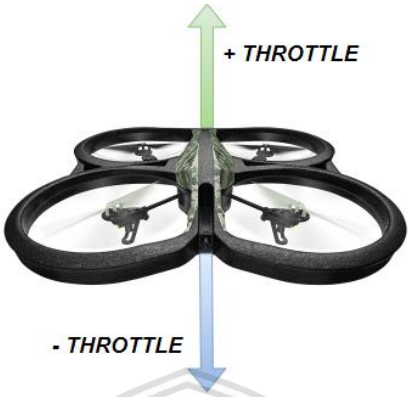
Gambar 2.2 Gerakan Pitch *Quadcopter*

3. *Yaw* merupakan gerakan berputar ke kiri dan ke kanan. Gerakan ini dilakukan dengan melakukan perubahan kecepatan pada pasangan motor secara silang seperti pada gambar 2.3. Gerakan ini berjalan secara rotasi pada koordinat sumbu z dari *quadcopter*.

Gambar 2.3 Gerakan Yaw *Quadcopter*

4. *Throttle* merupakan gerakan dari *quadcopter* untuk naik atau turun. Seperti pada gambar 2.4, saat *quadcopter* naik maka keseluruhan motor akan ditambah kecepatan. Sedangkan jika *quadcopter* turun maka keseluruhan

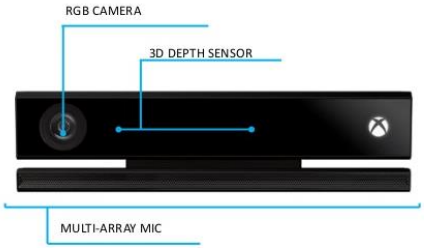
kecepatan motor akan diturunkan. Gerakan ini berjalan secara linear pada koordinat sumbu z dari *quadcopter*.



Gambar 2.4 Gerakan *Throttle Quadcopter*

2.2.2 Microsoft Kinect

Microsoft Kinect adalah perangkat *input* yang digunakan untuk mendeteksi gerakan. Perangkat ini diproduksi oleh microsoft sebagai *controller game XBOX* 360 dan PC dengan sistem operasi *Windows*. Dengan menggunakan *Kinect*, pengguna tidak perlu menyentuh *controller game* secara langsung dan cukup dengan melakukan gerakan-gerakan yang alami. Dengan kata lain pengguna dapat melakukan suatu pengoperasian komputer hanya dengan menggunakan gerakan tangan atau gerakan tubuh lainnya. *Kinect* didasarkan dari teknologi perangkat lunak yang dikembangkan secara internal oleh *Rare*, anak perusahaan dari *Microsoft Game Studios* milik *Microsoft*, dan teknologi kamera oleh pengembang Israel, PrimeSense. PrimeSense mengembangkan sistem yang dapat mengintepretasikan gerakan secara spesifik, sehingga pengendalian tanpa memerlukan sentuhan dapat dilakukan pada perangkat elektronik menggunakan proyektor *infrared* dan kamera, serta *prosesor* khusus untuk melacak pergerakan individu maupun objek pada bidang tiga dimensi. Sistem tiga dimensi *scanner* tersebut dinamakan *light coding* yang menggunakan variasi dari rekonstruksi gambar.



Gambar 2.5 *Microsoft Kinect*

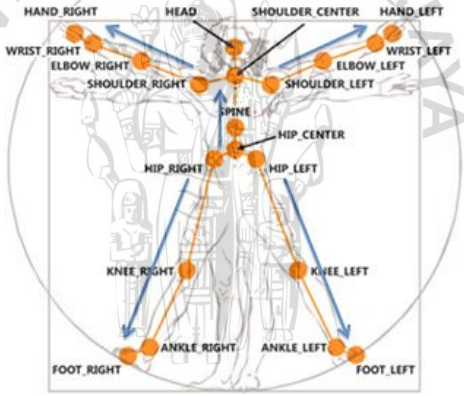
Kinect memiliki beberapa fitur meliputi kamera RGB, *depth sensor* sensor kedalaman, dan *multi-array microphone* seperti pada gambar 2.5. Dari keseluruhan fitur tersebut *depth sensor* memiliki peran paling penting dalam membedakan antara *kinect* dengan sebuah kamera. *Depth sensor* digunakan

untuk mendapatkan data sebuah area dalam bentuk 3D tanpa mempedulikan kondisi cahaya pada area tersebut.

2.2.2.1 Sceleton Tracking

Sceleton Tracking merupakan fitur dari *Kinect Software Developmetn Kit* (SDK) yang telah disediakan oleh *Microsoft*. Fitur ini dapat melacak titik sendi utama pada tubuh manusia dan mengubahnya menjadi kerangka berbentuk *stick figure* yang terdiri dari satu set tulang saling berhubungan. Lalu pada gambar 2.6 diperlihatkan data titik sendi yang telah disediakan oleh *Microsoft* dalam *Kinect SDK*. *Sceleton tracking* ini tidak lepas dari hasil penggunaan *depth sensor*.

Depth sensor yang memetakan objek-objek berdasarkan jarak yang akan dibandingkan dengan data hasil *training* yang ada dalam prosesor *Kinect*. Data *training* tersebut dibuat menggunakan 100.000 *frame* gambar manusia yang diambil dari posisi yang berbeda-beda pada saat berdiri. Oleh karena itu pada saat pengguna dalam posisi duduk, *Kinect* tidak dapat mendeteksi kaki manusia tersebut (Aron, 2011).



Gambar 2.6 Titik-titik *sceleton tracking* yang disediakan *Microsoft*

Sumber: (*Microsoft*, 2017)

2.2.3 AT Command

AT *command* atau juga dikenal sebagai *Hayes Standard* AT *Command* adalah sebuah petunjuk standar untuk mengkonfigurasi dan mengendalikan modem. Perangkat ini dikembangkan oleh *Hayes Microcomputer Product* yang merupakan pelopor dibidang modem dan digunakan secara keseluruhan atau sebagaian oleh hampir setiap produsen modern untuk digunakan dengan komputer probadi. (Linux Information Project, 2005)

AT *command* merupakan sekumpulan perintah yang dapat digunakan dalam komunikasi dengan menggunakan *serial port*. Penerapan AT *command* umumnya terdapat pada penggunaan modem dan *handphone*. Dengan menggunakan AT command, maka dapat dijalankan beberapa fungsi seperti melihat vendor dari

modem atau *handphone* yang digunakan, kekuatan sinyal, mengirim, membaca, menghapus dan memperbarui SMS, dan sebagainya. AT *command* pada dasarnya mirip dengan perintah *promt* pada DOS yaitu perintah-perintah yang digunakan untuk penulisan ke *port* komputer.

Untuk penulisan AT *command* akan selalu diawali dengan kata AT yang merupakan singkatan dari *attention,* kemudian diikuti dengan karakter lainnya yang memiliki fungsi masing-masing. Selain digunakan untuk penulisan ke *port*, AT *command* juga dapat diguanakan untuk penulisan instruksi ke modem. Beberapa contoh penulisan AT *command* adalah sebagai berikut.

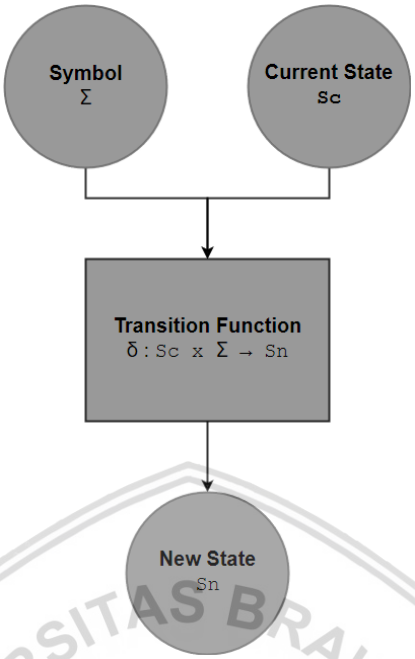
1. AT : mengetahui kondisi *port* juga siap untuk berkomunikasi
2. AT+CGMI : perintah untuk mengetahui merek handphone
3. AT+CMGR : perintah untuk membaca isi dari SMS

2.2.4 *Finite State Machine*

Finite state machine adalah sebuah model komputasi yang terdiri dari sekumpulan kondisi, sebuah kondisi awal, sebuah masukan, dan sebuah fungsi transisi yang memetakan masukan dan kondisi saat ini ke kondisi selanjutnya. Proses komputasi dimulai pada dengan sebuah *input string*. Kemudian berubah menjadi kondisi baru sesuai dengan *transition function*. *Finite state machine* secara formal didefinisikan dengan menggunakan lima *tuple* yaitu (Q, Σ, δ, S, F) yang setiap *tuple* memiliki penjelasan sebagai berikut :

1. Q : Himpunan terbatas dari state
2. Σ : Himpunan terbatas dari simbol *input*
3. δ : Sebuah fungsi transisi yang merupakan hasil pemetaan dari Q dengan Σ
4. S : State awal atau initial state, yang dimana S ∈ Q
5. F : Kumpulan state yang diterima oleh sistem

Berdasarkan pada interpretasi matematika diatas, *finite state machine* memiliki sebuah state dengan jumlah yang terbatas. Setiap *states* menerima masukan dari sebuah input dengan jumlah yang terbatas dan setiap states memiliki aturan yang mendeskripsikan aksi pada mesin untuk setiap input yang diberikan. Pada gambar 2.7 ditunjukkan sistem *finite state machine* yang disajikan dalam bentuk *state diagram*. Pada gambar state diagram tersebut, state direpresentasikan dalam bentuk node, transisi direpresentasikan dalam bentuk panah, dan hubungan antara input dan output dilambangkan dalam bentuk simbol.



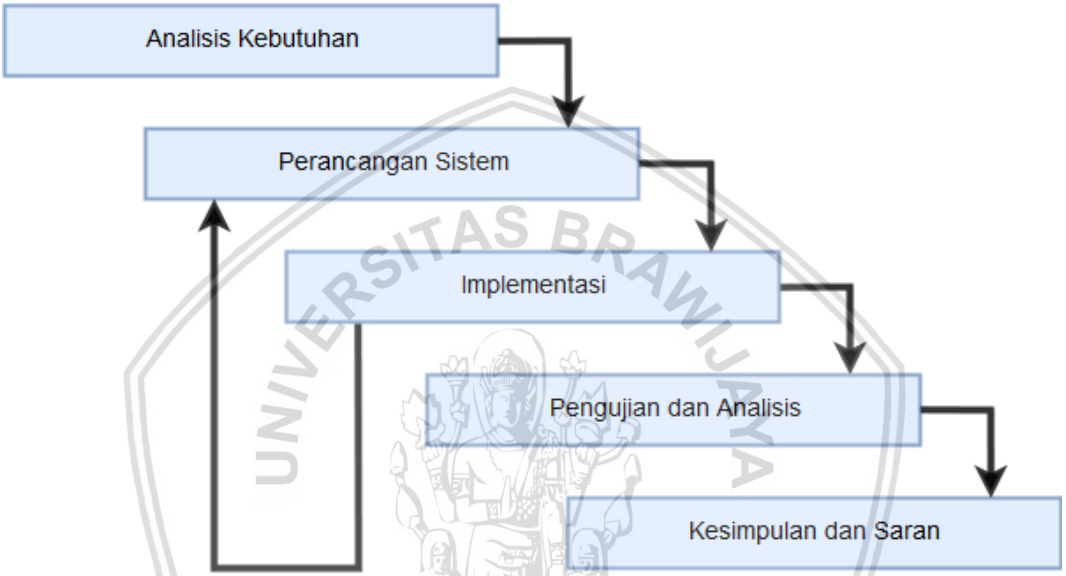
Gambar 2.7 Diagram *finite state machine*

Pada gambar tersebut juga terdapat simbol *input* yang digunakan sebagai fungsi transisi state, simbol *current state* yang menunjukkan keadaan state yang sedang berlangsung pada saat itu yang dilambangkan dengan simbol Sc, simbol *transition function* yang berguna untuk mengubah state yang sedang berlangsung, dan simbol new state yang merupakan sebuah state yang telah melakukan perubahan state yang disebabkan oleh inputan sistem dan fungsi transisi yang lambangkan dengan simbol Sn.

BAB 3 METODOLOGI

3.1 Metode Penelitian

Bab ini menjelaskan langkah-langkah yang dilakukan dalam pembuatan sistem. Adapun langkah-langkah yang akan dilakukan secara berurutan yaitu analisis kebutuhan, perancangan sistem, implementasi, pengujian dan analisis, serta pengambilan kesimpulan dan saran sebagai referensi untuk pengembangan penelitian selanjutnya.



Gambar 3.1 Diagram alir penelitian

Pada gambar 3.1 diperlihatkan diagram alir tahap pengerjaan penelitan ini. Pada bab analisis kebutuhan akan dibahas apa saja kebutuhan yang diperlukan oleh sistem, mulai dari dari kebutuhan antarmuka pengguna, kebutuhan perangkat keras, kebutuhan perangkat lunak, kebutuhan fungsional, dan kebutuhan non-fungsional. Paa bab perancangan sistem akan dibahas mengenai tahap-tahap dalam merancang sistem. Kemudian dilanjutnya dengan bab implementasi sistem yang akan membahas bagimana mengimplementasikan sistem yang telah dirancang sebelumnya. Apabila hasil pada tahap implementasi tidak sesuai dengan yang diharapkan, maka akan dilakukan tahap perancangan sistem kembali. Jika hasil implementasi sudah sesuai dengan yang diharapkan, maka akan dilanjutnya ke bab pengujian dan analisis. Pada bab pengujian dan analisis terhadap sistem yang telah dirancang akan dilakukan pengujian dan dilakukan penganalisisan terhadap hasil dari pengujian tersebut lalu ditarik kesimpulan dan pengambilan saran.

3.2 Analisis Kebutuhan

Pada bagian ini dijelaskan mengenai segala kebutuhan yang diperlukan untuk membangun sistem meliputi gambaran umum sistem, kebutuhan antarmuka pengguna, kebutuhan perangkat keras, kebutuhan perangkat lunak, serta kebutuhan fungsional dan non-fungsional. Penjelasan lebih rinci mengenai analisis kebutuhan akan dibahas pada bab 4 analisis kebutuhan.

3.2.1 Gambaran Umum Sistem

Pada gambaran umum sistem akan dijelaskan mengenai penelitian tentang penggunaan state machine untuk navigasi dan kendali *AR.Drone quadcopter* menggunakan *microsoft Kinect* secara singkat.

3.2.1 Kebutuhan Antarmuka Pengguna

Pada kebutuhan antarmuka pengguna akan dibahas apa saja yang diperlukan agar pengguna dapat melakukan interaksi dengan sistem yang telah dirancang pada penelitian ini.

3.2.2 Kebutuhan Perangkat Keras

Pada kebutuhan perangkat keras akan dijelaskan mengenai semua perangkat keras yang dibutuhkan oleh sistem pada penelitan ini.

3.2.3 Kebutuhan Perangkat Lunak

Pada kebutuhan perangkat lunak akan dijelaskan mengenai semua perangkat lunak yang dibutuhkan oleh sistem pada penelitian ini.

3.2.4 Kebutuhan Fungsional

Kebutuhan fungsional merupakan pernyataan yang menjelaskan bagaimana sistem yang akan dibuat nantinya dapat menghasilkan keluaran sesuai dengan yang diinginkan, sehingga kebutuhan fungsional harus dapat dipenuhi.

3.2.5 Kebutuhan Non-Fungsional

Hal yang akan dibahas pada kebutuhan non-fungsional meliputi karakteristik pengguna, lingkungan operasi sistem, batasan perancangan sistem, pengumpulan data, serta asumsi dan ketergantungan sistem.

3.2.6 Kebutuhan Komunikasi

Pada kebutuhan komunikasi sistem akan dijelaskan mengenai semua proses komunikasi yang dibutuhkan oleh sistem dalam penelitian ini.

3.3 Perancangan Sistem

Perancangan sistem dilakukan setelah mengetahui kebutuhan yang diperlukan untuk melakukan penelitian ini agar pada saat tahap implementasi berjalan secara terstruktur dan sesuai dengan tujuan yang akan dicapai. Pada sistem kendali

navigasi *quadcopter* ini terdapat *input* dan *output* dari sistem. Dimana input pada sistem ini berupa gerakan dari pengguna yang akan dideteksi menggunakan sensor kamera kinect dengan hasil berupa nilai dari sceleton tracking yang telah disediakan oleh SDK dari *kinect*. Lalu untuk output dari sistem ini adalah berupa pergerakan dari *quadcopter*. Kemudian pemrograman yang digunakan untuk mengolah data dari kinect serta untuk mengirim data dari komputer ke *quadcopter* adalah javascipt. Selain itu sistem ini juga akan menampilkan beberapa info data navigasi dari *quadcopter*. Pada sistem ini juga diimplementasikan model finite state machine yang menyebabkan *quadcopter* hanya akan berjalan jika ada masukan dari pengguna melalui *kinect* dan juga jika beberapa kondisi yang telah diatur pada *quadcopter* telah terpenuhi. Perancangan pada sistem ini akan dibagi menjadi perancangan *finite state machine*, perancangan gerakan tubuh, perancangan pengiriman AT *command*, perancangan antarmuka pengguna, dan perancangan komunikasi sistem.

3.4 Implementasi Sistem

Implementasi sistem dilakukan sesuai dengan perancangan yang telah ditentukan sebelumnya. Pada tahap ini akan dilakukan implementasi dari keseluruhan perancangan yang telah dilakukan. Implementasi ini akan melalui beberapa tahapan. Pada tahap pertama dilakukan implementasi dari model *finite state machine* pada *AR.Drone* dengan menggunakan Kinect. Selanjutnya dilakukan implementasi gerakan tubuh. Tahap ini akan dilakukan pembuatan kode program berdasarkan data yang telah dipeloreh dari *kinect* sesuai dengan perancangan yang telah dilakukan sebelumnya. Pada tahap ketiga dilakukan implementasi pada pengiriman AT *command*. Pada tahap keempat dilakukan implementasi komunikasi sistem. Pada tahap terakhir akan dilakukan implementasi komunikasi sistem.

3.5 Pengujian dan Analisis

Pengujian dan analisis sistem dilakukan untuk mengetahui apakah kinerja dan performa keseluruhan sistem yang telah dirancang ini telah sesuai dengan yang dinginkan. Untuk skenario pengujian yang akan dilakukan yaitu pengujian metode *finite state machine*, pengujian ketepatan gerakan, dan pengujian *delay* sistem.

3.6 Kesimpulan dan Saran

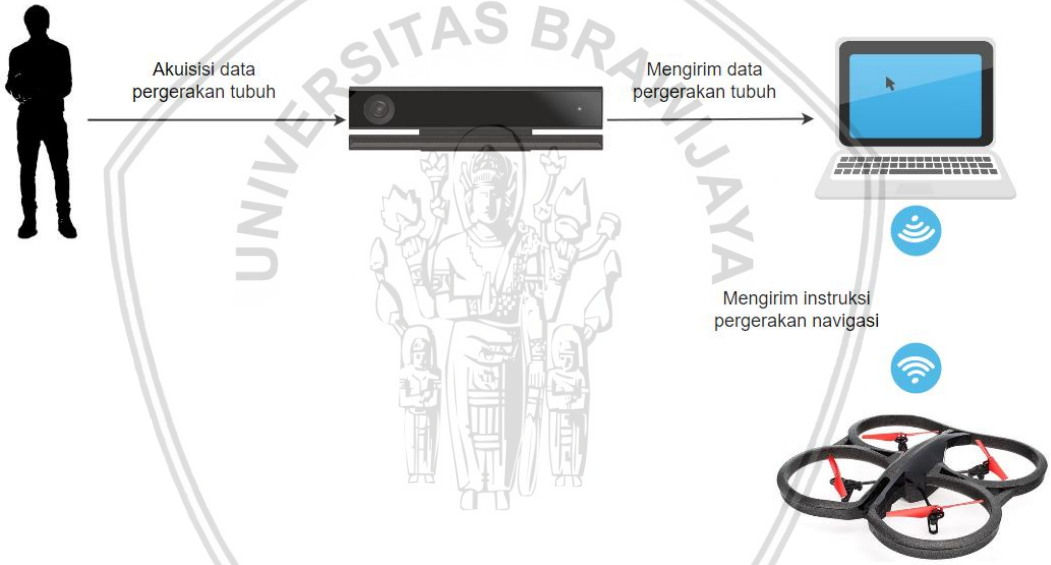
Kesimpulan didapatkan setelah melakukan perancangan, implementasi, pengujian dan analisis terhadap sistem. Kesimpulan disusun berdasarkan hasil pengujian dan anlisis yang dibuat. Isi dari kesimpulan diharapkan dapat menjadi acuan pada penelitian lain untuk mengembangkan sistem ini. Di dalam penulisan akhir terdapat saran yang bertujuan untuk memberikan masukan dan ide pengembangan dari penelitian ini sehingga diharapkan penelitan ini akan menjadi lebih baik lagi.

BAB 4 ANALISIS KEBUTUHAN

Bab ini akan menjelaskan berbagai macam kebutuhan yang dibutuhkan oleh sistem, dimulai dari gambaran umum sistem, kebutuhan antarmuka pengguna, kebutuhan perangkat keras, kebutuhan perangkat lunak, kebutuhan komunikasi, kebutuhan fungsional, dan kebutuhan non-fungsional sistem.

4.1 Gambaran Umum Sistem

Sistem kendali dan navigasi pada *quadcopter* dilakukan dengan menangkap data pergerakan tubuh dengan menggunakan *microsoft kinect*. Kemudian data pergerakan tubuh yang diterima oleh kinect dikirim ke komputer melalui USB 3.0. Program yang telah dirancang pada komputer memanfaatkan pergerakan data pergerakan tubuh yang dikirim dari kinect. Data pergerakan tubuh yang ada pada komputer dikirm ke *quadcopter* melalui komunikasi *wi-fi*.



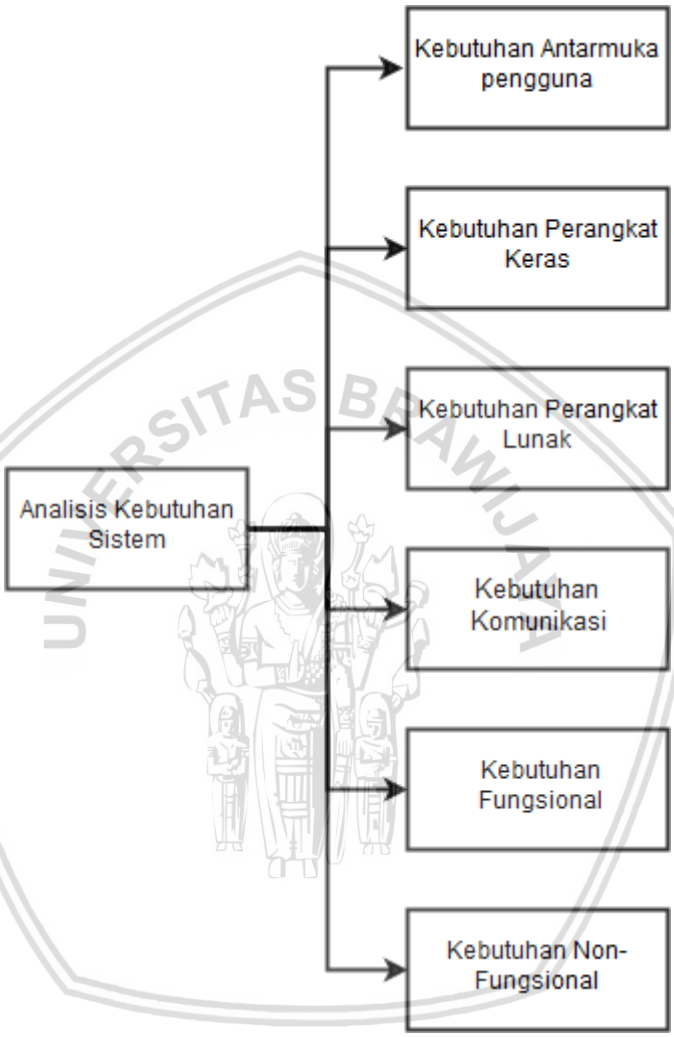
Gambar 4.1 Gambaran umum sistem

Sistem ini akan bekerja sesuai dengan diagram blok pada gambar 4.1. Gerakan tubuh dari pengguna dideteksi menggunakan *kinect*. Lalu kinect akan mengirimkan data gerakan pengguna menuju komputer melalui USB. Selanjutnya data gerakan akan diproses oleh program yang dibuat di komputer dan data akan diteruskan menuju *quadcopter* melalui jaringan *Wi-Fi*. Terdapat input dan output pada sistem kendali *quadcopter*. Input pada sistem berupa data dari pengguna yang dideteksi menggunakan kamera pada kinect yang berfungsi sebagai pendeteksi gerakan tubuh. Data yang diakuisisi dari sensor kinect berupa nilai sceleton tracking pengguna. Output pada sistem ini berupa gerakan pada *quadcopter*. Kemudian pada sistem ini dilakukan penerapan metode *finite state machine* yang menyebabkan *output* sistem tidak hanya tergantung dari inputan sistem yaitu pergerakan tubuh dari pengguna, tetapi juga tergantung pada kondisi *quadcopter* pada saat itu. Bahasa pemograman yang digunakan untuk mengolah data yang didapat dari sensor kinect ke komputer

dan meneruskan data dari komputer ke *quadcopter* adalah javascript. Selain itu sistem juga akan menampilkan info data navigasi yang didapat dari *quadcopter*.

Penelitian ini dilakukan untuk mengetahui pengaruh dari penggunaan mode *finite state machine* terhadap kendali dan navigasi pada quadcopter menggunakan kinect.

4.2 Analisis Kebutuhan Sistem

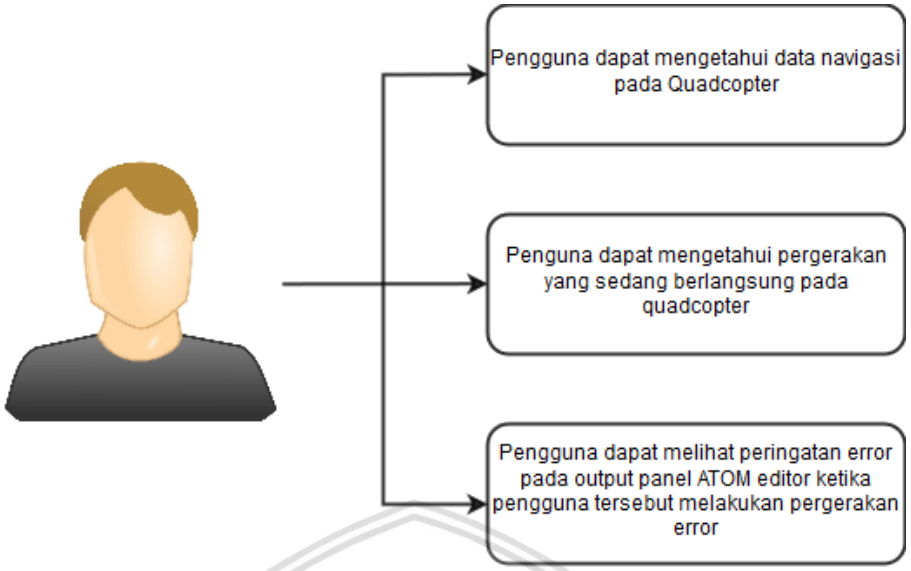


Gambar 4.2 Analisis kebutuhan sistem

Analisis kebutuhan sistem menjelaskan apa saja kebutuhan yang diperlukan oleh sistem, mulai dari dari kebutuhan antarmuka pengguna, kebutuhan perangkat keras, kebutuhan perangkat lunak, kebutuhan fungsional, kebutuhan non-fungsional, dan kebutuhan komunikasi. Berikut uraian mengenai analisis kebutuhan sistem berdasarkan skema analisis kebutuhan sistem pada Gambar 4.2.

4.2.1 Kebutuhan Antarmuka Pengguna

Kebutuhan antarmuka pengguna merupakan kebutuhan yang harus terpenuhi agar pengguna dapat berinteraksi dengan sistem. Kebutuhan antarmuka pengguna pada sistem ini diperlihatkan pada gambar 4.3.



Gambar 4.3 Kebutuhan antarmuka pengguna

kebutuhan antarmuka pengguna pada sistem ini menggunakan *command line interface* pada *output panel ATOM editor* yang akan menampilkan berbagai informasi yang telah diolah oleh sistem. Informasi dari *quadcopter* yang akan ditampilkan pada *command line interface* yaitu data navigasi pada *quadcopter* berupa ketinggian, kecepatan, dan kondisi dari pergerakan *quadcopter* tersebut serta peringatan *error* pada output panel ATOM editor ketika pengguna tersebut melakukan pergerakan error.

4.2.2 Kebutuhan Perangkat Keras

Pada bagian ini dijelaskan mengenai semua perangkat keras yang dibutuhkan oleh sistem yaitu:

a. Parrot A.R. Drone 2.0

Parrot *AR.Drone* merupakan sebuah *quadcopter* yang dikendalikan secara remote control. Parrot *AR.Drone* dikembangkan oleh perusahan dari prancis yaitu Parrot. Drone ini dirancang untuk dapat dikendalikan melalui sistem operasi tablet atau mobile seperti contohnya mendukung sistem operasi IOS dan Android. Parrot *AR.Drone* pertama kali diresmikan pada International CES di Lav Vegas pada tahun 2010, bersamaan dengan demonstrasi dari aplikasi IOS yang digunakan untuk mengontrol drone tersebut. Aplikasi pada IOS tersebut dirancang untuk melakukan operasi pada drone secara leluasa. Parrot juga merilis *AR.Race*, yang memperbolehkan pengguna untuk ikut serta didalam permainan tunggal atau berinterkasi dengan drone-drone yang lain didalam simulasi pertarungan.

Gambar 4.4 Parrot *AR.Drone* versi 2.0

AR.Drone versi 2.0 diresmikan di CES Las Vegas pada tahun 2012. *AR.Drone* versi 2.0 dapat dilihat pada gambar 4.4. Pada *AR.Drone* versi 2.0 dilakukan penyempurnaan pada sisi fungsionalitas. Perlengkapan pada Parrot *AR.Drone* telah dikembangkan secara signifikan. Kualitas kamera pada Drone telah dinaiikkan menjadi 720 pixel dan banyak sensor yang dipasang onBoard yang menyebabkan *quadcopter* menjadi lebih sensitif, yang menyebabkan quadcoptoer dapat dikendalikan dengan kontrol yang lebih baik. Spesifikasi para Parrot AR drone versi 2.0 dapat dilihat pada tabel 4.1.

Tabel 4.1 Spesifikasi *AR.Drone Quadcopter* 2.0

Sumber: (Parrot, 2017)

Komponen	Keterangan
Prosessor	1GHz 32 bit ARM *Cortex* A8 prosessor dengan 800 MHz
Sistem Operasi	*Linux* 2.6.32
RAM	1GB DDR2 RAM Berkecepatan 200 MHz
Konektivitas	Wi-Fi b, g, n
Gyroscope	3 axis *gyroscope* 2000° / *seconds precision*
Accelerometer	3 axis *Accelerometer* ± 50mg *precision*
Magnetometer	3 axis *magnetometer* 6° *precision*
Ultrasonik Sensor	Ultrasonik sensor untuk penghitungan ketinggian
HD Kamera	720p 30 FPS

Berdasarkan pada spesifikasi diatas, maka AR drone *quadcopter* versi 2.0 dipilih sebagai objek dalam penelitian ini. *Quadcopter* parrot AR *drone* versi 2.0 memiliki sistem operasi tertanam yang bersifat *open source* yaitu linux yang menyebabkan *quadcopter* tersebut mudah untuk dikembangkan dan diprogram oleh banyak peneliti. Pada *quadcopter* parrot AR *drone* ini juga dilengkapi dengan *prosessor* ARM Cortex A8 dengan *clock speed* sebesar 1GHz dan RAM sebesar 1GB yang menyebabkan *quadcopter* tersebut dapat memproses

instruksi-instruksi yang diberikan oleh pengguna dengan cepat. Dalam segi keamanan, *quadcopter* parrot AR *drone* versi 2.0 telah dilengkapi dengan pelindung dikeseluruhan baling-balingnya berupa *styrofoam* seperti pada gambar 4.4, sehingga aman untuk digunakan dalam kegiatan penelitian.

b. *Microsoft Kinect* Versi 2.0

Kinect adalah suatu perangkat keras yang mendeteksi pergerakan tubuh manusia yang dikembangkan oleh perusahaan microsoft. Kinect generasi pertama diperkenalkan secara umum pada bulan november 2010. Kinect versi 1.0 untuk *miscrosoft windows* dirilis pada tanggal 1 februari 2012. Kemudian microsoft merilis versi beta untuk software development kit yang ditujukan untuk kinect pada tanggal 7 juni 2010. Software development kit bertujuan tujuan untuk memungkinkan para developer menuliskan aplikasi kinect didalam bahasa pemograman seperti C++. Visual basic, atau javascript. Kemudia pada tahun 2012 microsoft merilis kinect versi 2.0.

Terdapat peningkatan dari segi spesifikasi hardware pada kinect versi 2.0 dibandingkan dengan versi sebelumnya. Perbandingan dari segi hardware dapat dilihat pada tabel 4.2. kinect versi 2.0 memiliki kualitas pixel kamera RGB yang lebih tinggi dibandingkan dengan versi sebelumnya yaitu 1920 x 1080 pixel. Lalu kinect versi 2.0 memiliki pixel dept kamera yang lebih tinggi dibandingkan dengan versi sebelumnya. Dan kinect versi 2.0 menggunakan USB 3.0 yang memiliki kecepatan transfer data yang lebih cepat dibandingkan dengan USB 2.0 yang digunakan pada kinect versi 1.0.

Tabel 4.2 Spesifikasi *kinect*

Sumber: (*Microsoft*, 2017)

Fitur	Spesifikasi
Kamera RGB	1920x1080 px @ 30 Hz
Kedalaman kamera	512x424 px @ 30 Hz
Jarak maksimal kedalaman kamera	4 meter
Jarak pandang horizontal	70 derajat
Jarak pandang vertikal	60 derajat
Versi USB	3.0
Minimum sistem operasi	Windows 8

Berdasarkan spesifikasi diatas maka *kinect* versi 2.0 dipilih sebagai perangkat untuk mendeteksi pergerakan tubuh dalam penelitian ini. Dengan kualitas kamera 1920x1080 *pixel* dan depth kamera 512x424 *pixel* mampu untuk menangkap data pada pergerakan tubuh dari pengguna secara akurat. Selain itu kinect versi 2.0 menggunakan USB 3.0 sehingga pertukaran data antara kinect dengan komputer dapat dilakukan dengan cepat.

Menurut David Catuhe (2012) sensor kinect dapat bekerja secara optimal berdasarkan pada rentang jarak sebagai berikut :

1. *Horizontal viewing angel* sebesar 57 derajat.
2. *Vertical viewing angel* sebesar 43 derajat Memiliki port USB 3.0.
3. Jarak pengguna untuk hasil yang optimal adalah 1.2 m hingga 4 m dari depan sensor kinect.
4. *Temperature* ruangan berada pada suhu 5 derajat celsius hingga 35 derajat celsius.

c. Komputer atau Leptop

Komputer diguanakan sebagai tempat mengolah data pergerakan tubuh yang didapatkan dari sensor kinect. Kemudian data pergerakan tubuh diolah menggunakan bahasa pemograman node.js yang nantinya mengasilkan keluaran perintah pada *quadcopter*. Spesifikasi minimal yang harus dipenuhi oleh komputer agar dapat digunakan pada penelitian adalah sebagai berikut:

1. Memiliki 64 bit (x64) processor.
2. Memiliki Random-access memory minimal sebesar 4 GB.
3. Memiliki port USB 3.0.
4. Memiliki Prosesor minimal Intel Core i5.

4.2.3 Kebutuhan Perangkat Lunak

Pada bagian ini dijelaskan semua kebutuhan perangkat lunak yang diperlukan oleh sistem ini antara lain sebagai berikut.

a. Sistem Operasi *Microsoft Windows* 10

Sistem operasi yang dapat digunakan dalam penggunaan sensor kinect versi dua yaitu *microsoft windows* 8, *micosoft windows* 8.1, dan *microsoft windows* 10. Sensor kinect tidak dapat digunakan pada sistem operasi *microsoft windows* yang memiliki versi dibawah dari windows 8. Sedangkan pada sistem operasi selain *windows*, dibutuhkan perangkat lunak tambahan agar dapat menjalankan sensor kinect versi 2.

b. Node.js

Node js merupakan *framework/platform* berbasis *javascript* yang dibangun didalam *Google Chrome's JavaScript V8 Engine*. Node.js menggunakan sebuah *event-driven*, *non-blocking I/O* model yang memungkinkan untuk mengembangkan *intensive web applications* seperti situs video streaming, *single-page applications*, dan aplikasi web lainnya. Node.js memiliki fitur *package manager* yaitu NPM yang merupakan sebuah *open source libraries* yang siap digunakan sesuai dengan kebutuhan para pengembang. Node Package Manager inilah yang menjadi alasan dalam pemilihan node.js sebagai bahasa program yang dipakai pada penelitian ini.

c. ATOM *Text Editor*

Atom merupakan sebuah *open-source text* dan *source code editor* yang dapat digunakan pada sistem operasi *macOS*, *linux*, dan *Micrososft Windows* dengan fitur bantuan berupa plug-ins yang berbasis bahasa pemograman node.js. ATOM merupakan sebuah desktop application yang dibangun menggunakan web technologies yang dikembangkan oleh GitHub. ATOM dibangun diatas Electron, yaitu sebuah framwork yang memungkinkan cross platform pada aplikasi desktop menggunakan chromium dan node.js. ATOM juga dapat digunakan sebagai integrated development environment (IDE)

d. Adobe Premiere

Adobe Premier merupakan sebuah perangkat lunak yang digunakan untuk video editing yang dibuat publish oleh *Adobe system*. Adobe premier pada penelitian ini digunakan untuk pengujian performa sistem dari segi waktu delay.

e. Micrososft Excel

Microsoft Excel merupakan perangkat lunak yang digunakan dalam pengolahan angka yang dipublish oleh microsoft. perangkat lunak ini digunakan dalam pengolahan data yand dipeloreh dari *quadcopter*.

4.2.4 Kebutuhan Komunikasi

Pada bagian ini dijelaskan mengenai kebutuhan komunikas sistem. Pada sistem ini terdapat dua jenis komunikasi yang digunakan yaitu komunikasi menggunakan Kabel USB 3.0 untuk menghubungkan antara sensor kinect dengan leptop dan komunikasi menggunakan Wi-Fi untuk menghubunkan leptop dengan *quadcopter*. Skema keseluruhan komunikasi pada sistem dapat dilihat pada gambar 4.5.

Gambar 4.5 Kebutuhan Komunikasi

Penjelasan detail pada kebutuhan komunikasi yang ditunjukkan pada gambar 4.5 adalah sebagai berikut:

a. Komunikasi antara Kinect dan Komputer.

Komunikasi dengan menggunakan kabel USB 3.0 digunakan Untuk menghubungkan kinect dengan komputer. Data yang diterima pada kinect akan diteruskan ke komputer dengan menggunakan bahasa pemograman javascript untuk diolah sesuai dengan kebutuhan sistem.

b. Komunikasi antara komputer dan *quadcopter*.

Data dari kinect yang diterima oleh komputer kemudian diolah untuk mengirimkan instruksi pergerakan pada *quadcopter*. Untuk dapat mengirimkan instruksi pada *quadcopter*, komputer harus sudah terhubung dengan *quadcopter*. Untuk menghubungkan komputer dengan *quadcopter*, maka digunakan komunikasi Wi-Fi. Metode komunikasi Wi-Fi yang digunakan dalam pengiriman data yaitu menggunakan protokol UDP. Data yang dikirim oleh komputer kepada *quadcopter* berupa AT Command.

4.2.5 Kebutuhan Fungsional

Kebutuhan fungsional merupakan sebuah pernyataan yang digunakan untuk memberikan rincian terhadap kebutuhan pada penelitian dan menjelaskan bagaimana input dari sistem yang dibuat nantinya dapat menghasilkan output yang diinginkan pada penelitian. Berikut kebutuhan fungsional dari sistem yang harus dipenuhi:

a. Sistem dapat menampilkan output dari pergerakan error yang dilakukan oleh pengguna. Sebagai contoh jika pengguna melakukan gerakan error, maka sistem akan menampilkan output berupa text pada console log.
b. Sistem dapat menampilkan output dari pergerakan *quadcopter* yang sedang berlangsung. Sebagai contoh jika pengguna melakukan instruksi pada *quadcopter* untuk bergerak ke kanan, maka sistem akan menampilkan output berupa text dengan tulisan ke kanan pada console log sebagai indikator bahwa instruksi pergerakan ke kanan pada *quadcopter* telah berhasil dijalankan.
c. Sistem dapat melakukan emergency state ketika user melakukan pergerakan error. Sebagai contoh jika pengguna melakukan pergerakan error, maka *quadcopter* akan masuk kedalam kondisi *hover*.
d. Sistem dapat menampilkan beberapa data navigasi pada *quadcopter* seperti ketinggian, kecepatan, kapasitas batterau , dan lain sebagainya.

4.2.6 Kebutuhan Non-Fungsional

Kebutuhan yang menjelaskan batasan terhadap kebutuhan perancangan sistem dijelaskan pada kebutuhan non fungsional. Berikut kebutuhan non fungsional pada sistem.

4.2.6.1 Karakteristik Pengguna

Sistem ini rancang untuk dapat digunakan oleh masyarakat umum dengan harapan mempermudah penggunaan *quadcopter* dalam bidang kontrol dengan menggunakan metode natural user interface. Dalam sistem ini user dengan gesture tubuh yang berbeda mampu untuk mengkontrol *quadcopter* dengan tidak memerlukan keahlian khusus.

4.2.6.2 Lingkungan Operasi

Persyaratan lingkungan operasi yang harus terpenuhi agar dapat mengendalikan *quadcopter* dengan optimal adalah sebagai berikut:

a. Jarak antara user dengan sonsor kinect berada pada 1.2 sampai 4 meter.

b. Untuk mengendalikan *quadcopter* dengan aman dan dapat terbang dengan kondisi optimal maka diperlukan sebuah ruangan dengan luas minimal 10m.

4.2.6.3 Asumsi dan Ketergantungan

Berikut beberapa asumsi dan ketergantungan pada sistem:

1. Sensor kinect dapat digunakan ketika sudah terhubung dengan *power supply*.
2. Pengendalian *quadcopter* baru dapat dilakukan ketika kinect sudah terhubung dengan komputer dan komputer sudah terhubung dengan *quadcopter*.
3. Sistem hanya akan memberikan instruksi pergerakan pada *quadcopter* ketika pengguna berada didepan sensor kinect. Jika tidak ada pengguna yang terdeteksi oleh sensor kinect, maka kinect tidak akan memberikan instruksi apapun kepada *quadcopter*.
4. Sistem pengendalian *quadcopter* menggunakan sensor kinect hanya dapat dilakukan pada komputer yang memiliki Kinect software development kit (SDK).
5. Keseluruan pergerakan *quadcopter* seperti kiri, kanan, maju, mundur dan lain sebagainya hanya dapat dijalankan setelah perintah takeoff pada *quadcopter* telah dijalankan.

4.2.6.4 Batasan Perancangan dan Implementasi

Batasan perancangan dan implementasi digunakan sebagai penentu batasan pada sistem dengan tujuan membuat penelitian ini menjadi spesifik dan berjalan sesuai harapan. Berikut adalah batasan-batasan perancangan dan implementasi pada sistem:

a. Jarak antara user dan sensor kinect hanya boleh antara 1.2 sampai 4 meter.

b. Jumlah user yang dapat mengendalikan *quadcopter* dengan menggunakan sensor kinect hanya berjumlah satu orang.

c. Gerakan dari pengguna yang digunakan untuk mengendalikan *quadcopter* berfokus pada data pergerakan pada kedua tangan.

d. Instruksi gerakan yang dikendalikan pada *quadcopter* yaitu gerakan *takeoff*, *landing*, *up*, *down*, *left*, *right*, *yaw*, dan *hover*.

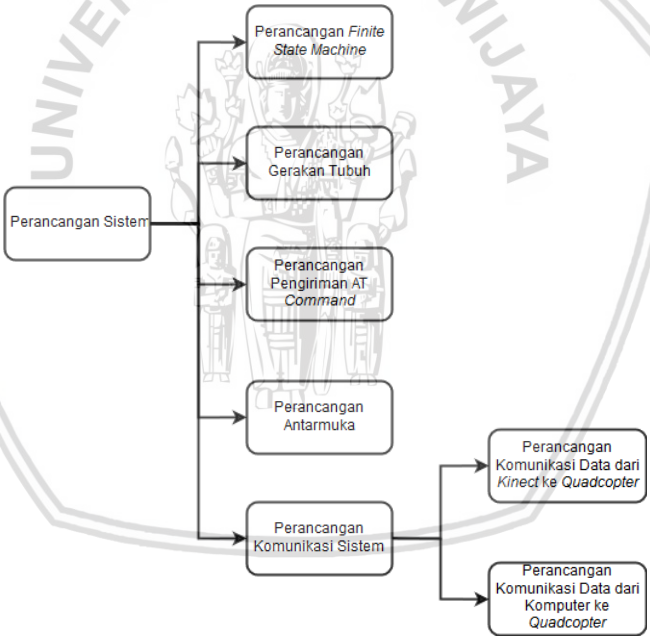
e. Data navigasi pada *quadcopter* yang ditampilkan kepada user adalah sudut *roll*, sudut *pitch*, sudut *yaw* ,ketinggian, baterai, pergerakan *error*, dan status pergerakan pada *quadcopter* yang sedang terjadi.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

Setelah dilakukan analisis kebutuhan pada tahap sebelumnya, maka dilanjutnya dengan tahap perancangan dan implementasi pada sistem yang dijelaskan pada bab ini. Pada bab ini dibagi menjadi menjadi dua bagian yaitu tahap perancangan sistem dan tahap implementasi sistem. Pada tahap perancangan sistem dilakukan perancangan terhadap setiap komponen penyusun sistem. Kemudian dilakukan implementasi terhadap rancangan sistem yang telah dibagun sebelumnya pada tahap implementasi sistem.

5.1 Perancangan Sistem

Perancangan sistem merupakan tahap mengolah informasi dari analisis kebutuhan sistem agar informasi tersebut dapat digunakan pada tahap implementasi sistem. Untuk memudahkan dalam melakukan perancangan sistem, maka dilakukan pengelompokan perancangan sesuai dengan diagram blok sistem pada bab sebelumnya. Berikut penjelasan perancangan sistem yang diperlihatkan dalam bentuk skema pada gambar 5.1.



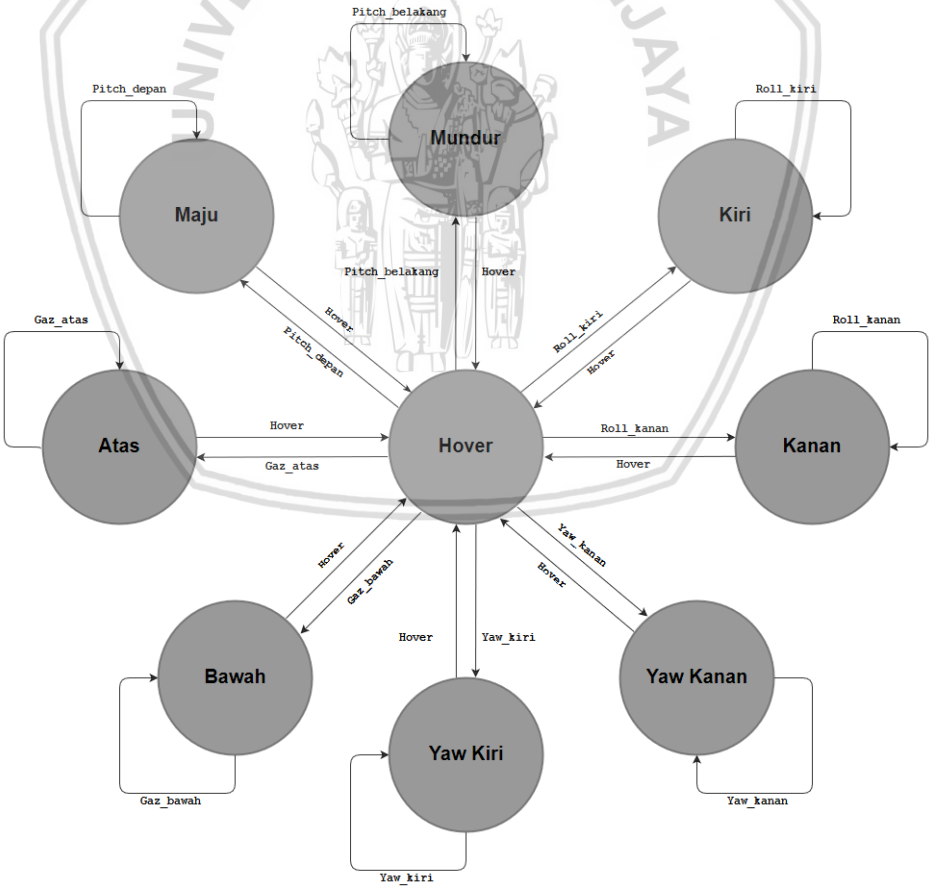
Gambar 5.1 Alur Perancangan Sistem

Berdasarkan gambar 5.1 dapat dilihat bahwa perancangan sistem dibagi menjadi lima bagian, yaitu perancangan *finite state machine*, perancangan gerakan tubuh, perancangan pengiriman AT *command*, perancangan antarmuka pengguna, dan perancangan komunikasi sistem. Pada perancangan komunikasi sistem dibagi menjadi bagian yaitu perancangan komunikasi data dari *kinect* ke komputer dan komunikasi data dari komputer ke *quadcopter*. Berikut penjelasan lebih detail mengenai perancangan sistem yang berdasarkan skema pada gambar 5.1.

5.1.1 Perancangan *Finite State Machine*

Pada bagian ini dijelaskan mengenai perancangan sistem yang dibangun dengan menggunakan metode *finite state machine*. Metode *finite state machine* digunakan dengan tujuan untuk membatasi instruksi-instruksi yang diberikan oleh pengguna dalam sistem yang dibangun yaitu pengendalian *quadcopter* dengan menggunakan pergerakan tubuh dihadapan sensor tubuh.

Jenis metode *finite state machine* yang digunakan pada penelitian ini adalah *transducers*. Hal ini dikarenakan sistem *finite state machine* yang dirancang akan menghasilkan *output* berdasarkan input yang diberikan oleh pengguna tanpa memiliki sebuah *final state*. Penggunaan metode *finite state machine* menyebabkan *output* sistem yaitu pergerakan *quadcopter* tidak hanya tergantung dari inputan pergerakan tubuh dari pengguna, tetapi juga tergantung dari kondisi pada *quadcopter* pada saat pengendalian *quadcopter* sedang dilakukan. Perancangan metode *finite state machine* pada penelitian ini diperlihatkan pada gambar 5.2 yang dimana dideklarasikan *intial state* yang dimulai dari hover, simbol input yang terdiri dari *hover*, *gaz* bawah, *gaz* atas, *pitch* depan, *pitch* belakang, *roll* kiri, *roll* kanan, *yaw* kiri, *yaw* kanan dan himpunan terbatas state yang terdiri dari bawah, atas, maju, mundur, kiri , kanan, *yaw* kanan, dan *yaw* kiri.



Gambar 5.2 Rancangan *state diagram finite state machine*

Berdasarkan pada rancangan *finite state machine* tersebut, pengguna hanya dapat melakukan perpindahan dari suatu instruksi gerakan ke instruksi gerakan lain

ketika pengguna melakukan gerakan hover diantara perpindahan gerakan tersebut. Sistem yang dirancang juga hanya memperbolehkan pengguna untuk melakukan suatu instruksi navigasi pada *quadcopter* apabila instruksi yang diberikan tersebut merupakan instruksi terakhir yang diberikan kepada *quadcopter* pada saat *quadcopter* belum memasuki *state hover*. Ketika pengguna tidak mengikuti pada aturan *finite state machine* yang telah rancang sebelumnya, maka *quadcopter* akan mengeksekusi instruksi hover secara otomatis. Sebagai contoh ketika pengguna melakukan pergerakan ke kanan kemudian melakukan pergerakan kekiri tanpa melakukan pergerakan hover terlebih dahulu, maka *quadcopter* tidak akan mengeksekusi intruksi pergerakan yang telah diberikan oleh pengguna yaitu gerakan ke kanan dan *quadcopter* akan melakukan instruksi gerakan hover secara otomatis. Tujuan dari *quadcopter* dirancang untuk melakukan gerakan hover ketika pengguna melakukan dua instruksi pergerakan tanpa melalui gerakan hover yaitu mencegah *quadcopter* untuk melakukan pergerakan yang membahayakan lingkungan sekitar dan *quadcopter* memasuki keadaan paling aman dari lingkungan sekitar ketika *quadcopter* tersebut berada dalam keadaan hover. Untuk keseluruhan dari transisi state yang dirancang pada sistem ini dapat dilihat pada tabel 5.1. Tabel 5.1 merupakan keseluruhan aturan state yang diperbolehkan oleh sistem.

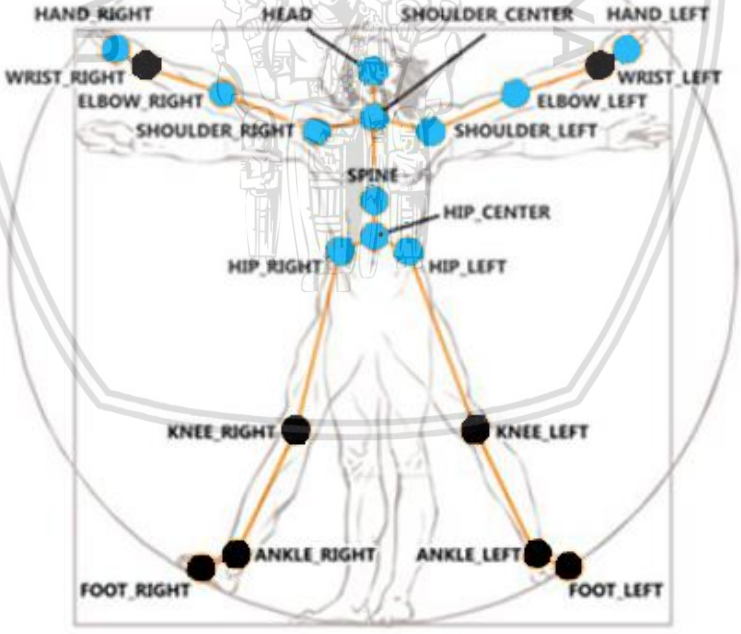
Tabel 5.1 Aturan *state* yang diperbolehkan oleh sistem

Current State	*Input State*	*Next State*
Hover	*Gaz* bawah	*Gaz* bawah
Hover	*Gaz* atas	*Gaz* atas
Hover	*Pitch* depan	*Pitch* depan
Hover	*Pitch* belakang	*Pitch* belakang
Hover	*Roll* kiri	*Roll* kiri
Hover	*Roll* kanan	*Roll* kanan
Hover	*Yaw* kanan	*Yaw* kanan
Hover	*Yaw* kiri	*Yaw* kiri
Gaz bawah	*Hover*	*Hover*
Gaz atas	*Hover*	*Hover*
Pitch depan	*Hover*	*Hover*
Pitch belakang	*Hover*	*Hover*
Roll kiri	*Hover*	*Hover*
Roll kanan	*Hover*	*Hover*
Yaw kanan	*Hover*	*Hover*
Yaw kiri	*Hover*	*Hover*
Gaz bawah	*Gaz* bawah	*Gaz* bawah

Gaz atas	*Gaz* atas	*Gaz* atas
Pitch depan	*Pitch* depan	*Pitch* depan
Pitch belakang	*Pitch* belakang	*Pitch* belakang
Roll kiri	*Roll* kiri	*Roll* kiri
Roll kanan	*Roll* kanan	*Roll* kanan
Yaw kanan	*Yaw* kanan	*Yaw* kanan
Yaw kiri	*Yaw* kiri	*Yaw* kiri

5.1.2 Perancangan Gerakan Tubuh

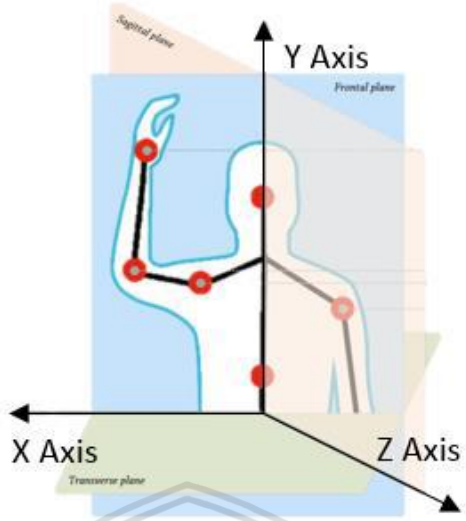
Data joint user yang didapat melalui sceleton tracking pada sensor kinect akan digunakan dalam perancangan gerakan tubuh. Penelitian ini berfokus pada joint bagian atas tubuh, yaitu dimulai dari data joint pada kepala hingga data joint pada pinggang. Data Joint tubuh yang dipakai pada penelitian ini ditandai dengan warna hijau. Sedangkan Data joint yang tidak digunakan ditandai dengan warna hitam yang diperlihatkan pada gambar 5.3.



Gambar 5.3 Data *sceleton tracking* yang digunakan

Sumber : (Microsoft, 2017)

Setiap data joint yang dipeloreh melalui sceleton tracking memiliki posisi data tiga dimensi yang disimpan dalam koordinat X, Y, dan Z seperti pada gambar 5.4.



Gambar 5.4 Koordinat pada *Kinect*

Nilai koordinat X, Y, dan Z pada masing-masing joint didapatkan berdasarkan dari kondisi pengguna sebagai berikut:

1. Nilai data joint pada koordinat Z didapat dari posisi pengguna tepat berada didepan kamera kinect. Jika pengguna menjauh dari kamera kinect, maka nilai joint pada koordinat Z akan semakin besar. Sedangkan ketika pengguna mendekat ke arah kamera kinect, maka nilai joint pada koordinat Z akan semakin kecil.
2. Untuk koordinat X posisi pengguna diukur dari kanan ke kiri dari kamera kinect. Jika pengguna menggunakan suatu joint ke arah kiri, maka nilai joint pada koordinat X akan semakin besar. Sedangkan ketika pengguna menggunakan suatu joint ke arah kanan, maka nilai joint pada koordinat X akan semakin kecil.
3. Untuk nilai joint data pada koordinat Y didapat dari posisi pengguna yang diukur dari atas ke bawah dari kamera kinect. Jika pengguna menggunakan suatu joint ke arah atas, maka nilai joint pada koordinat Y akan semakin besar. sedangkan ketika pengguna menggunakan suatu joint ke arah bawah, maka nilai joint pada koordinat Y akan semakin kecil.

Sistem pengendalian pergerakan *quadcopter* yang dirancang berfokus terhadap pergerakan pada joint tangan kanan dan tangan kiri. Tidak semua pergerakan yang dirancang dideklarasikan dengan tiga posisi koordinat dikarenakan pada pergerakan tertentu. Hal ini dikarenakan pengkondisikan posisi pergerakan pada pengguna agar sesuai dengan perancangan pergerakan yang diinginkan sudah bisa dilakukan dengan hanya menggunaakan satu posisi koordinat. Pergerakan yang diguanakan pada sistem kendali dan navigasi *quadcopter* adalah sebagai berikut:

a. Gerakan merentangkan kedua tangan.

   Gerakan merentangkan kedua tangan diguanakan untuk melakukan instruksi gerakan hover pada *quadcopter*. Joint pada tangan kanan sejajar dengan joint pada tangan kiri seperti pada gambar 5.5. Nilai joint pada pergerakan ini ditinjau

dari tiga koordinat yaitu X, Y, dan Z. Pada koordinat X, hasil pengurangan dari joint tangan kanan dengan tangan kiri harus lebih besar dari 0.7 dengan tujuan gerakan hover tidak dapat dilakukan oleh pengguna ketika joint pada tangan kanan tidak sejajar dengan joint pada tangan kiri. Untuk koordinat Y, joint pada kepala dan dada berada diantara joint tangan kanan dan tangan kiri. Untuk koordinat Z, nilai joint pada tangan kanan dan tangan kiri harus bernilai 0 jika dikurangi dengan nilai joint pada leher dengan tujuan sistem tidak akan memberikan instruksi hover pada *quadcopter* ketika gerakan pengguna tidak sesuai dengan gerakan yang ada pada gambar 5.5. Keseluruhan kondisi joint sceleton yang harus dipenuhi oleh pengguna agar dapat melakukan gerakan hover adalah sebagai berikut:

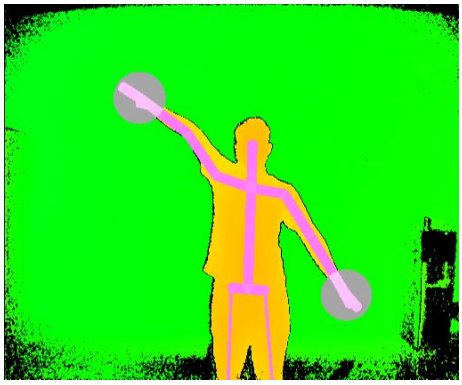
1. Koordinat X :
   Tangan kanan – tangan kiri > 0.7
2. Koordinat Y :
   kepala > tangan kanan, tangan kanan > dada, kepala > tangan kiri, tangan kiri > dada.
3. Koordinat Z :
   -0.1 < tangan kanan - leher < 0.1, -0.1 < tangan_kiri - leher < 0.1

Gambar 5.5 Gerakan hover

b. Gerakan tangan kanan lebih besar daripada tangan kiri.

Gerakan tangan kanan lebih tinggi daripada tangan kiri digunakan untuk melakukan instruksi gerakan ke kanan pada *quadcopter*. Posisi joint pada tangan kanan dan kiri dikondisikan sehingga menyerupai pada gambar 5.6. Nilai joint pada pergerakan ini ditinjau dari koordinat Y, yaitu posisi joint pada tangan kanan lebih besar daripada joint kepala dan posisi joint pada tangan kiri lebih kecil daripada joint dada. Keseluruhan kondisi joint sceleton yang harus dipenuhi oleh pengguna agar dapat melakukan gerakan ke kanan adalah sebagai berikut:

1. Koordinat Y :
   tangan kiri < siku kiri, siku kiri < bahu kiri, bahu kiri < bahu Kanan, bahu Kanan < siku Kanan, siku Kanan < tangan Kanan, dan tangan Kanan > kepala.
2. Koordinat Z :
   -0.1 < tangan kanan - leher < 0.1 dan -0.1 < tangan kiri - leher < 0.1.

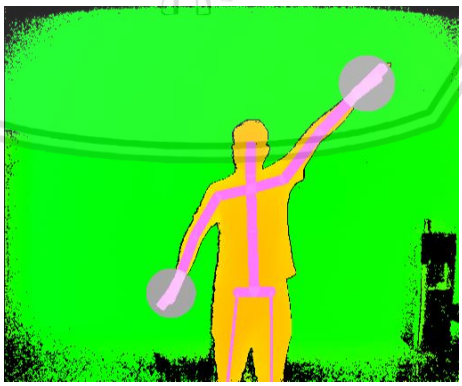


Gambar 5.6 Gerakan ke kanan

c. Gerakan tangan kiri lebih besar daripada tangan kanan.

Gerakan tangan kanan lebih tinggi daripada tangan kiri digunakan untuk melakukan instruksi gerakan ke kiri pada *quadcopter*. Posisi joint pada tangan kanan dan tangan kiri dikondisikan sehinnga menyerupai pada gambar 5.7. Nilai joint pada pergerakan ini ditinjau dari koordinat Y, yaitu posisi joint pada tangan kiri lebih besar daripada joint kepala dan posisi joint pada tangan kanan lebih kecil daripada joint dada. Keseluruhan kondisi joint sceleton yang harus dipenuhi oleh pengguna agar dapat melakukan gerakan ke kiri adalah sebagai berikut:

1. Koordinat Y :
   Tangan kiri > siku kiri, siku kiri > bahu kiri, bahu_kiri > bahu kanan, bahu kanan > siku kanan, siku kanan > tangan kanan, tangan kiri > kepala.
2. Koordinat Z :
   -0.1 < tangan kanan - leher < 0.1 dan -0.1 < tangan kiri - leher < 0.



Gambar 5.7 Gerakan ke kiri

d. Gerakan tangan kanan dan kiri diatas kepala.

Gerakan tangan kanan dan kiri diatas kepala digunakan untuk melakukan instruksi gerakan ke atas pada *quadcopter*. Posisi joint pada tangan kanan dan tangan kiri dikondisikan sehinnga menyerupai pada gambar 5.8. Nilai joint pada pergerakan ini ditinjau dari koordinat Y, yaitu nilai joint pada tangan kiri dan

kanan lebih besar daripada joint dada. Keseluruhan kondisi joint sceleton yang harus dipenuhi oleh pengguna agar dapat melakukan gerakan ke atas adalah sebagai berikut:

1. Koordinat Y :
   Tangan kanan > kepala, tangan kiri > kepala, 0 < tangan kanan, 0.4 > tangan kanan, 0 < tangan kiri, 0.4 > tangan kiri.
2. Koordinat Z :
   -0.1 < tangan kanan - leher < 0.1 dan -0.1 < tangan kiri - leher < 0.

Gambar 5.8 Gerakan ke atas

e. Gerakan tangan kanan dan kiri dibawah kepala.

Gerakan tangan kanan dan kiri diatas kepala digunakan untuk melakukan instruksi gerakan ke bawah pada *quadcopter*. Posisi joint pada tangan kanan dan tangan kiri dikondisikan sehinnga menyerupai pada gambar 5.9. Nilai joint pada pergerakan ini ditinjau dari koordinat Y, yaitu nilai joint pada tangan kiri dan kanan lebih kecil daripada joint dada. Keseluruhan kondisi joint sceleton yang harus dipenuhi oleh pengguna agar dapat melakukan gerakan ke bawah adalah sebagai berikut:

1. Koordinat Y :
   Tangan kanan < dada, tangan kiri < dada, tangan kanan > pinggang, tangan kiri > pinggang.
2. Koordinat Z :
   Tangan kanan < dada, tangan kiri < dada, tangan kanan – pinggang < -0.2, tangan kiri – pinggang < -0.2.

Gambar 5.9 Gerakan ke bawah

f. Gerakan tangan kanan dan kiri didepan bahu

Gerakan tangan kanan dan kiri didepan bahu digunakan untuk melakukan instruksi gerakan ke depan pada *quadcopter*. Posisi joint pada tangan kanan dan tangan kiri dikondisikan sehinnga menyerupai pada gambar 5.10. Nilai joint pada pergerakan ini ditinjau dari koordinat X dan Y. Pada koordinat X, hasil pengurangan dari joint tangan kanan dengan tangan kiri harus lebih kecil dari 0.3 dengan tujuan gerakan ke depan tidak dapat dilakukan oleh pengguna ketika joint pada tangan kanan dan tangan kiri tidak sejajar dan tidak berada didepan joint dada jika pantau dari koordinat X kamera kinect. Untuk koordinat Y, nilai joint pada kepala lebih kecil daripada joint tangan kanan dan tangan kiri serta nilai joint pada pinggang lebih kecil daripada joint tangan kanan dan kiri. Keseluruhan kondisi joint sceleton yang harus dipenuhi oleh pengguna agar dapat melakukan gerakan kedepan adalah sebagai berikut:

1. Koordinat X :
   Tangan kanan – tangan kiri < 0.3.
2. Koordinat Y :
   Tangan kanan < kepala, tangan kanan > pinggang.

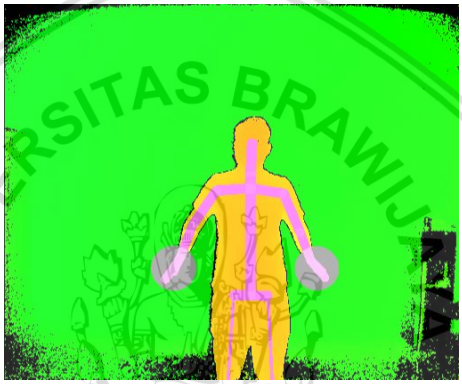
Gambar 5.10 Gerakan ke depan

g. Gerakan tangan kanan dan kiri dibelakang bahu

Gerakan tangan kanan dan kiri dibelakang bahu digunakan untuk melakukan instruksi gerakan ke belakang pada *quadcopter*. Posisi joint pada tangan kanan

dan tangan kiri dikondisikan sehinnga menyerupai pada gambar 5.11. Nilai joint pada pergerakan ini ditinjau dari koordinat Y dan Z. Pada koordinat Y, nilai joint pada dada lebih besar daripada joint tangan kanan dan kiri serta nilai joint pada pinggang lebih kecil daripada joint tangan kanan dan kiri. Untuk koordinat Z, nilai joint pada dada lebih kecil daripada joint tangan kanan dan tangan kiri. Keseluruhan kondisi joint sceleton yang harus dipenuhi oleh pengguna agar dapat melakukan gerakan ke belakang adalah sebagai berikut:

1. Koordinat Y :
   Tangan kananY < dada, tangan kiri < dada, tangan kanan > pinggang, tangan kiri > pinggang.
2. Koordinat Z :
   Tangan kanan > dada, tangan kiri > dada.

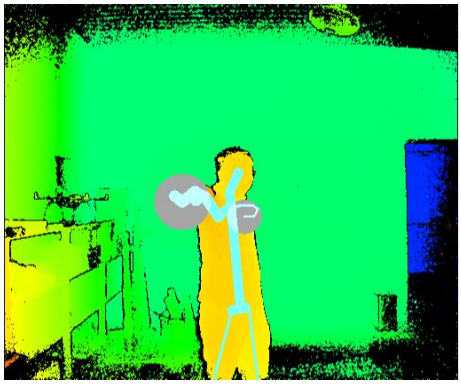


Gambar 5.11 Gerakan Ke belakang

h. Gerakan tangan kanan didepan dan tangan kiri dibelakang

Gerakan tangan kanan didepan dan kiri dibelakang digunakan untuk melakukan instruksi gerakan berputar searah jarum jam pada *quadcopter*. Posisi joint pada tangan kanan dan tangan kiri dikondisikan sehinnga menyerupai pada gambar 5.12. Nilai joint pada pergerakan ini ditinjau dari koordinat Y dan Z. Pada koordinat Y, nilai joint pada kepala lebih besar daripada joint tangan kanan dan kiri serta nilai joint pada dada lebih kecil daripada joint tangan kanan dan kiri. Untuk koordinat Z, nilai joint pada tangan kanan lebih kecil daripada joint kepala dan nilai joint pada tangan kiri lebih besar daripada joint dada. Keseluruhan kondisi joint sceleton yang harus dipenuhi oleh pengguna agar dapat melakukan gerakan berputar searah jarum jam adalah sebagai berikut:

1. Koordinat Y :
   kepala > tangan kanan, tangan kanan > dada, kepala > tangan kiri, tangan kiri > dada.
2. Koordinat Z :
   Tangan kanan < kepala, tangan kiri > dada.

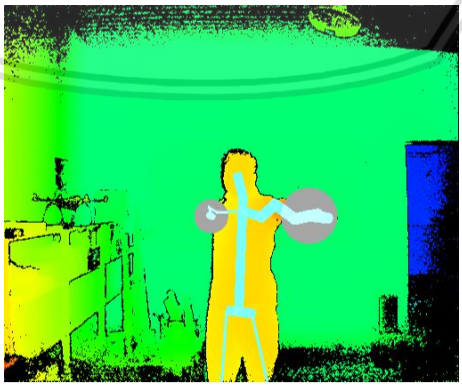


Gambar 5.12 Gerakan Yaw Ke kanan

i. Gerakan tangan kiri dedepan dan tangan kanan dibelakang

Gerakan tangan kiri didepan dan kanan dibelakang digunakan untuk melakukan instruksi gerakan berputar berlawanan arah jarum jam pada *quadcopter*. Posisi joint pada tangan kanan dan tangan kiri dikondisikan sehinnga menyerupai pada gambar 5.13. Nilai joint pada pergerakan ini ditinjau dari koordinat Y dan Z. Pada koordinat Y, nilai joint pada kepala lebih besar daripada joint tangan kanan dan kiri serta nilai joint pada dada lebih kecil daripada joint tangan kanan dan kiri. Untuk koordinat Z, nilai joint pada tangan kiri lebih kecil daripada joint kepala dan nilai joint pada tangan kanan lebih besar daripada joint dada. Keseluruhan kondisi joint sceleton yang harus dipenuhi oleh pengguna agar dapat melakukan gerakan berputar searah jarum jam adalah sebagai berikut:

1. Koordinat Y :
   Kepala > tangan kanan, tangan kanan > dada, kepala > tangan kiri, tangankiri > dada.
2. Koordinat Z :
   Tangan kiri < kepala, tangan kanan > dada.



Gambar 5.13 Gerakan Yaw Ke kiri

j. Gerakan lasso tangan kanan

Gerakan lasso tangan kanan digunakan untuk melakukan instruksi takeoff pada *quadcopter*. Gerakan ini dilakukan dengan cara mengangkat satu jari pada tangan kanan dihadapan sensor kamera kinect seperti pada gambar 5.14.

Gambar 5.14 Gerakan Lasso Kanan

k. Gerakan lasso tangan kiri

Gerakan lasso tangan kiri digunakan untuk melakukan instruksi landing pada *quadcopter*. Gerakan ini dilakukan dengan cara mengangkat satu jari pada tangan kiri dihadapan sensor kamera kinect seperti pada gambar 5.15.

Gambar 5.15 Gerakan Lasso Kiri

5.1.3 Perancangan Pengiriman AT Command

AT command digunakan untuk menghubungkan komunikasi antara *quadcopter* dengan komputer. AT command dikirim dari komputer ke *quadcopter* dengan menggunanakan komunikasi wi-fi dengan protokol UDP yang menggunakan port 5556. pengiriman AT command dilakukan setiap 30ms sekali agar pengendalian *quadcopter* dapat dilakukan dengan baik. AT command yang dikirim ke komputer berbentuk text string yang berguna untuk mengontrol *quadcopter*. Text string tersebut di *encode* sebagai 8-bit ASCII character, diikuti dengan sebuah karakter berupa *line feed* (LF). AT command yang disediakan pada SDK *AR.drone* dapat dilihat pada tabel berikut:

Tabel 5.2 Daftar nama AT *command* pada *AR.Drone*

Nama command	Argumen	Deskripsi
AT*REF	Input	Takeoff/landing/emergeny
AT*PCMD	Flags, roll, pitch, gaz, yaw	Pergerakan *quadcopter*
AT*FTRIM	-	Penyetelan sudut horizontal *quadcopter*
AT*CONFIG	Key, value	Konfigurasi
AT*CONFIG_IDS	Session, users, application ids	Identifier untuk AT*CONFIG
AT*CALIB	Device number	Kalibrasi magnetometer

Pada peneltian ini , AT command yang digunakan adalah AT*PCMD dan AT*REF. AT*PCMD diguanakan untuk melakukan instruksi *roll*, *pitch*, *gaz*, dan *yaw* pada *quadcopter*. Sedangkan AT*REF diguanakan untuk melakukan instruksi *takeoff* dan *landing* pada *quadcopter*. Aturan penulisan text string pada AT*PCMD dan AT*REF dijelaskan sebagai berikut:

a. AT*PCMD.

AT*PCMD digunakan ketika *quadcopter* melakukan pergerakan *roll*, pitch, gaz, dan yaw. Format *syntax* yang ada pada AT*PCMD adalah sebagai berikut:

AT*PCMD = [sequence number] , [Flag bit-field], [Drone Roll], [Drone Pitch], [Drone Gaz], [Drone Yaw] <LF>

Berikut adalah penjelasan syntax pada AT*PCMD:

1. Kegunaan bit nol.

   Bit ke nol pada syntax AT*PCMD berisi argument sequence number. Sequence number selalu mengirimkan nilai satu ketika instruksi pertama dikirimkan ke *quadcopter*. Ketika suatu instruksi selanjutnya dikirimkan, nilai sequence number akan bertambah sebanyak satu dari nilai sequence number sebelumnya. Sequence number akan direset ke satu ketika client melakukan pemutusan sambungan dari AT command pada port UDP.

2. Kegunaan bit pertama.

   Bit pertama pada *syntax* AT*PCMD berisi argument flag bit field 32 bit yang memiliki fungsi untuk mengontrol perintah progresive pada *quadcopter*. Perintah progresive pada *quadcopter* meliputi pergerakan *roll*, *pitch*, *yaw*, dan *gaz*. Nilai *flag* pada bit ke-nol bernilai 0 pada saat *quadcopter* masuk ke mode *hover*. Untuk mengaktifkan mode perintah progesive pada *quadcopter*, maka nilai pada bit

kenol diatur dengan nilai 1. Pada saat bit pertama diatur dengan nilai 1, maka sistem akan mengaktifkan mode kombinasi pada gerakan yaw. Sedangkan pada saat bit kesatu diatur dengan nilai 0, maka sistem akan menonaktifkan mode kombinasi pada pergerkaan yaw. Nilai pada bit 2 hingga 31 diatur dengan nilai 0 karena bit tersebut tidak dipergunakan oleh sistem.

3. Kegunaan bit kedua.

   Bit kedua pada syntax AT*PCMD berisi argument drone *roll* yang bertujuan untuk melakukan pergerakan kekanan dan kekiri pada *quadcopter*. Pada argument drone *rol*l memiliki nilai floating point atara -1 hingga 1. nilai negative pada argument drone roll menyebabkan *quadcopter* bergerak kearah kiri. Sedangkan nilai positif pada argument drone roll menyebabkan *quadcopter* bergerak kearah kanan.

4. Kegunaan bit ketiga.

   Bit ketiga pada *syntax* AT*PCMD berisi argument drone *pitch* yang bertujuan untuk melakukan pergerakan kedepan dan kebelakang pada *quadcopter*. Pada argument drone roll memiliki nilai floating point atara -1 hingga 1. nilai negative pada argument drone pitch menyebabkan *quadcopter* bergerak kearah depan. Sedangkan nilai positif pada argument drone pitch menyebabkan *quadcopter* bergerak kearah belakang.

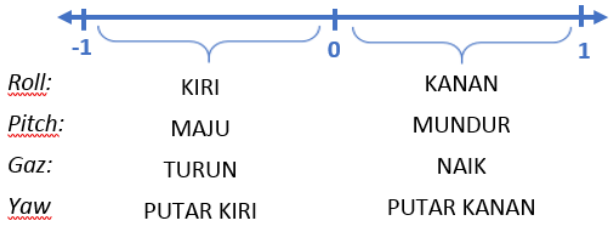
5. Kegunaan bit keempat.

   Bit kelima pada *syntax* AT*PCMD berisi argument drone gaz yang bertujuan untuk melakukan pergerakan keatas dan kebawah pada *quadcopter*. Pada argument drone gaz memiliki nilai floating point atara -1 hingga 1. nilai negative pada argument drone gaz menyebabkan *quadcopter* bergerak kearah bawah. Sedangkan nilai positif pada argument drone gaz menyebabkan *quadcopter* bergerak kearah atas.

6. Kegunaan bit kelima.

   Bit keenam pada *syntax* AT*PCMD berisi argument drone yaw yang bertujuan untuk melakukan pergerakan berputar kekanan dan berputar kekanan pada *quadcopter*. Pada argument drone roll memiliki nilai floating point atara -1 hingga 1. nilai negative pada argument drone yaw menyebabkan *quadcopter* bergerak berputar kekiri. Sedangkan nilai positif pada argument drone yaw menyebabkan *quadcopter* bergerak berputar kekanan.

   Pengaruh pemberian nilai floating negatif dan positif pada setiap argument secara keseluruhan dapat dilihat pada gambar 5.16.



Gambar 5.16 Arah gerakan *quadcopter* berdasarkan *floating point*

b. AT*REF.

AT*PCMD digunakan ketika *quadcopter* melakukan pergerakan takeoff dan landing. Format syntax yang ada pada AT*PCMD adalah sebagai berikut:

AT*REF = [sequence number] , [Input], <LF>

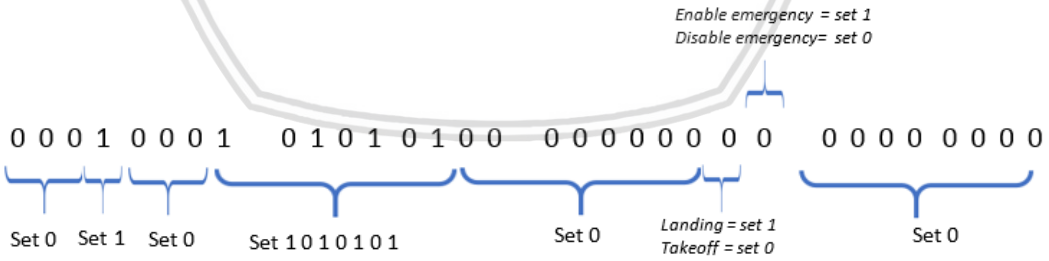
Berikut adalah penjelasan syntax pada AT*REF:

1. Kegunaan bit nol.

   Bit ke nol pada syntax AT*REF berisi argument sequence number. Sequence number selalu mengirimkan nilai satu ketika instruksi pertama dikirimkan ke *quadcopter*. Ketika suatu instruksi selanjutnya dikirimkan, nilai sequence number akan bertambah sebanyak satu dari nilai sequence number sebelumnya. Sequence number akan direset ke satu ketika client melakukan pemutusan sambungan dari AT command pada port UDP.

2. Kegunaan bit pertama.

   Bit PERTAMA pada syntax AT*REF berisi argument input yang memiliki fungsi untuk mengontrol pergerakan takeoff , takeoff, dan emergency landing. Argument input pada AT*REF memiliki format penulisan berupa 32 *bit field integer*. Aturan penulisan bit flag pada AT*REF dapat dilihat pada gambar 5.17.



Gambar 5.17 Pengaturan nilai pada argument input

Bit 0 hingga 7 diatur secara default dengan nilai 0. Pada saat quacopter berada pada posisi darurat, maka nilai flag pada bit ke 8 bernilai 1 yang dimana sistem akan menonaktifkan seluruh mesin pada *quadcopter*. Bit 9 diatur dengan nilai 1 untuk mengaktifkan mode *takeoff* dan diatur dengan nilai 0 untuk mengaktifkan mode landing pada *quadcopter*. Bit 10 hingga 17 diatur secara default dengan nilai 0. Pada bit 18, 20, 22, 24, 28 diatur secara default dengan nilai 1. Terakhir bit 19, 21, 23, 25, 26, 27, 29, 30, 31 diatur secara default dengan nilai 0. Pada program

keseluruhan nilai binary pada bit flag AT*REF diubah ke dalam bentuk desimal sehingga format *syntax* pada AT*REF dapat dideklarasikan sebagai berikut:

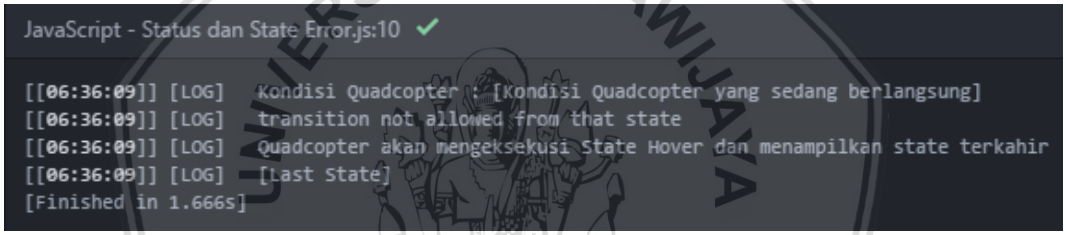
Takeoff : AT*REF=1,290718208 <LF>

Landing : AT*REF=1,290717696 <LF>

Emergency Mode : AT*REF=1,290717952 <LF>

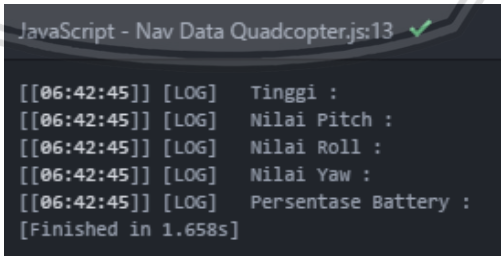
5.1.4 Perancangan Antarmuka Pengguna

Pada subbab ini dijelaskan perancangan tampilan yang digunakan oleh sistem. Tampilan antar muka pada sistem ini menggunakan ATOM Text Editor. Pada sistem ini dirancang dua buah *Command Line Inteface*. Pada *Command Line Interface* pertama dirancang untuk menampilkan status navigasi *quadcopter* pada saat qoadcopter tersebut sedang dikendalikan, menampilkan data pada saat pengguna melakukan input pergerakan *error* atau pergerakan yang tidak terdapat pada perancangan sistem, dan menampilkan status koneksi yang terjadi pada kinect. Tampilan rancangan *Command line Interface* pertama yang dapat dilihat dapat gambar 5.18.



Gambar 5.18 Rancangan Tampilan Status Navigasi dan Keadaan Error

Sedangkan *command line interface* yang kedua dirancang untuk menampilkan data kondisi batterai pada *quadcopter*, menampilkan data ketinggan pada *quadcopter*, dan menampilkan data berupa nilai sudut pada *quadcopter* yaitu berupa nilai sudut pitch, nilai sudut roll, dan nilai sudut yaw. Tampilan rancangan *Command line Interface* pertama yang dapat dilihat dapat gambar 5.19.

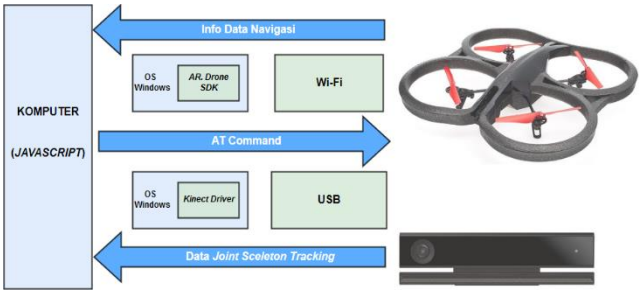


Gambar 5.19 Rancangan Tampilan Data Navigasi Pada *Quadcopter*

5.1.5 Perancangan Komunikasi Sistem

Alur keseluruhan pertukaran data pada sistem ini dapat dilihat pada gambar 5.20. *Sensor sceleton tracking* yang ada pada *kinect* akan menghasilkan data *joint* dari pengguna. Kemudian data joint tersebut dikirimkan ke komputer melalui kabel USB 3.0. Data joint tersebut kemudian diolah dikomputer dengan memanfaatkan *software development kit* yang telah disediakan oleh pihak *microsoft* dengan

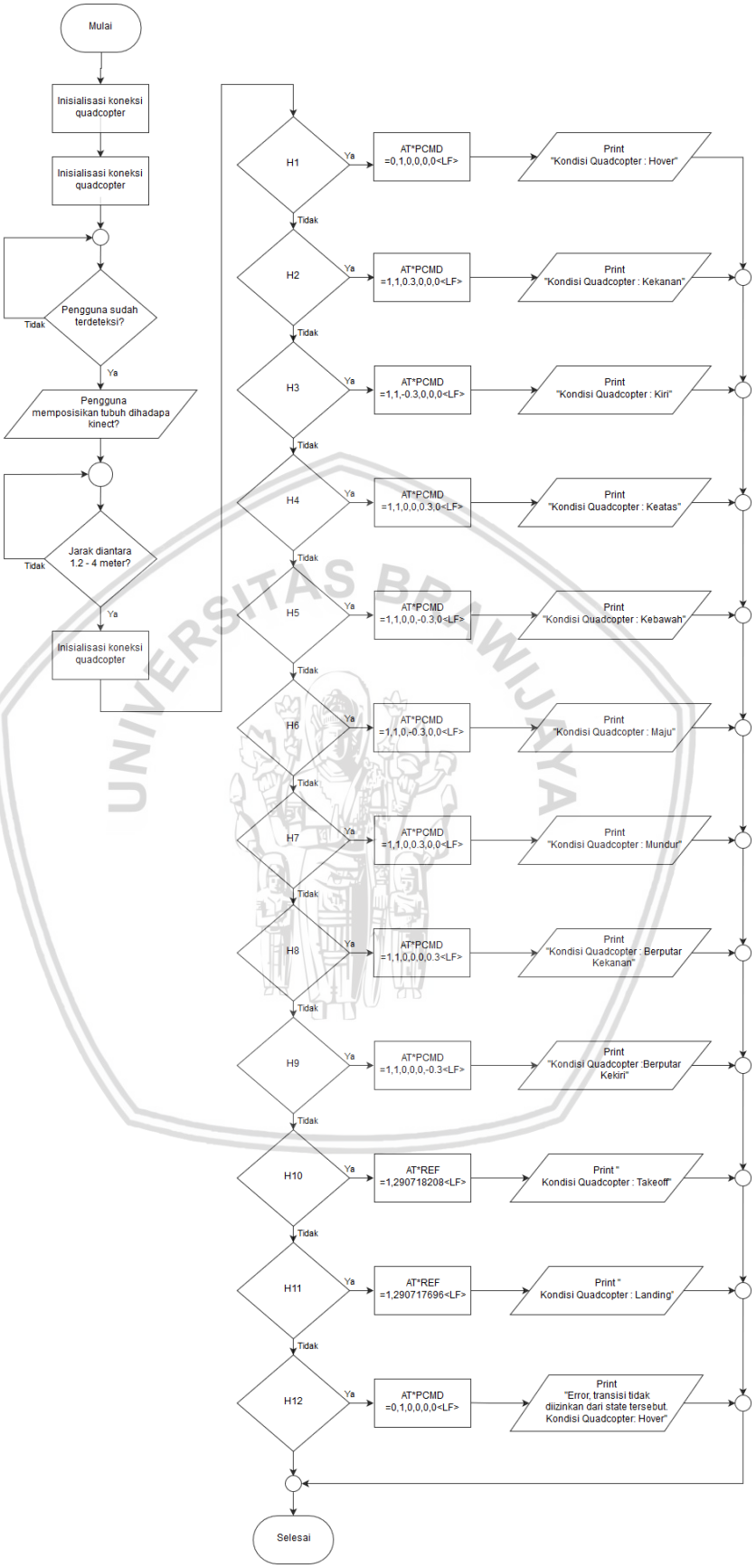
menggunakan bahasa pemograman *javascript*. Program yang dirancang memanfaatkan data pergerakan tubuh dari pengguna yang akan menghasilkan serangkaian instruksi pada *quadcopter*.



Gambar 5.20 Alur pertukaran data pada sistem

Kemudian komunikasi antara komputer dengan *quadcopter* menggunakan jaringan *wi-fi*. Metode komunikasi yang dipakai antara komputer dengan *quadcopter* adalah menggunakan AT *Command* yang dikirim melalui protokol jaringan UDP. Alur kerja keseluruhan komunikasi pada sistem ini diperlihatkan dengan menggunakan *flowhchart* pada gambar 5.21.

Untuk dapat melakukan pertukaran data seperti yang diperlihatkan pada gambar 5.20, maka langkah pertama yang dilakukan adalah melakukan inisialisasi koneksi antara *quadcopter* dan Komputer. Penghubungan koneksi antara komputer dan quadcopter dilakukan melalui komunikasi *Wi-Fi*. Selanjutnya dilakukan proses inisialisasi antara Kinect dan komputer. Untuk dapat melakukan pertukaran data antara komputer dengan kinect, maka digunakan komunikasi dengan menggunakan kabel USB versi 3.0. Setelah itu dilakukan proses pendeteksian gerakan tubuh oleh sensor *kinect*. Pada proses ini *kinect* akan mendeteksi apakah pengguna berada pada posisi di hadapan sensor pergerakan tubuh. Jika tidak ada pengguna didepan kinect maka proses akan melakukan perulangan hingga terdapat pengguna yang berdiri didepan kinect. Ketika sudah terdeteksi ada pengguna didepan kinect, maka akan dilakukan pengecekan lagi apakah jarak pengguna berada antara 1 meter hingga 4 meter. Ketika pengguna sudah berada pada jarak tersebut, maka sensor *kinect* siap digunakan. Kemudian ketika pengguna memberikan sebuah instruksi pada *quadcopter* dengan menggunakan pergerakan tubuh, maka komputer akan mengirimkan perintah AT *command* dengan *floating point* yang dimana nilai pada *floating point* yang dikirimkan tergantung dari jenis intruksi yang diberikan dari pengguna yang diikuti dengan *output text* pada *output panel* yang menunjukkan kondisi pergerakan pada *quadcopter* yang sedang berlangsung. Ketika pengguna melakukan pergerakan yang tidak sesuai dengan aturan *finite state machine* yang telah dirancang sebelumnya, maka *quadcopter* akan masuk kedalam mode *hover* dan menampilkan *ouput text* pada *ouput panel* berupa peringatan bahwa pengguna telah melakukan gerakan yang tidak diizinkan pada sistem.



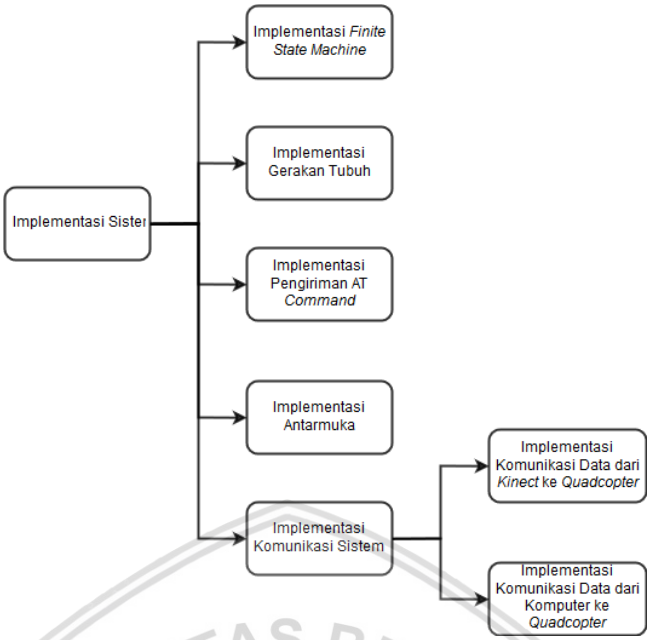
Gambar 5.21 *Flowchart* sistem kendali navigasi *quadcopter*

Tabel 5.3 Hubungan antar proses pada sistem

Kode status	Gerakan tubuh	Gerakan *quadcopter*	Pengiriman data
H1	Merentangkan kedua tangan	Hover	AT*PCMD =0,1,0,0,0,0<LF>
H2	Gerakan tangan kanan lebih besar daripada tangan kiri	*Roll* ke kanan	AT*PCMD =1,1,0.3,0,0,0<LF>
H3	Gerakan tangan kiri lebih besar daripada tangan kanan	*Roll* ke kiri	AT*PCMD =1,1,0.3,0,0,0<LF>
H4	Gerakan tangan kanan dan kiri diatas kepala	*Gaz* ke atas	AT*PCMD =1,1,0,0,0.3,0<LF>
H5	Gerakan tangan kanan dan kiri dibawah kepala	*Gaz* ke bawah	AT*PCMD =0,1,0,0,0.3,0<LF>
H6	Gerakan tangan kanan dan kiri didepan bahu	*Pitch* ke depan	AT*PCMD =1,1,0,0.3,0,0<LF>
H7	Gerakan tangan kanan dan kiri dibelakang bahu	*Pitch* ke belakang	AT*PCMD =0,1,0,0.3,0,0<LF>
H8	Gerakan tangan kanan didepan dan tangan kiri dibelakang	*Yaw* ke kanan	AT*PCMD =1,1,0,0,0,0.3<LF>
H9	Gerakan tangan kiri dedepan dan tangan kanan dibelakang	*Yaw* ke kiri	AT*PCMD =1,1,0,0,0,0.3<LF>
H10	Gerakan *lasso* tangan kanan	*Takeoff*	AT*REF =1,290718208<LF>
H11	Gerakan *lasso* tangan kiri	*Landing*	AT*REF =1,290717696<LF>
H12	Gerakan yang tidak dirancang pada sistem	*Hover*	AT*PCMD =0,1,0,0,0,0<LF>

5.2 Implementasi Sistem

Pada bagian ini dijelaskan mengenai tahap implementasi sistem berdasarkan dari perancangan sistem sebelumnya. Proses implementasi dilakukan secara berkelompok sesuai dengan perancangan sistem sebelumnya. Hal ini bertujuan untuk memudahkan proses implementasi sistem. Skema pengelompokan implementasi sistem dapat dilihat pada gambar 5.22.



Gambar 5.22 Skema implementasi sistem

Berdasarkan pada gambar 5.22 , implementasi sistem dibagi menjadi lima bagian yaitu, implementasi *finite state machine*, implementasi gerakan tubuh, implementasi AT *command*, implementasi antarmuka pengguna dan implementasi komunikasi sistem. Pada implementasi komuniasi sistem dibagi menjadi dua bagian yaitu implementasi komunikasi data dari *kinect* ke komputer dan implementasi komunikasi data dari komputer ke *quadcopter*. Penjelasan secara detail mengenai tahap implementasi sistem yang ditujukkan pada gambar 5.2 adalah sebagai berikut:

5.2.1 Implementasi *Finite State Machine*

Pada bagian ini dijelaskan mengenai implementasi metode *finite state machine* yang telah dirancang ke dalam sistem. Implementasi metode finite *state machine* dilalukan melalui tiga tahap, yaitu pada tahap pertama melakukan pendeklarasian semua state dan transisi yang telah dirancang pada perancangan *state machine* sebelumnya, kemudian tahap kedua melakukan pendeklarasian sebuah fungsi yang berguna untuk mengontrol setiap transisi-transisi yang terjadi pada setiap state yang telah dideklarasikan sebelumnya, kemudian langkah yang terakhir menerapkan state dan fungsi transisi tersebut kedalam pergerakan tubuh yang dilakukan oleh pengguna. Ketiga tahap tersebuat dijelaskan secara detail sebagai berikut:

1. Pendeklarasian *state* dan transisi.

   Source code pada tahap pendeklarasian state dan transisi dapat dilihat sebagai berikut:

```
1  var fsm = new StateMachine({
2  init: 'hover',
3  transitions: [
4  { name: 'right',     from: [ 'kanan', 'hover' ],  to: 'kanan' },
5  { name: 'left',   from: [ 'kiri', 'hover' ], to: 'kiri'  },
6  { name: 'bright', from: [ 'bkanan', 'hover' ], to: 'bkanan'    },
```

```
7  { name: 'bleft', from: [ 'bkiri', 'hover' ], to: 'bkiri'    },
8  { name: 'front', from: [ 'maju', 'hover' ], to: 'maju'    },
9  { name: 'up', from: [ 'atas', 'hover' ], to: 'atas'    },
10 { name: 'down', from: [ 'bawah', 'hover' ], to: 'bawah'    },
11 { name: 'back', from: [ 'mundur', 'hover' ],    to: 'mundur' },
12 { name: 'reset', from: [ '*' ], to: 'hover' }
13 ],
```

Program diawali dengan menginisialisasi sebuah variabel dengan nama StateMachine yang dideklarasikan pada baris petama. Lalu pada *syntax* baris kedua dideklarasikan sebuah variabel yang berguna sebagai *inital state* pada sistem *finite state machine* yang dibangun yaitu hover. Kemudian pada *syntax* baris ketiga hingga kesebelas dideklarasikan semua *states* yang akan digunakan pada sistem dan mendeklarasikan semua transisi sesuai dengan rancangan yang telah dibangun pada bab perancangan *finite state machine* sebelumnya. Pada *syntax* baris ke-4 dideklarasikan sebuah variabel transisi *state* dengan nama 'right' yang berguna untuk melakukan transisi dari gerakan hover ke gerakan *roll* kanan pada *quadcopter*. Pada *syntax* baris ke-5 dideklarasikan sebuah variabel transisi dengan nama 'left' yang berguna untuk melakukan transisi dari gerakan hover ke gerakan roll kiri pada *quadcopter*. Pada *syntax* baris ke-6 dideklarasikan sebuah variabel transisi dengan nama 'bright' yang berguna untuk melakukan transisi dari gerakan hover ke gerakan *yaw* kanan pada *quadcopter*. Pada *syntax* baris ke-7 dideklarasikan sebuah variabel transisi *state* dengan nama 'bleft' yang berguna untuk melakukan transisi dari gerakan hover ke gerakan *yaw* kiri pada *quadcopter*. Pada *syntax* baris ke-8 dideklarasikan sebuah variabel transisi dengan nama 'front' yang berguna untuk melakukan transisi dari gerakan hover ke gerakan *pitch* depan pada *quadcopter*. Pada *syntax* baris ke-9 dideklarasikan sebuah variabel transisi dengan nama 'up' yang berguna untuk melakukan transisi dari gerakan hover ke gerakan *gaz* atas pada *quadcopter*. Pada *syntax* baris ke-10 dideklarasikan sebuah variabel transisi dengan nama 'down' yang berguna untuk melakukan transisi dari gerakan hover ke gerakan *gaz* bawah pada *quadcopter*. Pada *syntax* baris ke-11 dideklarasikan sebuah variabel transisi dengan nama 'back' yang berguna untuk melakukan transisi dari gerakan hover ke gerakan *pitch* belakang pada *quadcopter*. Pada *syntax* baris ke 12 dideklarasikan sebuah variabel dengan nama 'reset' yang berguna untuk melakukan reset *state* dari semua pergerakan yaitu *pitch* depan dan belakang, *roll* kanan dan kiri, *gaz* atas dan bawah, dan *yaw* kanan dan kiri menjadi *state* hover pada *quadcopter*.

2. Pendeklarasian fungsi untuk mengontrol antar transisi *state*.

Source code pada tahap Pendeklarasian fungsi untuk mengontrol antar transisi state dapat dilihat sebagai berikut:

```
1  methods: {
2  onInvalidTransition: function(transition, from, to) {
3     console.log("transition not allowed from that state");
4     console.log("Quadcopter Akan Mengeksekusi State Hover dan
5                  Menampilkan State Terakhir");
6        client.stop();
```

```
7             },
8
9   onEnterHover: function() {
10     client.stop();
11     console.log('Kondisi Quadcopter : Berhenti');},
12  onEnterMaju: function() {
13     client.front();
14     console.log('Kondisi Quadcopter : Maju')},
15
16  onEnterMundur: function() {
17     client.back();
18     console.log('Kondisi Quadcopter : Mundur')},
19
20  onEnterAtas: function() {
21     client.up();
22     console.log('Kondisi Quadcopter : Ke Atas');
23
24  onEnterBawah: function() {
25     client.down();
26     console.log('Kondisi Quadcopter : Ke Bawah')},
27
28  onEnterKanan: function() {
29     console.log('Kondisi Quadcopter : Ke Kanan');
30     client.right();},
31
32  onEnterKiri: function() {
33     console.log('Kondisi Quadcopter : Ke Kiri')
34     client.left();},
35
36  onEnterBkanan: function() {
37     console.log('Kondisi Quadcopter : Putar Kanan')
38     client.clockwise();},
39
40  onEnterBkiri: function() {
41     console.log('Kondisi Quadcopter : Putar Kiri')
42     client.counterClockwise();},
43  }
```

Syntax pada baris pertama hingga ketujuh bertujuan untuk memberikan instruksi hover kepada *quadcopter* ketika pengguna melakukan transisi pergerakan tubuh yang tidak sesuai dengan transisi state yang telah dirancang sebelumnya. *Syntax* pada baris kesembilan hingga keempat puluh lima merupakan sebuah fungsi yang bertujuan untuk memberikan instruksi pergerakan pada *quadcopter* sesuai dengan transisi state yang dilakukan oleh pengguna. *Syntax* pada baris kesembilan hingga kesebelas merupakan sebuah fungsi yang berguna untuk memberikan instruksi pergerakan *hover* pada *quadcopter* ketika pengguna melakukan pergerakan tubuh yang menyebabkan sistem memasuki state *hover* dan akan menampilkan *text* berupa kondisi *quadcopter* sedang dalam keadaan berhenti pada *output panel*. *Syntax* pada baris kedua belas hingga keempat belas merupakan sebuah fungsi yang berguna untuk memberikan instruksi pergerakan *pitch* ke depan pada *quadcopter* ketika pengguna melakukan pergerakan tubuh yang menyebabkan sistem memasuki state *front* dan akan menampilkan *text* berupa kondisi *quadcopter* sedang dalam keadaan ke depan pada *output panel*. *Syntax* pada baris keenam belas hingga kedelapan belas merupakan sebuah fungsi yang berguna untuk memberikan

instruksi pergerakan *pitch* ke belakang pada *quadcopter* ketika pengguna melakukan pergerakan tubuh yang menyebabkan sistem memasuki state *back* dan akan menampilkan *text* berupa kondisi *quadcopter* sedang dalam keadaan ke belakang pada *output panel*. *Syntax* pada baris kedua puluh hingga kedua puluh dua merupakan sebuah fungsi yang berguna untuk memberikan instruksi pergerakan *gaz* ke atas pada *quadcopter* ketika pengguna melakukan pergerakan tubuh yang menyebabkan sistem memasuki state *up* dan akan menampilkan *text* berupa kondisi *quadcopter* sedang dalam keadaan ke atas pada *output panel*. *Syntax* pada baris kedua puluh empat hingga kedua puluh enam merupakan sebuah fungsi yang berguna untuk memberikan instruksi pergerakan *gaz* ke bawah pada *quadcopter* ketika pengguna melakukan pergerakan tubuh yang menyebabkan sistem memasuki state *down* dan akan menampilkan *text* berupa kondisi *quadcopter* sedang dalam keadaan ke bawah pada *output* panel. *Syntax* pada baris kedua puluh delapan hingga ketiga puluh merupakan sebuah fungsi yang berguna untuk memberikan instruksi pergerakan *roll* ke kanan pada *quadcopter* ketika pengguna melakukan pergerakan tubuh yang menyebabkan sistem memasuki state *right* dan akan menampilkan *text* berupa kondisi *quadcopter* sedang dalam keadaan ke kanan pada *output* panel. *Syntax* pada baris ketiga puluh dua hingga ketiga puluh empat merupakan sebuah fungsi yang berguna untuk memberikan instruksi pergerakan *roll* ke kiri pada *quadcopter* ketika pengguna melakukan pergerakan tubuh yang menyebabkan sistem memasuki state *left* dan akan menampilkan *text* berupa kondisi *quadcopter* sedang dalam keadaan ke kiri pada *output* panel. *Syntax* pada baris ketiga puluh enam hingga ketiga puluh delapan merupakan sebuah fungsi yang berguna untuk memberikan instruksi pergerakan *yaw* ke kanan pada *quadcopter* ketika pengguna melakukan pergerakan tubuh yang menyebabkan sistem memasuki state *bright* dan akan menampilkan *text* berupa kondisi *quadcopter* sedang dalam keadaan berputar ke kanan pada *output* panel. Dan s*yntax* pada baris keempat puluh hingga keempat puluh dua merupakan sebuah fungsi yang berguna untuk memberikan instruksi pergerakan *yaw* ke kiri pada *quadcopter* ketika pengguna melakukan pergerakan tubuh yang menyebabkan sistem memasuki state *bleft* dan akan menampilkan *text* berupa kondisi *quadcopter* sedang dalam keadaan berputar ke kiri pada *output panel.*

3. Penerapan state dan fungsi transisi ke dalam pergerakan tubuh

Setelah dilakukan Pendeklarasian state dan transisi dan fungsi untuk mengontrol antar transisi state, maka metode *finite state machine* telah siap diterapkan ke dalam pergerakan tubuh yang akan dilakukan oleh pengguna. Source Code penerapan metode *finite state machine* ke dalam pergerakan tubuh pengguna dalam dilihat sebagai berikut:

```
1  var if (hover) {
2  fsm.reset();
3  console.log(fsm.state);}
4
5  else if (gerakKanan) {
6  fsm.right();
7  console.log(fsm.state);}
```

```
8
9   else if (gerakKiri) {
10  fsm.left();
11  console.log(fsm.state);}
12
13  else if (gazAt) {
14  fsm.up();
15  console.log(fsm.state);}
16
17  else if (gazBa) {
18  fsm.down();
19  console.log(fsm.state);}
20
21  else if (yawKa) {
22  fsm.bright();
23  console.log(fsm.state);}
24
25  else if (yawKi) {
26  fsm.bleft();
27  console.log(fsm.state);}
28
29  else if (pitchDpn) {
30  fsm.front();
31  console.log(fsm.state);}
32
33  else if (pitchBlk) {
34  fsm.back();
35  console.log(fsm.state);}
36
37  else if (lKanan) {
38  client.takeoff();
39  console.log("Kondisi Quadcopter : Take Off");}
40
41  else if (lKiri) {
42  client.land();
43  console.log("Kondisi Quadcopter : Landing")
44  setTimeout(function(){
45        process.exit();
46        console.log("Exit Program Success");}, 5000);
```

Syntax pada baris pertama hingga keempat puluh enam bertujuan untuk mengaktifkan metode *finite state machine* yang telah dirancang ke dalam pergerakan tubuh pengguna. *Syntax* pada baris pertama hingga ketiga berguna untuk mengaktifkan fungsi transisi *reset* dan akan memberikan instruksi hover pada *quadcopte*. *Syntax* pada baris kelima hingga ketujuh berguna untuk mengaktifkan fungsi transisi *right* dan akan memberikan instruksi *roll* ke kanan pada *quadcopter*. *Syntax* pada baris kesembilan hingga kesebelas berguna untuk mengaktifkan fungsi transisi *left* dan akan memberikan instruksi *roll* ke kiri pada *quadcopter*. *Syntax* pada baris ketiga belas hingga kelima belas berguna untuk mengaktifkan fungsi transisi *up* dan akan memberikan instruksi *gaz* ke atas pada *quadcopter*. *Syntax* pada baris ketujuh belas hingga kesembilan belas berguna untuk mengaktifkan fungsi transisi *down* dan akan memberikan instruksi *gaz* ke bawah pada *quadcopter*. *Syntax* pada baris kedua puluh satu hingga kedua puluh tiga berguna untuk mengaktifkan fungsi transisi *bright* dan akan memberikan instruksi *yaw* ke kanan pada *quadcopter*. *Syntax* pada baris kedua puluh lima hingga kedua puluh tujuh berguna untuk mengaktifkan fungsi transisi *bleft* dan

akan memberikan instruksi *yaw* ke kiri pada *quadcopter*. *Syntax* pada baris kedua puluh sembilan hingga ketiga puluh satu berguna untuk mengaktifkan fungsi transisi *front* dan akan memberikan instruksi pitch ke depan pada *quadcopter*. *Syntax* pada baris ketiga puluh tiga hingga ketiga puluh lima berguna untuk mengaktifkan fungsi transisi *back* dan akan memberikan instruksi pitch ke belakang pada *quadcopter*. *Syntax* pada baris ketiga puluh tujuh hingga ketiga puluh sembilan berguna untuk memberikan instruksi *takeoff* pada *quadcopter*. *Syntax* pada baris keempat puluh satu hingga keempat puluh dua berguna untuk memberikan instruksi *landing* pada *quadcopter*. Sebagai contoh jika pengguna melakukan gerakan ke kekiri, maka *syntax* pada baris kesembilan hingga kesebelas akan dijalankan kemudian sistem akan melihat terlebih dahulu state terakhir dari *quadcopter*. Jika state terakhirnya adalah hover dan kiri, maka sistem memperbolehkan *quadcopter* untuk menerima instruksi gerakan kekiri dari pengguna. Namun jika state terkahir dari *quadcopter* bukan *hover* dan kiri, maka sistem akan menolak inputan dari pengguna dan *quadcopter* menjalankan instruksi gerakan *hover* secara otomatis. *Syntax* pada baris keempat puluh satu hingga keempat puluh enam bertujuan untuk mengakhiri program dengan *delay* waktu selama 5000 ms ketika pengguna melakukan pergerakan landing pada *quadcopter*.

5.2.2 Implementasi Gerakan Tubuh

Implementasi gerakan tubuh dimulai dengan melakukan inisialisasi terhadap tiap -tiap joint pada tubuh pengguna. Setiap Setiap Joint diinisialisasi dengan menggunakan koordinat X, Y, dan Z. Joint dari butuh pengguna yang digunakan pada penelitian ini yaitu tangan, bahu, siku, leher, dan telunjuk tangan. Data joint yang digunakan diambil dari class kinect2.js. Pengambilan data pada class tersebut diawali dengan mengaktifkan fungsi openBodyReader() pada class kinect2.js. *Syntax* pada baris pertama berguna untuk mengaktifkan fungsi bodyFrame yang berguna untuk membaca data pergerakan pada pengguna ketika pengguna tersebut berada didepan sensor kamera *kinect*. Pada *syntax* baris ketiga hingga keenam dideklarasikan sebuah variabel dengan nama kepala yang berguna untuk menyimpan data pergerakan *joint* pada kepala yang dimana nilai jointnya ditinjau dari koordinat Y dan Z. Pada *syntax* baris ketujuh hingga kesepuluh dideklarasikan sebuah variabel dengan nama dada yang berguna untuk menyimpan data pergerakan *joint* pada dada yang dimana nilai jointnya ditinjau dari koordinat Y dan Z. Pada *syntax* baris kesebelas hingga keempat belas dideklarasikan sebuah variabel dengan nama pinggang yang berguna untuk menyimpan data pergerakan *joint* pada pinggang yang dimana nilai jointnya ditinjau dari koordinat Y dan Z. Pada *syntax* baris ketujuh belas hingga kedua puluh dua dideklarasikan sebuah variabel dengan nama tangan_kanan yang berguna untuk menyimpan data pergerakan *joint* pada tangan kanan yang dimana nilai jointnya ditinjau dari koordinat X, Y, dan Z. Pada *syntax* baris kedua puluh tiga hingga kedua puluh delapan dideklarasikan sebuah variabel dengan nama tangan_kiri yang berguna untuk menyimpan data pergerakan *joint* pada tangan kiri yang dimana nilai jointnya ditinjau dari koordinat X, Y, dan Z. Pada *syntax* baris ketiga puluh satu hingga ketiga puluh sembilan dideklarasikan sebuah variabel dengan nama bahu_tengah

yang berguna untuk menyimpan data pergerakan *joint* pada bahu tengah yang dimana nilai jointnya ditinjau dari koordinat X, Y, dan Z. Pada *syntax* baris keempat puluh hingga keempat puluh lima dideklarasikan sebuah variabel dengan nama bahu_kiri yang berguna untuk menyimpan data pergerakan *joint* pada bahu kiri yang dimana nilai jointnya ditinjau dari koordinat X, Y, dan Z. Pada *syntax* baris keempat puluh hingga keempat puluh lima dideklarasikan sebuah variabel dengan nama bahu_kiri yang berguna untuk menyimpan data pergerakan *joint* pada bahu kiri yang dimana nilai jointnya ditinjau dari koordinat X, Y, dan Z. Pada *syntax* baris keempat puluh tujuh hingga kelima puluh lima dideklarasikan sebuah variabel dengan nama bahu_kanan yang berguna untuk menyimpan data pergerakan *joint* pada bahu kanan yang dimana nilai jointnya ditinjau dari koordinat X, Y, dan Z. Pada *syntax* baris lima puluh enam hingga kelima puluh tujuh dideklarasikan sebuah variabel dengan nama leher yang berguna untuk menyimpan data pergerakan *joint* pada leher yang dimana nilai jointnya ditinjau dari koordinat Z. Pada *syntax* baris lima puluh delapan hingga ketujuh puluh dideklarasikan sebuah variabel dengan nama tangan yang berguna untuk menyimpan data pergerakan *joint* pada tangan yang dimana nilai jointnya terdiri dari beberapa *state*, yaitu tangan terbuka, tangan tertutup, mengacungkan satu jari, dan posisi tangan yang tidak terdeteksi oleh *sensor kinct*. Keseluran *syntax* yang digunakan untuk menginisialisasi joint dapat dilihat sebagai berikut:

```
1  kinect.on('bodyFrame', function(bodyFrame){
2
3  var kepala_Y =
4  bodyFrame.bodies[i].joints[Kinect2.JointType.head].cameraY;
5  var kepala_Z =
6  bodyFrame.bodies[i].joints[Kinect2.JointType.head].cameraZ;
7  var dada_Y =
8  bodyFrame.bodies[i].joints[Kinect2.JointType.spineMid].cameraY;
9  var dada_Z =
10 bodyFrame.bodies[i].joints[Kinect2.JointType.spineMid].cameraZ;
11 var pinggang_Y =
12 bodyFrame.bodies[i].joints[Kinect2.JointType.spineBase].cameraY;
13 var pinggang_Z =
14 bodyFrame.bodies[i].joints[Kinect2.JointType.spineBase].cameraZ;
15
16 //Variabel Untuk Tangan
17 var tangan_kananX =
18 bodyFrame.bodies[i].joints[Kinect2.JointType.handRight].cameraX;
19 global.tangan_kananY =
20 bodyFrame.bodies[i].joints[Kinect2.JointType.handRight].cameraY;
21 var tangan_kananZ =
22 bodyFrame.bodies[i].joints[Kinect2.JointType.handRight].cameraZ;
23 var tangan_kiriX =
24 bodyFrame.bodies[i].joints[Kinect2.JointType.handLeft].cameraX;
25 global.tangan_kiriY =
26 bodyFrame.bodies[i].joints[Kinect2.JointType.handLeft].cameraY;
27 var tangan_kiriZ =
28 bodyFrame.bodies[i].joints[Kinect2.JointType.handLeft].cameraZ;
29
30 //Variabel Untuk Bahu
31 var bahu_tengahX =
32 bodyFrame.bodies[i].joints[Kinect2.JointType.spineShoulder].cameraX
33 ;
34
35
```

```
36  var bahu_tengahY =
37  bodyFrame.bodies[i].joints[Kinect2.JointType.spineShoulder].cameraY
38  ;
39  var bahu_tengahZ =
40  bodyFrame.bodies[i].joints[Kinect2.JointType.spineShoulder].cameraZ
41  ;
42  var bahu_kiriX =
43  bodyFrame.bodies[i].joints[Kinect2.JointType.shoulderLeft].cameraX;
44  var bahu_kiriY =
45  bodyFrame.bodies[i].joints[Kinect2.JointType.shoulderLeft].cameraY;
46  var bahu_kiriZ =
47  bodyFrame.bodies[i].joints[Kinect2.JointType.shoulderLeft].cameraZ;
48  var bahu_kananX =
49  bodyFrame.bodies[i].joints[Kinect2.JointType.shoulderRight].cameraY
50  ;
51  var bahu_kananY =
52  bodyFrame.bodies[i].joints[Kinect2.JointType.shoulderRight].cameraY
53  ;
54  var bahu_kananZ =
55  bodyFrame.bodies[i].joints[Kinect2.JointType.shoulderRight].cameraZ
56  ;
57  //Variabel Untuk leher
58  var leher =
59  bodyFrame.bodies[i].joints[Kinect2.JointType.neck].cameraZ;
60  //Variabel Mix tangan
61  var closedState = bodyFrame.bodies[i].leftHandState==3 &&
62  bodyFrame.bodies[i].rightHandState==3;
63  var openState = bodyFrame.bodies[i].leftHandState==2 &&
64  bodyFrame.bodies[i].rightHandState==2;
65  var openStateKanan = bodyFrame.bodies[i].rightHandState==2;
66  var openStateKiri = bodyFrame.bodies[i].leftHandState==2;
67  var lassoState = bodyFrame.bodies[i].rightHandState==4 &&
68  bodyFrame.bodies[i].leftHandState==4;
69  var lassoKanan = bodyFrame.bodies[i].rightHandState==4;
70  var lassoKiri = bodyFrame.bodies[i].leftHandState==4;
    var notTracked = bodyFrame.bodies[i].leftHandState==0 &&
    bodyFrame.bodies[i].rightHandState==0;
```

Setelah inisialisasi variabel untuk tiap-tiap joint tubuh dilakukan, maka dideklarasikan sebuah variabel baru yang berguna agar pengguna dapat mengendalikan *quadcopter* sesuai dengan kombinasi gerakan tubuh yang telah dijelaskan pada bab perancangan gerakan tubuh. Adapun *syntax* untuk setiap gerakan tubuh diperlihatkan sebagai berikut:

a. Gerakan merentangkan kedua tangan.

Kondisi pada pergerakan ini disimpan dalam variabel hover yang bertujuan untuk melakukan instruksi hover pada *quadcopter* ketika pengguna melakukan pergerakan tubuh dengan kondisi joint dibawah ini dihadapan sensor kamera kinect. Pada *syntax* baris ketiga, nilai selisih antara nilai joint pada tangan kanan yang diukur dari koordinat Z dan nilai joint pada leher yang diukur dari korrdinat Z berada diantara -0.1 hingga 0.1. Pada syntax baris *keempat*, hasil selisih antara nilai pada joint tangan kanan yang diukur dari koordinat Z dengan nilai joint leher yang diukur dari koordinat Z berada diantara -0.1 hingga 0.1.

```
1  var hover =  kepala > tangan_kananY && tangan_kananY > dada &&
2               kepala > tangan_kiriY && tangan_kiriY > dada &&
3               -0.1 < tangan_kananZ - leher < 0.1 &&
4               -0.1 < tangan_kiriZ - leher < 0.1
```

b. Gerakan tangan kiri lebih tinggi daripada tangan kanan.

Kondisi pada pergerakan ini disimpan dalam variabel gerakKiri yang bertujuan untuk melakukan instruksi *roll* kekiri pada *quadcopter* ketika pengguna melakukan pergerakan tubuh dengan kondisi joint dibawah ini dihadapan sensor kamera kinect. Pada *syntax* baris keempat hingga kelima, nilai selisih antara nilai joint pada tangan kanan yang diukur dari koordinat Z dengan nilai joint pada leher yang diukur dari koordinat Z berada diantara -0.1 hingga 0.1. Pada *syntax* baris kelima hingga keenam, hasil selisih antara nilai *joint* pada tangan kiri yang diukur dari koordinat Z dengan nilai *joint* pada leher yang diukur dari koordinat Z berada diantara -0.1 hingga 0.1.

```
1  var gerakKiri =  tangan_kiriY > siku_kiriY && siku_kiriY >
2                   bahu_kiriY && bahu_kiriY > bahu_kananY &&
3                   bahu_kananY > siku_kananY && siku_kananY >
4                   tangan_kananY && tangan_kiriY > kepala &&  -0.1 <
5                   tangan_kananZ - leher < 0.1 && -0.1 < tangan_kiriZ
6                   - leher < 0.1;
```

c. Gerakan tangan kanan lebih tinggi daripada tangan kiri.

Kondisi pada pergerakan ini disimpan dalam variabel gerakKanan yang bertujuan untuk melakukan instruksi roll ke kanan pada *quadcopter* ketika pengguna melakukan pergerakan tubuh dengan kondisi joint dibawah ini dihadapan sensor kamera *kinect*. Pada *syntax* baris keempat hingga kelima, nilai selisih antara nilai joint pada tangan kanan yang diukur dari koordinat Z dengan nilai joint pada leher yang diukur dari koordinat Z berada diantara -0.1 hingga 0.1. Pada *syntax* baris kelima hingga keenam, hasil selisih antara nilai *joint* pada tangan kiri yang diukur dari koordinat Z dengan nilai *joint* pada leher yang diukur dari koordinat Z berada diantara -0.1 hingga 0.1.

```
1  var gerakKanan =  tangan_kiriY < siku_kiriY && siku_kiriY <
2                    bahu_kiriY && bahu_kiriY < bahu_kananY &&
3                    bahu_kananY < siku_kananY && siku_kananY <
4                    tangan_kananY && tangan_kananY > kepala && -0.1
5                    < tangan_kananZ - leher < 0.1 && -0.1 <
6                    tangan_kiriZ - leher < 0.1;
```

d. Gerakan tangan kanan dan kiri diatas kepala.

Kondisi pada pergerakan ini disimpan dalam variabel gazAt yang bertujuan untuk melakukan instruksi gaz ke atas pada *quadcopter* ketika pengguna melakukan pergerakan tubuh dengan kondisi joint dibawah ini dihadapan sensor kamera kinect. Pada *syntax* baris kedua, nilai *joint* pada tangan kanan yang diukur dari koordinat Y berada diantara 0 dan 0.4. Pada *syntax* baris ketiga, nilai *joint* pada tangan kiri yang diukur dari koordinat Y berada diantara 0 dan 0.4. Pada *syntax* baris ketiga hingga keempat, nilai selisih antara nilai joint pada tangan kanan yang diukur dari koordinat Z dengan nilai joint pada leher yang diukur dari koordinat Z berada diantara -0.1 hingga 0.1. Pada *syntax* baris keempat hingga kelima, hasil selisih antara nilai *joint* pada tangan kiri yang diukur dari koordinat Z dengan nilai *joint* pada leher yang diukur dari koordinat Z berada diantara -0.1 hingga 0.1.

```
var gazAt = tangan_kananY > kepala && tangan_kiriY > kepala &&
            0 < tangan_kananY  && 0.4 > tangan_kananY && 0 <
            tangan_kiriY && 0.4 > tangan_kiriY && -0.1 <
            tangan_kananZ - leher < 0.1 && -0.1 < tangan_kiriZ -
            leher < 0.1
            ;
```

e. Gerakan tangan kanan dan kiri dibawah kepala.

Kondisi pada pergerakan ini disimpan dalam variabel gazBa yang bertujuan untuk melakukan instruksi gaz ke bawah pada *quadcopter* ketika pengguna melakukan pergerakan tubuh dengan kondisi joint dibawah ini dihadapan sensor kamera kinect. Pada *syntax* baris keempat hingga kelima, hasil selisih antara nilai *joint* pada tangan kanan yang diukur dari koordinat Z dengan nilai *joint* pada pinggang yang diukur dari koordinat Z lebih besar daripada -0.2. Pada *syntax* baris keempat hingga kelima, hasil selisih antara nilai *joint* pada tangan kiri yang diukur dari koordinat Z dengan nilai *joint* pada pinggang yang diukur dari koordinat Z lebih besar dari pada -0.2.

```
var gazBa = tangan_kananY < dada && tangan_kiriY < dada &&
            tangan_kananY > pinggang && tangan_kiriY > pinggang &&
            tangan_kananZ < dada_Z && tangan_kiriZ < dada_Z &&
            tangan_kananZ - pinggang_Z < -0.2 && tangan_kiriZ -
            pinggang_Z < -0.2 ;
```

f. Gerakan tangan kanan dan kiri didepan bahu.

Kondisi pada pergerakan ini disimpan dalam variabel pitchDpn yang bertujuan untuk melakukan instruksi pitch ke depan pada *quadcopter* ketika pengguna melakukan pergerakan tubuh dengan kondisi joint dibawah ini dihadapan sensor kamera kinect. Pada *syntax* baris pertama hasil selisih antara nilai *joint* pada tangan kanan yang diukur dari koordinat X dengan nilai *joint* pada tangan kiri yang diukur dari koordinat X lebih besar daripada 3.

```
var pitchDpn =  tangan_kananX - tangan_kiriX < 0.3 && tangan_kananY
                < kepala && tangan_kananY > pinggang  ;
```

g. Gerakan tangan kanan dan kiri dibelakang pinggang.

Kondisi pada pergerakan ini disimpan dalam variabel pitchBlk yang bertujuan untuk melakukan instruksi pitch ke belakang pada *quadcopter* ketika pengguna melakukan pergerakan tubuh dengan kondisi joint dibawah ini dihadapan sensor kamera kinect.

```
var pitchBlk =  tangan_kananY < dada && tangan_kiriY < dada &&
                tangan_kananY > pinggang && tangan_kiriY > pinggang
                && tangan_kananZ > dada_Z && tangan_kiriZ > dada_Z;
```

h. Gerakan tangan kiri didepan dan tangan kanan dibelakang.

Kondisi pada pergerakan ini disimpan dalam variabel yawKi yang bertujuan untuk melakukan instruksi yaw ke kiri pada *quadcopter* ketika pengguna melakukan pergerakan tubuh dengan kondisi joint dibawah ini dihadapan sensor kamera kinect.

```
var yawKi = kepala > tangan_kananY && tangan_kananY > dada &&
            kepala > tangan_kiriY && tangan_kiriY > dada &&
            tangan_kiriZ < kepala_Z && tangan_kananZ > dada_Z ;
```

i. Gerakan tangan kanan didepan dan tangan kiri dibelakang.

Kondisi pada pergerakan ini disimpan dalam variabel yawKa yang bertujuan untuk melakukan instruksi *yaw* ke kanan pada *quadcopter* ketika pengguna melakukan pergerakan tubuh dengan kondisi joint dibawah ini dihadapan sensor kamera kinect.

```
var yawKa = kepala > tangan_kananY && tangan_kananY > dada && kepala
            > tangan_kiriY && tangan_kiriY > dada && tangan_kananZ <
            kepala_Z && tangan_kiriZ > dada_Z ;
```

j. Gerakan lasso pada tangan kanan.

Kondisi pada pergerakan ini disimpan dalam variabel lKanan yang bertujuan untuk melakukan instruksi takeoff pada *quadcopter* ketika pengguna melakukan pergerakan tubuh dengan kondisi joint dibawah ini dihadapan sensor kamera kinect.

```
var lKanan = lassoKanan && tangan_kananY < kepala  ;
```

k. Gerakan lasso pada tangan kiri.

Kondisi pada pergerakan ini disimpan dalam variabel lKiri yang bertujuan untuk melakukan instruksi landing pada *quadcopter* ketika pengguna melakukan pergerakan tubuh dengan kondisi joint dibawah ini dihadapan sensor kamera kinect.

```
var lKiri = lassoKiri && tangan_kiriY < kepala  ;
```

5.2.3 Implementasi Pengiriman AT Command

Pengendalian dan konfigurasi pada *quadcopter* dilakukan dengan mengirim AT Command melalui jaringan *wi-fi* dengan menggunakan protokol UDP pada port 5556. Jenis AT Command yang digunakan pada penelitian ini yaitu AT Command Reference (AT*REF) dan AT Command Per Call Measurement Data (AT*PCMD). Pada proses implementasi AT Comamand, terdapat tiga tahap penting yang harus dilakukan agar *quadcopter* dapat dikonfigurasi dan dikendalikan dengan optimal yaitu mengkonversi nilai kecepatan yang ada pada class client.js dari bentuk float menjadi hexadecimal, membentuk sebuah format pada AT*REF dan AT*PCMD, dan melakukan pengkondisian pada AT*REF dan AT*PCMD tersebut. Penjelasan pada masing-masing tahap tersebut dijelaskan sebagai berikut:

a. Mengkonversi nilai kecepatan dari bentuk float menjadi hexadecimal.

Untuk mengubah nilai kecepatan yang semula berbentuk float menjadi hexadecimal, maka dibuat sebuah class dengan nama at.js yang memiliki fungsi untuk mengubah nilai kecepatan yang didapat dari class client.js menjadi bentuk *hexadecimal* dengan nilai *base* sebesar 16.

```
1  at.floatString = function(number) {
2    var buffer = new Buffer(4);
3    buffer.writeFloatBE(number, 0);
4    return -~parseInt(buffer.toString('hex'), 16) - 1;
5  };
```

b. Menciptakan format pengiriman AT*PCMD dan AT*REF.

Untuk membentuk format pengiriman AT Command baik itu dalam bentuk AT* REF maupun dalam bentuk AT*PCMD, maka dibuat sebuah class dengan nama atCommand.js. Syntax this.type pada baris ketiga digunakan untuk menentukan jenis AT Command yang akan dipakai. Sedangkan syntax args.joint pada baris ketiga diguanakan untuk mengisi nilai sequence number dari masing-masing argument pada AT*REF dan AT*PCMD.

```
1  AtCommand.prototype.serialize = function(sequenceNumber) {
2    var args = [sequenceNumber].concat(this.args);
3    return 'AT*' + this.type + '=' + args.join(',') + '\r';
4  };
```

c. Melakukan pengkondisian argument pada masing-masing AT Command.

Untuk membentuk format pengiriman AT Command baik itu dalam bentuk AT* REF maupun dalam bentuk AT*PCMD, maka dibuat sebuah class dengan nama atCommand.js. Syntax *this.type* pada baris ketiga digunakan untuk menentukan jenis AT Command yang akan dipakai. Sedangkan syntax args.joint pada baris ketiga diguanakan untuk mengisi nilai sequence number dari masing-masing argument pada AT*REF dan AT*PCMD. Pada syntax baris kelima belas, nilai pada variabel emergency bernilai 1 << 8 yang jika ubah kedalam bentuk *binary* menjadi bernilai 100000000. Dan pada syntax baris keenam belas, nilai pada variabel takeoff bernilai 1 << 9 yang jika diubah kedalam bentuk *binary* menjadi bernilai 1000000000.

```
1  AtCommandCreator.prototype.ref = function(options) {
2    options = options || {};
3    var args = [0];
4    if (options.fly) {
5      args[0] = args[0] | REF_FLAGS.takeoff;
6      }
7
8      if (options.emergency) {
9        args[0] = args[0] | REF_FLAGS.emergency;
10     }
11     return this.raw('REF', args);
12 };
13
14 var REF_FLAGS = exports.REF_FLAGS = {
15 emergency : (1 << 8),
16 takeoff : (1 << 9),};
```

Fungsi pada script berikutnya berguna untuk membentuk suatu argument AT*PCMD. Syntax pada baris keempat berguna untuk mendeklarasikan array yang menyimpan nilai variabel pada flags, roll, pitch, gaz, dan yaw. Syntax pada baris kelima hingga kesebelas digunakan untuk melakukan pengkondisian yang didapat dari class client.js menjadi bernilai positif atau negatif sesuai dengan gerakan yang

dilakukan pada *quadcopter*. Sebagai contoh jika *quadcopter* melakukan gerakan ke kanan maka nilai variabel pada roll akan bernilai 1 dan jika *quadcopter* melakukan gerakan ke kiri maka variabel roll akan bernilai -1. Kemudian nilai argument pada variabel roll, pitch, gaz, dan yaw akan diubah ke dalam bentuk hexadecimal menggunakan fungsi konversi nilai pada class AT.js. nilai PCMD_FLAG diatur dengan nilai *bitwise* 1<<0 yang jika diubah ke nilai desimal menjadi bernilai 1. Nilai pada PCMD_FLAG diatur dengan nilai 1 agar sesuai dengan aturan *syntax* pada pengiriman AT*PCMD yang dimana jika ingin mengatifkan mode progesive command pada *quadcopter*, maka nilai pada PCMD FLAG harus diset dengan nilai 1. Aturan pada kondisi-kondisi gerakan yang dapat dilakukan oleh *quadcopter* terdapat pada syntax baris kedua puluh empat hingga ketiga puluh tiga.

```
1  AtCommandCreator.prototype.pcmd = function(options) {
2    options = options || {};
3    // flags, leftRight, frontBack, upDown, clockWise
4    var args = [0, 0, 0, 0, 0];
5
6    for (var key in options) {
7      var alias = PCMD_ALIASES[key];
8      var value = options[key];
9
10     if (alias.invert) {
11       value = -value;
12     }
13     args[alias.index] = at.floatString(value);
14     args[0] = args[0] | PCMD_FLAGS.progressive;
15     }
16
17     return this.raw('PCMD', args);
18 };
19
20 var PCMD_FLAGS = exports.PCMD_FLAGS = {
21   progressive : (1 << 0),
22 };
23
24 var PCMD_ALIASES = exports.PCMD_ALIASES = {
25   left : {index: 1, invert: true},
26   right : {index: 1, invert: false},
27   front : {index: 2, invert: true},
28   back : {index: 2, invert: false},
29   up : {index: 3, invert: false},
30   down : {index: 3, invert: true},
31   clockwise : {index: 4, invert: false},
32   counterClockwise : {index: 4, invert: true},
33 };
```

5.2.4 Implementasi Antarmuka Pengguna

Pada bagian ini dijelaskan mengenai implementasi antar muka pengguna pada sistem. Pada penelitian ini, hasil keluaran pada sistem ditampilkan dengan menggunakan dua jenis antar muka. Antarmuka yang pertama bertujuan untuk menampilkan kondisi terakhir pada saat *quadcopter* dikendalikan dan kondisi error yang disebabkan ketika pengguna melakukan perpindahan gerakan yang tidak seseuai dengan rancangan *finite state machine* yang telah dibangun pada sub bab

sebelumnya. Sedangkan antarmuka yang kedua digunakan untuk menampilkan data navigasi berupa sudut *roll*, *pitch*, *yaw*, dan ketinggian secara *realtime*. Kedua jenis antarmuka tersebut dijelaskan sebagai berikut:

1. Antarmuka untuk menampilkan kondisi terakhir *quadcopter* dan kondisi error.

   Sistem akan menampilkan kondisi terakhir pada saat *quadcopter* dikendalikan dan kondisi error yang disebabkan ketika pengguna melakukan perpindahan gerakan yang tidak seseuai dengan rancangan *finite state machine* yang telah dibangun pada sub bab sebelumnya berupa text pada output panel yang ada pada ATOM. Adapun semua kondisi yang dirancang dijelaskan sebagai berikut:

   a. Gerakan *Hover*.
      Source code dibawah ini digunakan untuk menampilkan kondisi hover output panel ATOM pada komputer ketika pengguna melakukan pergerakan hover.

```
1  if (hover) {
2      fsm.reset();
3      console.log(fsm.state);
4  }
```

   b. Gerakan ke kanan.
      Source code dibawah ini digunakan untuk menampilkan kondisi ke kanan pada output panel ATOM ketika pengguna melakukan pergerakan ke kanan dan akan mengeluarkan pesan peringatan ketika pengguna melakukan pergerakan error.

```
1  else if (gerakKanan) {
2  fsm.right();
3  console.log(fsm.state);
4  }
```

   c. Gerakan kekiri.
      Source code dibawah ini digunakan untuk menampilkan kondisi ke kiri pada output panel ATOM ketika pengguna melakukan pergerakan ke kiri dan akan mengeluarkan pesan peringatan ketika pengguna melakukan pergerakan error.

```
1  else if (gerakKiri) {
2  fsm.left();
3  console.log(fsm.state);
4  }
```

   d. Gerakan ke atas.
      Source code dibawah ini digunakan untuk menampilkan kondisi ke atas pada output panel ATOM ketika pengguna melakukan pergerakan keatas dan akan mengeluarkan pesan peringatan ketika pengguna melakukan pergerakan error.

```
1  else if (gazAt) {
2  fsm.up();
3  console.log(fsm.state);
```

```
4  }
```

e. Gerakan ke bawah.
Source code dibawah ini digunakan untuk menampilkan kondisi ke bawah pada output panel ATOM ketika pengguna melakukan pergerakan ke bawah dan akan mengeluarkan pesan peringatan ketika pengguna melakukan pergerakan error.

```
1  else if (gazBa) {
2  fsm.down();
3  console.log(fsm.state);
4  }
```

f. Gerakan ke depan.
Source code dibawah ini digunakan untuk menampilkan kondisi ke depan pada output panel ATOM ketika pengguna melakukan pergerakan kedepan dan akan mengeluarkan pesan peringatan ketika pengguna melakukan pergerakan error.

```
1  else if (pitchDpn) {
2  fsm.front();
3  console.log(fsm.state);
4  }
```

g. Gerakan kebelakang.
Source code dibawah ini digunakan untuk menampilkan kondisi kebelakang pada output panel ATOM ketika pengguna melakukan pergerakan kebelakang dan akan mengeluarkan pesan peringatan ketika pengguna melakukan pergerakan error.

```
1  else if (pitchBlk) {
2  fsm.back();
3  console.log(fsm.state);
4  }
```

h. Gerakan yaw ke kanan.
Source code dibawah ini digunakan untuk menampilkan yaw kanan pada output panel ATOM ketika pengguna melakukan pergerakan yaw kanan dan akan mengeluarkan pesan peringatan ketika pengguna melakukan pergerakan error.

```
1  else if (yawKa) {
2  fsm.bright();
3  console.log(fsm.state);
4  }
```

i. Gerakan yaw ke kiri.
Source code dibawah ini digunakan untuk menampilkan kondisi yaw ke kiri pada output panel ATOM ketika pengguna melakukan pergerakan yaw kiri dan akan mengeluarkan pesan peringatan ketika pengguna melakukan pergerakan error.

```
else if (yawKi) {
fsm.bleft();
console.log(fsm.state);
}
```

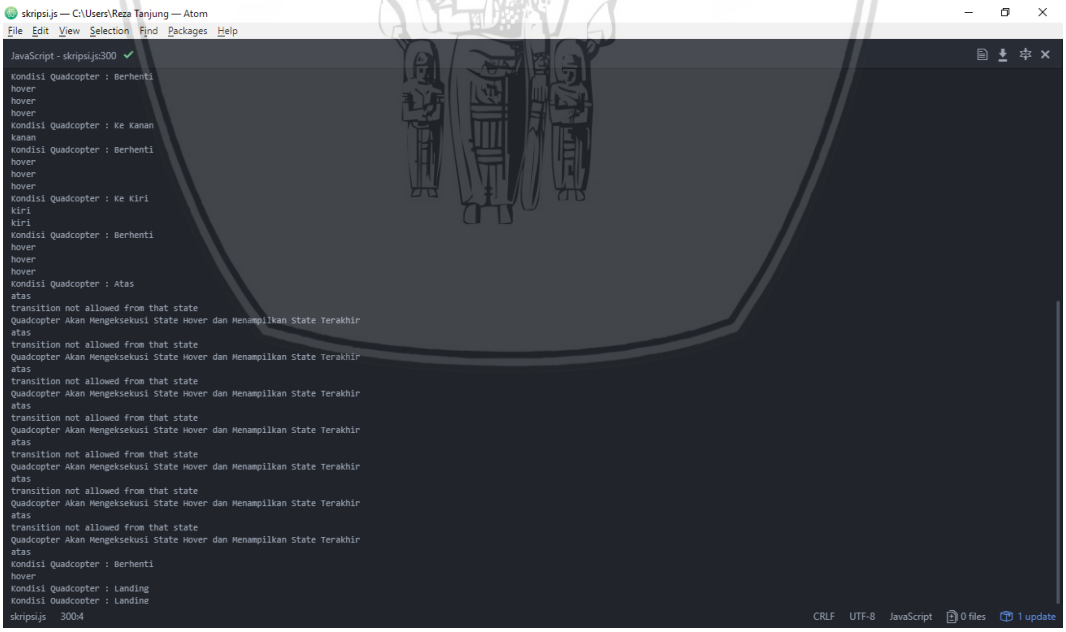
j. Gerakan lasso tangan kanan.
   Source code dibawah ini digunakan untuk menampilkan kondisi lasso tangan kanan pada output panel ATOM ketika pengguna melakukan pergerakan takeoff.

```
else if (lKanan) {
client.takeoff();
console.log("Kondisi Quadcopter : Take Off");
}
```

k. Gerakan tangan lasso kekiri.
   Source code dibawah ini digunakan untuk menampilkan kondisi lasso tangan kiri pada output panel ATOM ketika pengguna melakukan pergerakan landing.

```
else if (lKiri) {
client.land();
console.log("Kondisi Quadcopter : Landing")
)
```

   Keseluruhan hasil output sistem ketika pengguna melakukan kombinasi gerakan dapat dilihat pada gambar 5.23. Pada gambar tersebut pengguna melakukan 14 kombinasi gerakan dan melakukan 7 kali gerakan error.



```
Kondisi Quadcopter : Berhenti
hover
hover
hover
Kondisi Quadcopter : Ke Kanan
kanan
Kondisi Quadcopter : Berhenti
hover
hover
hover
Kondisi Quadcopter : Ke Kiri
kiri
kiri
Kondisi Quadcopter : Berhenti
hover
hover
hover
Kondisi Quadcopter : Atas
atas
transition not allowed from that state
Quadcopter Akan Mengeksekusi State Hover dan Menampilkan State Terakhir
atas
transition not allowed from that state
Quadcopter Akan Mengeksekusi State Hover dan Menampilkan State Terakhir
atas
transition not allowed from that state
Quadcopter Akan Mengeksekusi State Hover dan Menampilkan State Terakhir
atas
transition not allowed from that state
Quadcopter Akan Mengeksekusi State Hover dan Menampilkan State Terakhir
atas
transition not allowed from that state
Quadcopter Akan Mengeksekusi State Hover dan Menampilkan State Terakhir
atas
transition not allowed from that state
Quadcopter Akan Mengeksekusi State Hover dan Menampilkan State Terakhir
atas
transition not allowed from that state
Quadcopter Akan Mengeksekusi State Hover dan Menampilkan State Terakhir
atas
Kondisi Quadcopter : Berhenti
hover
Kondisi Quadcopter : Landing
Kondisi Quadcopter : Landing
```

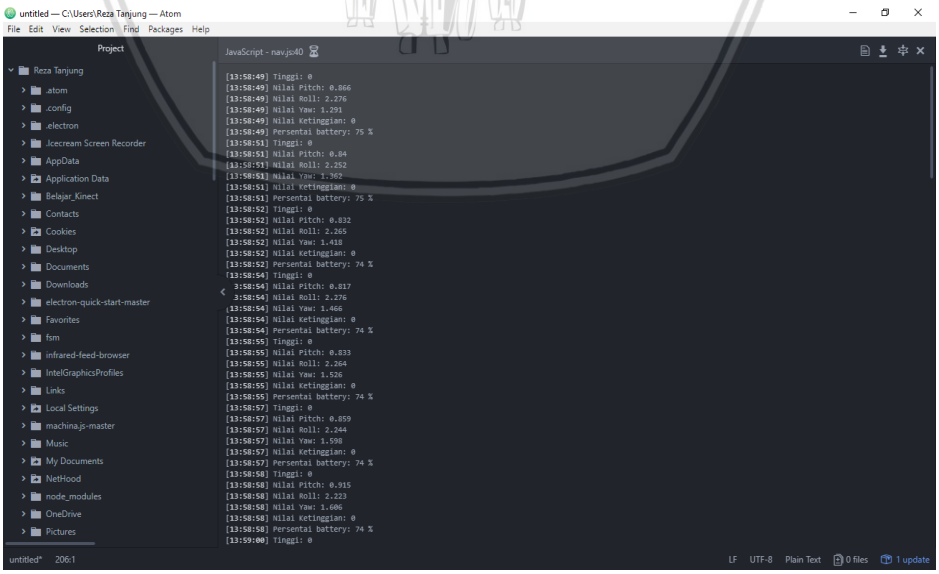
Gambar 5.23 Output Sistem Ketika Pengguna Melakukan Kombinasi Gerakan

2. Antarmuka untuk menampilkan data navigasi *quadcopter*.

   Sistem juga akan menampilkan data navigasi berupa sudut roll, pitch, yaw, ketinggian secara realtime dengan menggunakan source code sebagai berikut:

```
1  var arDrone = require('ar-drone')
2  var tty = require('tty')
3  var client = arDrone.createClient()
4
5  var nowMs = new Date().getTime();
6  lastwaktu = nowMs;
7
8  client.on('navdata', function(d){
9          if(d.demo){
10         lastwaktu = lastwaktu + 1;
11         if (lastwaktu - nowMs > 100) {
12           lastwaktu = nowMs;
13           log("Tinggi: " + d.demo.altitudeMeters);
14           log("Nilai Pitch: " + d.demo.frontBackDegrees);
15           log("Nilai Roll: " + d.demo.leftRightDegrees);
16           log("Nilai Yaw: " +  d.demo.clockwiseDegrees);
17           log("Nilai Ketinggian: " + d.demo.altitudeMeters);
18           log("Persentai battery: " + d.demo.batteryPercentage
19 + " %");
20
21               }
22             }
23           });
```

Source code pada bari pertama hingga keempat digunakan untuk memanggil fungsi *ar.drone* yang terdapat pada Node Package Manager(NPM) kemudian melakukan inisialisasi variabel baru dengan nama *client* untuk memanggil kelas *ar drone*. Kemudian *source code* pada baris kelima hingga kedua belas digunakan untuk membuat delay waktu pengambilan data sebesar 100 ms. Lalu *syntax* pada baris ketiga belas hingga kedua puluh tiga digunakan untuk menampilkan data navigasi berupa sudut *pitch*, *roll*, *yaw*, ketinggian, dan status persentase baterai secara berurut. Hasil antarmuka sistem pada panel output ATOM pada dilihat pada gambar 5.24.

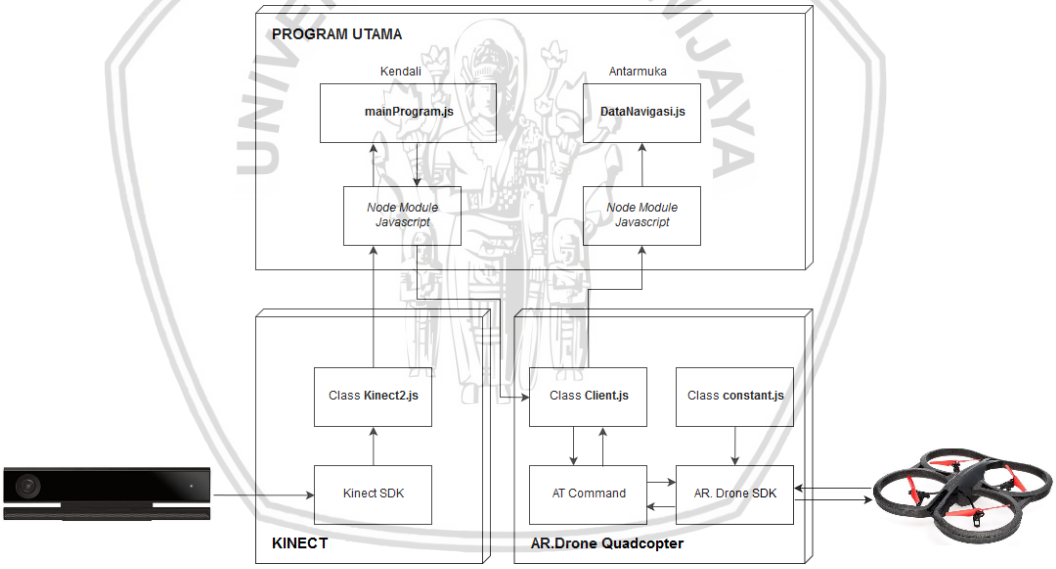


Gambar 5.24 Data Navigasi *Quadcopter*

Pada gambar tersebut *quadcopter* sedang melakukan instruksi pergerakan sebanyak tujuh kali yang menghasilkan nilai sudut *pitch* sebesar 0.86, sudah *roll* sebesar 2.27. Sudut *yaw* sebesar 1.29, terbang pada ketinggian 0 meter , dan memiliki kapasitas baterai sebesar 74 %.

5.2.5 Implementasi Komunikasi Sistem

Pada subbab ini membahas mengenai proses alur implementasi pada komunikasi yang terjadi dalam penelitian ini. Alur keseluruhan komunikasi sistem ditunjukkan pada gambar 5.25. *Script* program utama pada sistem ini diberi nama dengan *class* mainProgram.js yang bertujuan untuk memberikan instruksi pergerakan pada *quadcopter* dengan berdasarkan pada nilai pergerakan joint yang dikirimkan oleh pengguna melalui sensor kinect. *Class* mainProgram.js juga bertujuan untuk menerapkan metode *finite state machine* yang telah dirancang pada sub bab perancangan *finite state machine* sebelumnya. Data pergerakan joint dari pengguna akan dideteksi oleh sensor motion pada kinect dan nilai pergerakan joint tersebut akan diolah pada komputer dengan menggunakan SDK yang telah disediakan oleh microsoft. Nilai joint dari pengguna diakses melalui script program berbasis javascript dengan nama class kinect2.js.



Gambar 5.25 Implementasi komunikasi sistem

Keseluruhan komunikasi data yang terjadi pada sistem difokuskan pada class client.js. Class ini bertanggung jawab atas proses komunikasi yang terjadi antara komputer dengan *quadcopter*. pada class ini juga bertanggung jawab atas pengaturan instruksi yang akan dilakukan pada *quadcopter* dan pengiriman data navigasi yang terjadi pada *quadcopter* ke komputer.

5.2.5.1 Komunikasi Data Dari Kinect ke Komputer

Agar nilai data joint pada pengguna yang telah dideteksi oleh sensor motion kinect dapat digunakan pada sistem, maka dibuatlah class kinect2.js. yang dimana pada class ini memiliki fungsi untuk memetakan keseluruhan data joint pada tubuh manusia. Seluruh data joint pada user dipetakan dalam bentuk enumerasi, yang

dimana enumerasi yang dilakukan mengacu pada dokumentasi pemograman microsoft kinect versi dua yang tersedia pada website microsoft developer https://msdn.microsoft.com/en-us/library/windowspreview.kinect.jointype.aspx.

Pendeklarasian sebuah fungsi dengan nama kinect 2 dilakukan pada syntax baris pertama hingga keenam belas. *Syntax* pada baris kesepuluh hingga ketiga puluh enam memiliki fungsi untuk melakukan pemetaan dari data joint pada user dalam bentuk enumerasi dengan total keseluruhan sebanyak 25 tipe joint. Sedangkan pada syntax ketiga puluh tujuh hingga keempat puluh tiga memiliki fungsi untuk mengkondisikan keadaan tangan pada *user* menjadi 5 kondisi yang dideklarasikan pada object kinect2.handstate.

```
1  function Kinect2() {
2          self = this;
3          events.EventEmitter.call(this);
4
5          this.nativeKinect2 = require(binding_path);
6  }
7
8  util.inherits(Kinect2, events.EventEmitter);
9
10 Kinect2.JointType = {
11         spineBase           : 0,
12         spineMid            : 1,
13         neck                : 2,
14         head                : 3,
15         shoulderLeft        : 4,
16         elbowLeft           : 5,
17         wristLeft           : 6,
18         handLeft            : 7,
19         shoulderRight       : 8,
20         elbowRight          : 9,
21         wristRight          : 10,
22         handRight           : 11,
23         hipLeft             : 12,
24         kneeLeft            : 13,
25         ankleLeft           : 14,
26         footLeft            : 15,
27         hipRight            : 16,
28         kneeRight           : 17,
29         ankleRight          : 18,
30         footRight           : 19,
31         spineShoulder       : 20,
32         handTipLeft         : 21,
33         thumbLeft           : 22,
34         handTipRight        : 23,
35         thumbRight          : 24
36 };
37 Kinect2.HandState = {
38         unknown       : 0,
39         notTracked    : 1,
40         open          : 2,
41         closed        : 3,
42         lasso         : 4
43 };
```

Syntax pada baris ke pertama hingga keempat dideklarasikan sebuah fungsi dengan nama openBodyReader.js yang berguna untuk mengirimkan sensor motion

kinect untuk mengambil data joint sceleton pada pengguna yang berada didepan kinect. Pada class kinect2.js ini tedapat sebuah fungsi yang berfungsi untuk memberhentikan pembacaan data joint sceleton pada user yang dideklaraksikan dengan nama closeBodyReader yang berada pada syntax baris keenam hingga kedelapan. Syntax pada baris keempat belas memiliki fungsi untuk memperbolehkan class kinect2.js untuk diakses pada class lain yang membutuhkan pembacaan data *joint sceleton* pada pengguna.

```
1  Kinect2.prototype.openBodyReader = function() {
2         return
3  this.nativeKinect2.openBodyReader(this.bodyFrameCallback);
4  };
5
6  Kinect2.prototype.closeBodyReader = function() {
7         return this.nativeKinect2.closeBodyReader();
8  };
9
10 Kinect2.prototype.bodyFrameCallback = function(data) {
11        self.emit('bodyFrame', data);
12 };
13
14 module.exports = Kinect2;
```

5.2.5.2 Komunikasi Data Dari Komputer ke *Quadcopter*

Keseluruhan komunikasi yang terjadi antara komputer dan *quadcopter* diatur pada class client.js. Fungsi utama yang terdapat pada class client.js dijelaskan pada bagian ini.

Syntax pada baris pertama hingga keempat berguna untuk menghubungkan class client.js dengan class udpControl.js, repl.js, dan udpNavDataStream.js. Class udpControl.js dan repl.js berguna untuk mengirimkan AT Command. Sedangkan Class udpNavData.js berguna untuk menerima nilai data navigasi pada *quadcopter*. Syntax pada baris keempat berfungsi untuk memperbolehkan client.js untuk dapat diakses pada class yang lain.

```
1  Client.UdpControl = require('./control/UdpControl');
2  Client.Repl = require('./Repl');
3  Client.UdpNavdataStream = require('./navdata/UdpNavdataStream');
4  module.exports = Client;
```

Pengaturan pada format pengiriman instruksi AT Command pada *quadcopter* dideklarasikan pada fungsi dibawah ini. Syntax pada baris pertama hingga kelima belas berguna untuk mengirimkan 2 jenis AT Command yaitu AT*REF dan AT*PCMD yang dipeloreh melalui program utama menuju ke class udpControl.js. Syntax pada baris ketujuh belas hingga kedua puluh satu berguna untuk mengirimkan instruksi take off pada *quadcopter* kemudian membuat state pada *quadcopter* menjadi hover. Kemudian *syntax* pada baris kedua puluh tiga hingga kedua puluh tujuh berguna untuk mengirimkan instruksi landing pada *quadcopter* dan membuat state pada *quadcopter* menjadi landing. Selanjutnya *syntax* pada baris kedua puluh sembilan hingga ketiga puluh dua berguna untuk mengirimkan instruksi stop pada *quadcopter* dan menyebabkan *quadcopter* melakukan state berhenti.

```
1  Client.prototype._sendCommands = function() {
2    this._udpControl.ref(this._ref);
3    this._udpControl.pcmd(this._pcmd);
4    this._udpControl.flush();
5
6    this._repeaters
7      .forEach(function(repeat) {
8      repeat.times--;
9      repeat.method();
10     });
11
12     this._repeaters = this._repeaters.filter(function(repeat) {
13       return repeat.times > 0;\
14     });
15 };
16
17 Client.prototype.takeoff = function(cb) {
18   this.once('hovering', cb || function() {});
19   this._ref.fly = true;
20   return true;
21 };
22
23 Client.prototype.land = function(cb) {
24   this.once('landed', cb || function() {});
25   this._ref.fly = false;
26   return true;
27 };
28
29 Client.prototype.stop = function() {
30 this._pcmd = {};
31 return true;
32 };
```

Proses mapping pada AT*PCMD dari perogram utama dideklarasikan pada class dibawah ini. Syntax pada baris pertama hingga kelima berguna untuk mendeklarasikan sebuah variabel dengan nama pcmdOption yang berisi empat pasang array yaitu atas dan bawah, kiri dan kanan, depan dan belakang, putar kiri dan putar kanan. fungsi ini digunakan pada *quadcopter* sesuai dari instruksi yang akan dilakukan oleh user. Sebagai contoh jika pengguna mengirimkan instruksi maju, maka array front diaktifkan. Lalu pada syntax baris kesepuluh hingga kedua puluh berguna untuk melakukan pengkondisian pada pasangan array tersebut. Sebagai contoh jika pengguna melakukan instruksi maju pada *quadcopter*, maka nilai pada variabel up akan aktif, sedangkan pasangan variabelnya yaitu down akan diabaikan. Nilai kecepatan pada variabel up yang diterima diubah dari bentuk decimal manjadi *float*. Sedangkan syntax pada baris kedua puluh empat hingga ketiga puluh empat berguna untuk mengontrol variabel down yang dimana variabel ini akan aktif jika pengguna memberikan instruksi down pada *quadcopter*. Pada fungsi ini, nilai variabel down juga akan diubah dari bentuk desimal ke dalam bentuk *float*.

```
1  var pcmdOptions = [
2    ['up', 'down'],
3    ['left', 'right'],
4    ['front', 'back'],
5    ['clockwise', 'counterClockwise'],
6  ];
7
8  pcmdOptions.forEach(function(pair) {
```

```
  Client.prototype[pair[0]] = function(speed) {
    if (isNaN(speed)) {
      return;
    }

    speed = parseFloat(speed);
    this._pcmd[pair[0]] = speed;
    delete this._pcmd[pair[1]];
    return speed;
};

Client.prototype[pair[1]] = function(speed) {
    if (isNaN(speed)) {
      return;
    }

    speed = parseFloat(speed);
    this._pcmd[pair[1]] = speed;
    delete this._pcmd[pair[0]];
    return speed;
};
```

Data navigasi pada sistem diatur pada class dibawah ini. *Syntax* pada baris pertama hingga kesebelas digunakan untuk melakukan reset pada konfigurasi ACK dan juga melakukan *request* data navigasi pada *quadcopter* dengan waktu interval sebesar 30 ms. Pada *syntax* baris keenam, semua data navigasi yang telah diterima pada sistem akan dihapus dengan mengehentikan semua kegiatan pengambilan data navigasi pada quadcopter. Kemudian pada *syntax* baris ketujuh, sistem akan memulai nenangkap data navigasi pada quadcopter pada sesi yang baru.

```
Client.prototype.resume = function() {
  this._udpControl.ctrl(5, 0);
  this.config('general:navdata_demo', 'TRUE');
  this.disableEmergency();
  this._setInterval(30);
  this._udpNavdatasStream.removeAllListeners();
  this._udpNavdatasStream.resume();
  this._udpNavdatasStream
    .on('error', this._maybeEmitError.bind(this))
    .on('data', this._handleNavdata.bind(this));
};
```

BAB 6 PENGUJIAN DAN ANALISIS

Pada bab ini akan dilakukan pengujian dari sistem yang sudah dirancang dan diimplementasikan sebelumnya. Pengujian terdiri dari 3 bagian, yakni pengujian metode *finite state machine*, pengujian ketepatan gerakan, dan pengujian *delay* sistem.

6.1 Pengujian metode *finite state machine*

6.1.1 Tujuan Pengujian

Pengujian ini dilakukan untuk mengetahui apa pengaruh metode *finite state machine* terhadap sistem navigasi dan pengendalian *AR.Drone quadcopter* dengan menggunakan sensor *kinect*. Pada pengujian ini juga bertujuan untuk mengetahui perbandingan performa antara sistem yang menerapkan metode *finite state machine* dengan sistem yang tidak menerapkan metode *finite state machine* dalam segi kendali dan navigasi pergerakan pada quadcopter.

6.1.2 Pelaksanaan Pengujian

Pengujian ini dilaksanakan dengan melakukan dua buah set kombinasi gerakan. Pada dua buah set kombinasi pergerakan yang pertama, pengujian dilakukan dengan menggunakan metode *finite state machine*. Pada setiap pengujian set kombinasi gerakan, pengguna diberikan Instruksi untuk melakukan kombinasi sepuluh pergerakan tubuh yang digunakan dalam pengendalian *quadcopter* secara berturut dengan jeda waktu tidak lebih dari satu detik. Kemudian Setelah pengguna melakukan kombinasi sepuluh gerakan, pengguna diinstruksikan untuk melakukan empat pergerakan yang tidak dirancang didalam sistem seperti gerakan memutar ke dua tangan searah jarum jam atau berputar ke kanan sebesar 360 derajat. Setelah itu akan dilakukan analisis terhadap hasil pergerakan pada *quadcopter* yang berdasarkan dari sepuluh kombinasi gerakan dan empat pergerakan yang tidak dirancang didalam sistem tersebut.

6.1.3 Prosedur Pengujian

Agar pengujian dapat dilakukan dengan optimal maka pengujian ini dilakukan dengan prosedure berikut:

1. Menghubungkan sensor kinect dengan komputer menggunakan kabel USB versi 3.0.
2. Memasang batteray lithium pada *quadcopter* dan menunggu hingga *quadcopter* melakukan kablirasi dengan indikasi suare beep sebanyak empat kali. Setelah *quadcopter* malakukan kablirasi maka *quadcopter* siap untuk digunakan.
3. Melakukan hubungan koneksi jaringan dari PC ke *quadcopter* melalui koneksi *wi-fi.*
4. Membuka command promt node.js pada komputer dan mengetik syntax "node kendaliNavigasiDrone.js"

5. Melakukan dua buah set kombinasi gerakan tubuh dengan tidak menggunakan metode *finite state machine*. Pada setiap set kombinasi tersebut, pengguna diberikan Instruksi untuk melakukan kombinasi sepuluh pergerakan tubuh yang digunakan dalam pengendalian *quadcopter* secara berturut dengan jeda waktu tidak lebih dari satu detik kemudian diikuti dengan empat buah gerakan yang tidak dirancang pada sistem seperti berputar ke arah kanan sebesar 360 derajat.
6. Melakukan dua buah set kombinasi gerakan tubuh dengan menggunakan metode *finite state machine*. Pada setiap set kombinasi tersebut, pengguna diberikan Instruksi untuk melakukan kombinasi sepuluh pergerakan tubuh yang digunakan dalam pengendalian *quadcopter* secara berturut dengan jeda waktu tidak lebih dari satu detik kemudian diikuti dengan empat buah gerakan yang tidak dirancang pada sistem.
7. Mencatat hasil nilai keluaran pada sistem yang ditampilkan pada *output panel* ATOM kemudian memindahkan hasil nilai tersebut ke dalam ms. Word.

6.1.4 Hasil Pengujian

Setelah pengujian pada empat buah set kombinasi gerakan telah dilakukan, maka didapatkan hasil data berupa berapa kali *quacopter* bertabrakan pada object yang ada disekitar *quadcopter* tersebut seperti kursi ataupun dinding. Penyebab dari *quadcopter* bertabrakan dengan object lain seperti dinding ataupun kursi yaitu pengguna melakukan serangkaian kombinasi gerakan yang diikuti dengan empat buah pergerakan yang tidak dirancang pada sistem pada saat melakukan pengujian kombinasi gerakan dengan sistem yang tidak menggunakan metode *finite state machine*.

1. Kombinasi gerakan tanpa menggunakan metode *finite state machine* pertama.

   Hasil pengujian dari pengujian kombinasi gerakan yang dilakukan sebanyak empat belas kali tanpa menggunakan metode *finite state machine* (FSM) dapat dilihat pada tabel 6.1.

Tabel 6.1 Hasil pengujian kombinasi gerakan pertama tanpa FSM

Pergerakan ke-	Instruksi gerakan	Kondisi *quadcopter*	Terjadi tabrakan	*Quadcopter* dalam keadaan optimal
1	Hover	Hover	Tidak	Ya
2	Maju	Maju	Tidak	Ya
3	Atas	Atas	Tidak	Ya
4	Bawah	Bawah	Tidak	Ya
5	Kiri	Kiri	Tidak	Ya
6	Kanan	Kanan	Ya	Ya
7	Kiri	Kiri	Tidak	Ya

8	Hover	Hover	Tidak	Ya
9	Kanan	Kanan	Tidak	Ya
10	Kiri	Kiri	Tidak	Ya
11	Berputar ke kanan sebesar 360 derajat	Kanan	Tidak	Ya
12	Berputar ke kanan sebesar 360 derajat	Maju	Ya	Tidak
13	Berputar ke kiri sebesar 360 derajat	Nonaktif	-	Tidak
14	Berputar ke kiri sebesar 360 derajat	Nonaktif	-	Tidak

Setelah pengujian kombinasi gerakan yang dilakukan sebanyak empat belas kali tanpa menggunakan metode *finite state machine* dilakukan, dipeloreh hasil pergerakan quadcopter pada tabel 6.1 yang dimana terjadi tabrakan terhadap objek yang berada pada sekitar *quadcopter* yaitu dinding ruangan yang menyebabkan *quadcopter* tersebut menjadi nonaktif sehingga tidak dapat dikendalikan lebih lanjut.

2. Kombinasi gerakan tanpa menggunakan metode *finite state machine* kedua.

Hasil pengujian dari pengujian kombinasi gerakan yang dilakukan sebanyak empat belas kali tanpa menggunakan metode *finite state machine* (FSM) dapat dilihat pada tabel 6.2.

Tabel 6.2 Hasil pengujian kombinasi gerakan kedua tanpa FSM

Pergerakan ke-	Instruksi gerakan	Kondisi *quadcopter*	Terjadi tabrakan	*Quadcopter* dalam keadaan optimal
1	Hover	Hover	Tidak	Ya
2	Atas	Atas	Tidak	Ya
3	Kiri	Kiri	Tidak	Ya
4	Kanan	Kanan	Tidak	Ya
5	Kiri	Kiri	Tidak	Ya
6	Hover	Hover	Tidak	Ya
7	Kanan	Kanan	Tidak	Ya
8	Kiri	Kiri	Ya	Ya
9	Kanan	Kanan	Tidak	Ya

10	Kiri	Kiri	Tidak	Ya
11	Berputar ke kanan sebesar 360 derajat	Kiri	Tidak	Ya
12	Berputar ke kanan sebesar 360 derajat	Kanan	Tidak	Ya
13	Berputar ke kiri sebesar 360 derajat	Maju	Ya	Tidak
14	Berputar ke kiri sebesar 360 derajat	Nonaktif	-	Tidak

Setelah pengujian kombinasi gerakan yang dilakukan sebanyak empat belas kali tanpa menggunakan metode *finite state machine* dilakukan, dipeloreh hasil pergerakan quadcopter pada tabel 6.2 yang dimana terjadi tabrakan terhadap objek yang berada pada sekitar *quadcopter* yaitu dinding ruangan yang menyebabkan *quadcopter* tersebut menjadi nonaktif sehingga tidak dapat dikendalikan lebih lanjut.

3. Kombinasi gerakan menggunakan metode *finite state machine* ketiga.

Hasil pengujian dari pengujian kombinasi gerakan yang dilakukan sebanyak empat belas kali dengan menggunakan metode *finite state machine* (FSM) dapat dilihat pada tabel 6.3.

Tabel 6.3 Hasil pengujian kombinasi gerakan pertama dengan FSM

Pergerakan ke-	Instruksi gerakan	Kondisi *quadcopter*	Terjadi tabrakan	*Quadcopter* dalam keadaan optimal
1	Hover	Hover	Tidak	Ya
2	Atas	Atas	Tidak	Ya
3	Kanan	Hover	Tidak	Ya
4	Kiri	Hover	Tidak	Ya
5	Kanan	Hover	Tidak	Ya
6	Kiri	Hover	Tidak	Ya
7	Hover	Hover	Tidak	Ya
8	Kanan	Kanan	Tidak	Ya
9	Kiri	Hover	Tidak	Ya
10	Kanan	Hover	Tidak	Ya

11	Berputar ke kanan sebesar 360 derajat	Hover	Tidak	Ya
12	Berputar ke kanan sebesar 360 derajat	Hover	Tidak	Ya
13	Berputar ke kiri sebesar 360 derajat	Hover	Tidak	Ya
14	Berputar ke kiri sebesar 360 derajat	Hover	Tidak	Ya

Setelah pengujian kombinasi gerakan yang dilakukan sebanyak empat belas kali dengan menggunakan metode *finite state machine* dilakukan, dipeloreh hasil pergerakan quadcopter pada tabel 6.3 yang dimana tidak terjadi tabrakan terhadap objek yang berada pada sekitar *quadcopter*.

4. Kombinasi gerakan menggunakan metode finite state machine keempat.

Hasil pengujian dari pengujian kombinasi gerakan yang dilakukan sebanyak empat belas kali dengan menggunakan metode *finite state machine* (FSM) dapat dilihat pada tabel 6.4.

Tabel 6.4 Hasil pengujian kombinasi gerakan kedua dengan FSM

Pergerakan ke-	Instruksi gerakan	Kondisi *quadcopter*	Terjadi tabrakan	*Quadcopter* dalam keadaan optimal
1	Hover	Hover	Tidak	Ya
2	Maju	Maju	Tidak	Ya
3	Kanan	Hover	Tidak	Ya
4	Kiri	Hover	Tidak	Ya
5	Kanan	Hover	Tidak	Ya
6	Kiri	Hover	Tidak	Ya
7	Hover	Hover	Tidak	Ya
8	Atas	Atas	Tidak	Ya
9	Bawah	Hover	Tidak	Ya
10	Atas	Hover	Tidak	Ya
11	Berputar ke kanan sebesar 360 derajat	Hover	Tidak	Ya

12	Berputar ke kanan sebesar 360 derajat	Hover	Tidak	Ya
13	Berputar ke kiri sebesar 360 derajat	Hover	Tidak	Ya
14	Berputar ke kiri sebesar 360 derajat	Hover	Tidak	Ya

Setelah pengujian kombinasi gerakan yang dilakukan sebanyak empat belas kali dengan menggunakan metode *finite state machine* dilakukan, dipeloreh hasil pergerakan quadcopter pada tabel 6.4 yang dimana tidak terjadi tabrakan terhadap objek yang berada pada sekitar *quadcopter*.

6.1.5 Analisis Pengujian

Berdasarkan dari hasil pengujian metode *finite state machine* diatas, maka dapatkan data bahwa pada pengujian kombinasi gerakan yang dilakukan sebanyak empat belas kali dengan tidak menggunakan metode *finite state machine*, dipeloreh hasil pergerakan *quadcopter* yang dimana terjadi tabrakan terhadap objek yang berada pada sekitar *quadcopter*. Kemudian pada pengujian kombinasi gerakan yang dilakukan sebanyak empat belas kali dengan menggunakan metode *finite state machine*, dipeloreh hasil pergerakan quadcopter yang dimana tidak terjadi tabrakan terhadap objek yang berada pada sekitar *quadcopter*. Pada pengujian kombinasi gerakan set pertama dan kedua, *quadcopter* mengalami tabrakan dengan object yang ada disekitar *quadcopter* tersebut yaitu dinding ruangan sebanyak dua kali. Pada tabrakan dengan dinding ruangan yang pertama, *quadcopter* masih dalam keadaan optimal dan masih bisa untuk dioperasikan. Namun pada tabrakan dengan dinding ruangan untuk kedua kalinya, posisi badan *quadcopter* terbalik dan tergeletak ditanah yang menyebabkan *quadcopter* tersebut tidak dalam keadaan optimal untuk dioprasikan dan tidak dapat menerima instruksi dari pengguna.

6.2 Pengujian Ketepatan Gerakan

6.2.1 Tujuan Pengujian

Pengujian ini dilakukan untuk mengetahui apakah sistem pergerakan tubuh yang dirancang sebelumnya sudah berjalan dengan optimal dan mampu memberikan hasil output yang tepat.

6.2.2 Pelaksanaan Pengujian

Pengujian ketepatan gerakan dilakukan dengan beberapa kondisi yaitu pengujian dilakukan dengan jarak 1.2 meter dari sensor kinect dengan jumlah pengujian sebanyak 10 kali pada pengguna. Untuk setiap gerakan yang dilakukan oleh pengguna, dapat dilihat pada *command promt* pada layar komputer. Pada layar komputer juga memantau kondisi pada *quadcopter* yang sedang berjalan sehingga

dapat melakukan pemantauan apakah instruksi pada *quadcopter* sudah sesuai dengan pergerakan tubuh yang dilakukan oloh pengguna.

6.2.3 Prosedur Pengujian

Agar pengujian dapat dilakukan dengan optimal maka pengujian ini dilakukan dengan prosedure berikut:

1. Menghubungkan sensor kinect dengan komputer menggunakan kabel USB versi 3.0.
2. Memasang batteray lithium pada *quadcopter* dan menunggu hingga *quadcopter* melakukan kablirasi dengan indikasi suare *beep* sebanyak empat kali. Setelah *quadcopter* malakukan kablirasi maka *quadcopter* siap untuk digunakan.
3. Melakukan hubungan koneksi jaringan dari PC ke *quadcopter* melalui koneksi wi-fi.
4. Membuka command promt node.js pada komputer dan mengetik syntax "node kendaliNavigasiDrone.js".
5. Pengguna mempersiapkan posisi tubuh di depan sensor kinect dengan jarak sebesar 1.2 meter.
6. Pengguna melakukan pergerakan tubuh sesuai dengan instruksi yang telah ditentukan sebelumnya dengan tujuan memberikan instruksi pergerakan pada *quadcopter* seperti hover, pitch, yaw, gaz, takeoff, dan landing.
7. Memantau pada komputer apakah *quadcopter* telah bergerak sesuai dengan instruksi yang diberikan oleh pengguna.
8. Menyalin data navigasi pada *quadcopter* seperti data batteray, besar nilai pich, yaw, gaz, dan lain sebagainya ke dalam micrososft excel dan mengamati hasil nilainya.
9. Mengulangi langkah ke enam sebanyak sepuluh kali pada pengguna dan mencatat hasilnya.

6.2.4 Hasil Pengujian

Setelah semua gerakan yang dirancang pada *quadcopter* dilakukan pengujianm didapatkan hasil keluaran sebagai berikut:

a. Pengujian gerakan *pitch*.

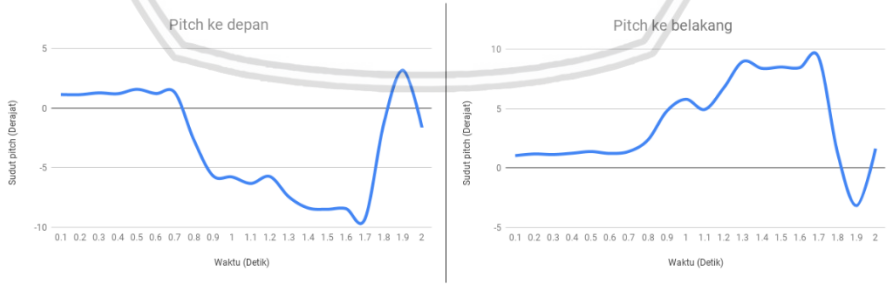
   Pengujian gerakan *pitch* dilakukan dengan melakukan pengujian gerakan ke arah depan dan ke arah belakang pada *quadcopter*. Untuk pengujian pitch dengan gerakan *quadcopter* ke arah depan dapat dilihat pada gambar 6.1. Pada gambar 6.1 bagian (a) pengguna melakukan pergerakan tubuh dengan tujuan untuk memberikan instruksi pergerkana pitch ke depan pada *quadcopter*. Kemudian pada gambar 6.1 bagian (b) *quadcopter* melakukan instruksi pergerakan pitch ke depan sesuai dengan input yang diberikan oleh pengguna.

Gambar 6.1 Pengujian Pergerakan Ke Depan

Untuk pengujian pitch dengan gerakan *quadcopter* ke arah belakang dapat dilihat pada gambar 6.2. Pada gambar 6.2 bagian (a) pengguna melakukan pergerakan tubuh dengan tujuan untuk memberikan instruksi pergerakan pitch ke belakang pada *quadcopter*. Kemudian pada gambar 6.2 bagian (b) *quadcopter* melakukan instruksi pergerakan pitch ke belakang sesuai dengan input yang diberikan oleh pengguna.

Gambar 6.2 Pengujian Pergerakan Ke Belakang

Setelah dilakukan pengujian pitch ke depan dan ke belakang, dihasilkan nilai yang ditampilkan dengan grafik pada gambar 6.3. Pada gambar 6.3 diperlihatkan perubahan nilai sudut pada saat pengujian pergerakan pitch ke depan dan ke belakang dilakukan. Ketika pengujian pergerakan pitch ke depan dilakukan, nilai sudut cenderung bernilai negatif. Sedangkan ketika pengujian pergerakan pitch ke belakang dilakukan, nilai sudut cenderung bernilai positif.



Gambar 6.3 Hasil nilai gerakan *pitch* pada *output panel*

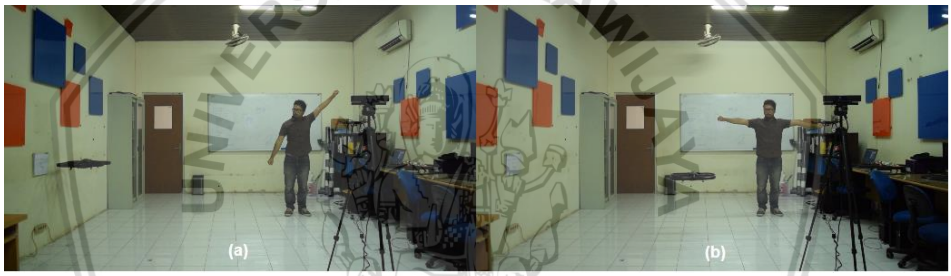
b. Pengujian gerakan roll.

Pengujian gerakan roll dilakukan dengan melakukan gerakan ke arah kanan dan ke arah kiri pada *quadcopter*. Untuk pengujian roll dengan gerakan *quadcopter* kearah kanan dapat dilihat pada gambar 6.4. Pada gambar 6.4 bagian (a) pengguna melakukan pergerakan tubuh dengan tujuan untuk memberikan instruksi pergerakan roll ke arah kanan pada *quadcopter*. Kemudian pada gambar

6.4 bagian (b) *quadcopter* melakukan instruksi pergerakan roll ke arah kanan sesuai dengan input yang diberikan oleh pengguna.

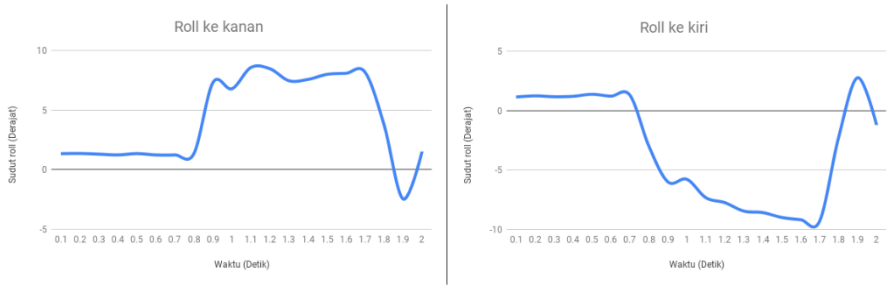
Gambar 6.4 Pengujian Pergerakan Ke Kanan

Untuk pengujian roll dengan gerakan *quadcopter* ke arah kiri dapat dilihat pada gambar 6.5. Pada gambar 6.5 bagian (a) pengguna melakukan pergerakan tubuh dengan tujuan untuk memberikan instruksi pergerakan roll ke arah kiri pada *quadcopter*. Kemudian pada gambar 6.5 bagian (b) *quadcopter* melakukan instruksi pergerakan roll ke arah kiri sesuai dengan input yang diberikan oleh pengguna.



Gambar 6.5 Pengujian Pergerakan Ke Kiri

Setelah dilakukan pengujian roll ke arah kanan dan kiri, dihasilkan nilai yang ditampilkan dengan grafik pada gambar 6.6. Pada gambar 6.6 diperlihatkan perubahan nilai sudut pada saat pengujian pergerakan roll ke kanan dan ke kiri dilakukan. Ketika pengujian pergerakan roll ke kanan dilakukan, nilai sudut cenderung bernilai positif. Sedangkan ketika pengujian pergerakan roll ke kiri dilakukan, nilai sudut cenderung bernilai negatif.



Gambar 6.6 Hasil nilai gerakan *roll* pada *output panel*

c. Pengujian gerakan gaz.

Pengujian gerakan gaz dilakukan dengan melakukan gerakan ke arah atas dan ke arah bawah pada *quadcopter*. Untuk pengujian gaz dengan gerakan *quadcopter*

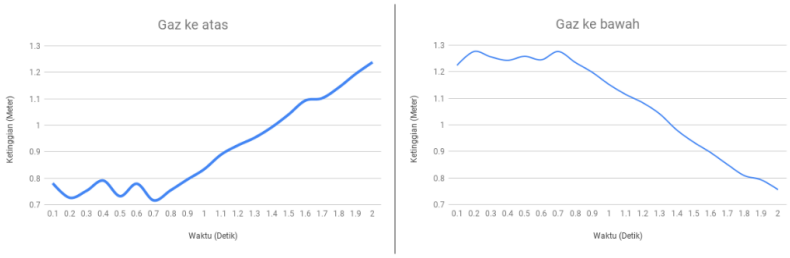
ke arah atas dapat dilihat pada gambar 6.7 Pada gambar 6.7 bagian (a) pengguna melakukan pergerakan tubuh dengan tujuan untuk memberikan instruksi pergerakan gaz ke arah depan pada *quadcopter*. Kemudian pada gambar 6.7 bagian (b) *quadcopter* melakukan instruksi pergerakan gaz ke arah depan sesuai dengan input yang diberikan oleh pengguna.

Gambar 6.7 Pengujian Pergerakan Ke Atas

Untuk pengujian gaz dengan gerakan *quadcopter* ke arah bawah dapat dilihat pada gambar 6.8. Pada gambar 6.8 bagian (a) pengguna melakukan pergerakan tubuh dengan tujuan untuk memberikan instruksi pergerakan gaz ke arah bawah pada *quadcopter*. Kemudian pada gambar 6.8 bagian (b) *quadcopter* melakukan instruksi pergerakan gaz ke arah bawah sesuai dengan input yang diberikan oleh pengguna.

Gambar 6.8 Pengujian Pergerakan Ke Bawah

Setelah dilakukan pengujian gaz ke atas dan ke bawah, dihasilkan nilai yang ditampilkan dengan grafik pada gambar 6.9. Pada gambar 6.9 diperlihatkan perubahan nilai altitude dalam satuan meter pada saat pengujian pergerakan gaz ke atas dan ke bawah dilakukan. Ketika pengujian pergerakan gaz ke atas dilakukan, nilai altitude semakin meningkat ke nilai positif. Sedangkan ketika pengujian pergerakan gaz ke bawah dilakukan, nilai altitude semakin menurun ke nilai negatif.



Gambar 6.9 Hasil nilai gerakan *gaz* pada *output panel*

d. Pengujian gerakan yaw.

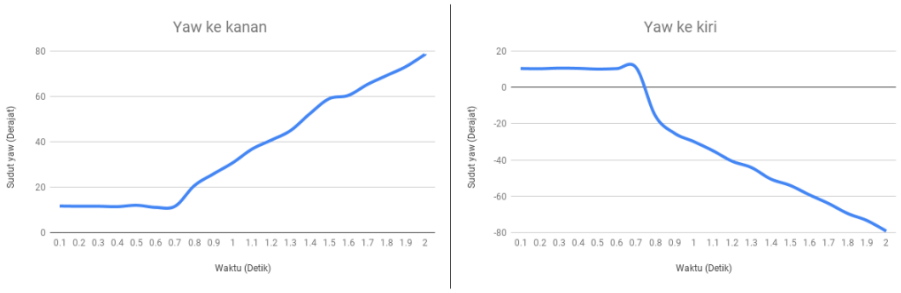
Pengujian gerakan *yaw* dilakukan dengan melakukan pengujian gerakan ke arah searah jarum jam dan ke arah berlawanan jarum jam pada *quadcopter*. Untuk pengujian yaw dengan gerakan *quadcopter* ke arah jarum jam dapat dilihat pada gambar 6.10 Pada gambar 6.10 bagian (a) pengguna melakukan pergerakan tubuh dengan tujuan untuk memberikan instruksi pergerakan yaw ke arah searah jarum jam pada *quadcopter*. Kemudian pada gambar 6.10 bagian (b) *quadcopter* melakukan instruksi pergerakan gaz ke arah searah jarum jam sesuai dengan input yang diberikan oleh pengguna.

Gambar 6.10 Pengujian gerakan yaw ke arah searah jarum jam

Untuk pengujian yaw dengan gerakan *quadcopter* ke arah berlawanan jarum jam dapat dilihat pada gambar 6.11. Pada gambar 6.11 bagian (a) pengguna melakukan pergerakan tubuh dengan tujuan untuk memberikan instruksi pergerakan yaw ke arah berlawanan jarum jam pada *quadcopter*. Kemudian pada gambar 6.11 bagian (b) *quadcopter* melakukan instruksi pergerakan yaw ke arah berlawanan jarum jam sesuai dengan input yang diberikan oleh pengguna.

Gambar 6.11 Pengujian gerakan yaw ke arah berlawanan jarum jam

Setelah dilakukan pengujian yaw ke arah searah jaum jam dan berlawanan jarum jam, dihasilkan nilai yang ditampilkan dengan grafik pada gambar 6.12. Pada gambar 6.12 diperlihatkan perubahan nilai sudut pada saat pengujian pergerakan yaw ke arah searah jarum jam dan ke arah berlawanan jarum jam dilakukan. Ketika pengujian pergerakan yaw ke ke arah searah jarum jam dilakukan, nilai sudut yaw bernilai positif. Sedangkan ketika pengujian pergerakan yaw ke arah berlawanan jarum jam dilakukan, nilai sudut yaw bernilai negatif.



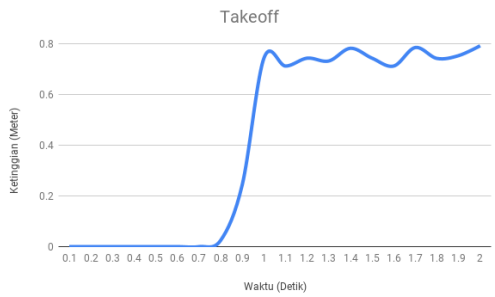
Gambar 6.12 Hasil nilai gerakan yaw pada output panel

e. Pengujian gerakan takeoff.

Untuk pengujian takeoff pada *quadcopter* dapat dilihat pada gambar 6.13. Pada gambar 6.13 bagian (a) pengguna melakukan pergerakan tubuh dengan tujuan untuk memberikan instruksi pergerakan takeoff depan pada *quadcopter*. Kemudian pada gambar 6.13 bagian (b) *quadcopter* melakukan instruksi pergerakan takeoff sesuai dengan input yang diberikan oleh pengguna.

Gambar 6.13 Pengujian Pergerakan *Takeoff*

Setelah dilakukan pengujian takeoff, dihasilkan nilai yang ditampilkan dengan grafik pada gambar 6.14. Pada gambar 6.14 diperlihatkan perubahan nilai altitude dalam satuan meter pada saat pengujian pergerakan takeoff dilakukan. Sebelum pengujian pergerakan takeoff dilakukan, nilai altitude pada *quadcopter* bernilai nol dikarenakan *quadcopter* dalam keadaan tidak terbang atau berada ditanah. Ketika pengujian pergerakan takeoff dilakukan, nilai altitude semakin meningkat dari nol hingga bernilai 0.758. Ketika *quadcopter* berhasil melakukan instruksi takeoff, maka *quadcopter* akan secara otomatis masuk ke dalam keadaan hover dan memiliki nilai altitude diantara 0.71 hingga 0.79.



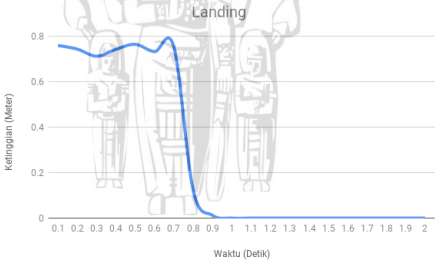
Gambar 6.14 Hasil nilai gerakan *takeoff* pada *output panel*

f. Pengujian gerakan landing.

Untuk pengujian landing pada *quadcopter* dapat dilihat pada gambar 6.15. Pada gambar 6.15 bagian (a) pengguna melakukan pergerakan tubuh dengan tujuan untuk memberikan instruksi pergerakan landing depan pada *quadcopter*. Kemudian pada gambar 6.15 bagian (b) *quadcopter* melakukan instruksi pergerakan landing sesuai dengan input yang diberikan oleh pengguna.

Gambar 6.15 Pengujian Pergerakan *Landing*

Setelah dilakukan pengujian landing, dihasilkan nilai yang ditampilkan dengan grafik pada gambar 6.16. Pada gambar 6.16 diperlihatkan perubahan nilai altitude dalam satuan meter pada saat pengujian pergerakan landing dilakukan. Sebelum pengujian pergerakan landing dilakukan, nilai altitude pada *quadcopter* bernilai 0.71 hingga 0.79. Ketika pengujian pergerakan landing dilakukan, nilai *altitude* semakin menurun dari satu hingga bernilai nol pada saat *quadcopter* berada pada *ground*.



Gambar 6.16 Hasil nilai gerakan *landing* pada *output panel*

Setelah pengguna melakukan percobaan gerakan sebanyak sepuluh kali, maka dipeloreh hasil yang dapat dilihat pada tabel 6.5, Percobaan pada keseluruhan gerakan telah berhasil dilakukan tanpa adanya kesalahan.

Tabel 6.5 Hasil nilai pengujian ketepatan gerakan

	Percobaan ke-									
Gerakan	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Pitch ke depan	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Pitch ke belakang	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Roll ke kanan	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Roll ke kiri	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Gaz ke atas	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Gaz ke bawah	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Yaw ke kanan	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Yaw ke kiri	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Takeoff	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Landing	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

6.2.5 Analisis Hasil Pengujian

Dari pengujian pergerakan *quadcopter* yang telah dilakukan, didapatkan hasil data berupa keberhasilan pengujian ketepatan gerakan oleh pengguna. Persentase keberhasilan pengujian pergerakan *quadcopter* dihitung dengan menggunakan rumus persamaan 6.1.

$$\text{Nilai persentase ketepatan} = \frac{\text{Jumlah gerakan benar}}{\text{Jumlah pengujian}} X\ 100\% \quad \textbf{(6.1)}$$

Ketika pengujian pergerakan telah dilakukan oleh pengguna , didapatkan hasil persentase keberhasilan pada tabel 6.6. Pada jarak ini sistem mampu menghasilkan persentase ketepatan gerakan sebesar 100 %.

Tabel 6.6 Hasil persentase ketepatan gerakan

No	Pergerakan tubuh	Jumlah pergerakan benar	Jumlah pengujian	Persentase ketepatan
1	Pitch ke depan	10	10	100 %
2	Pitch ke belakang	10	10	100 %
3	Roll ke kanan	10	10	100 %
4	Roll ke kiri	10	10	100 %
5	Gaz ke atas	10	10	100 %
6	Gaz ke bawah	10	10	100 %
7	Yaw ke kanan	10	10	100 %
8	Yaw ke kiri	10	10	100 %
9	*Takeoff*	10	10	100 %
10	*Landing*	10	10	100 %

6.3 Pengujian Delay Sistem

6.3.1 Tujuan Pengujian

Pengujian delay waktu sistem dilakukan untuk mengetahui berapa besar total selisih *delay* antara sistem kendali *quadcopter* dengan menggunakan metode *finite state machine* dengan sistem kendali quadcopter tanpa menggunakan metode *finite state* machine yang dilakukan dengan menggunakan sensor *kinect*. Pengujian delay waktu sistem juga dilakukan untuk mengetahui seberapa baik performa pada sistem kendali yang telah dirancang. Indikasi untuk mengetahui seberapa baik performa sistem adalah besar waktu *delay* yang terjadi pada saat pengguna melakukan pergerakan tubuh hingga *quadcopter* melakukan sebuah instruksi pergerakan dari pengguna.

6.3.2 Pelaksanaan Pengujian

Pengujian delay waktu sistem dilakukan dengan mengamati selisih waktu antara gerakan yang diberikan pengguna didepan sensor kamera kinect hingga *quadcopter* yang menerima instruksi dari pengguna dan melakukan pergerakan. Untuk dapat mengamati selisih waktu tersebut dengan akurat, maka pengendalian *quadcopter* dilakukan dengan menggunakan alat perekam seperti kamera ataupun webcam. Setelah melakukan perekaman, hasil file video yang dipeloreh akan dianalisis dengan menggunakan perangkat lunak untuk penyunting video adobe premier CC. File video yang diinput kedalam adobe premier dianalisis dengan menghitung selisih waktu frame antara gerakan yang diberikan oleh pengguna dengan gerakan yang dilakukan oleh *quadcopter*.

6.3.3 Prosedur Pengujian

Agar pengujian dapat dilakukan dengan optimal maka pengujian ini dilakukan dengan prosedure berikut:

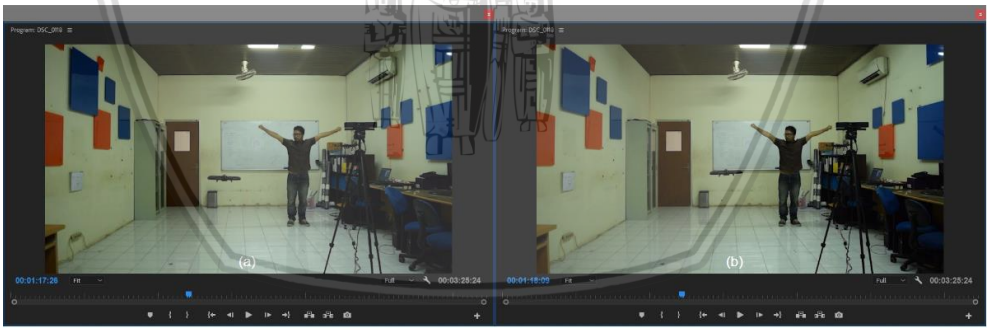
1. Menghubungkan sensor kinect dengan komputer menggunakan kabel USB versi 3.0.
2. Memasang *batteray* lithium pada *quadcopter* dan menunggu hingga *quadcopter* melakukan kablirasi dengan indikasi suare beep sebanyak empat kali. Setelah *quadcopter* malakukan kablirasi maka *quadcopter* siap untuk digunakan.
3. Melakukan hubungan koneksi jaringan dari PC ke *quadcopter* melalui koneksi wi-fi.
4. Membuka command promt node.js pada komputer dan mengetik syntax "node kendaliNavigasiDrone.js"
5. Pengguna mempersiapkan posisi tubuh di depan sensor kinect.
6. Saat pengguna melakukan pergerakan tubuh untuk memberikan instruksi pergerakan pada *quadcopter*, rekam keseluruhan proses menggunakan alat perekam video seperti kamera DSLR atau *smartphone*.
7. Ulangi pengujian sebanyak lima kali.

8. File video yang didapat daru hasil pengujian diimport ke software video editing yaitu adobe premier untuk melakukan pengamatan video per frame dengan agar mendapatkan nilai hasil delay dengan akurat. Nilai delay dihitung dengan perhitungan 1000 dibagi dengan besar frame per detik pada hasil rekaman video, lalu dikalikan dengan total selisih frame antara pengguna melakukan instruksi pergerakan secara sempurna hingga *quadcopter* menerima instruksi dari pengguna tersebut dan mulai melakukan pergerakan.
9. Setelah mendapatkan data *delay* pada sistem yang menggunakan metode *finite state machine*, maka data *delay* tersebut dibandingkan dengan data *delay* pada sistem yang tidak menggunakan metode finite state machine dengan tujuan untuk mengetahui total selisih delay pada kedua sistem tersebut.

6.3.4 Hasil Pengujian

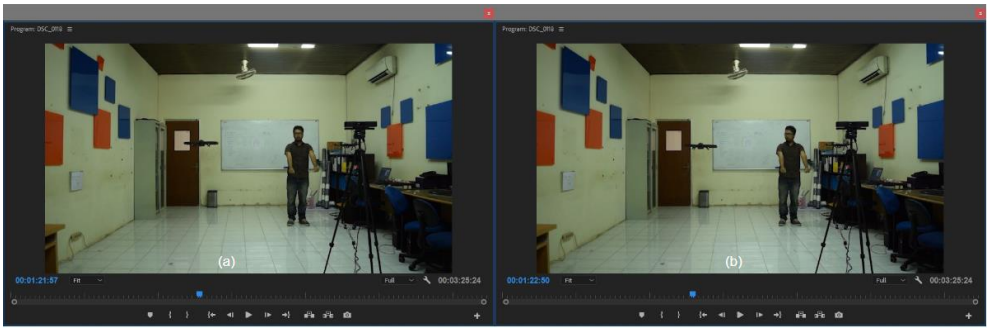
Setelah pengujian delay sistem dilakukan pada sistem yang menggunakan metode *finite state machine*, maka dipeloreh nilai hasil delay pada masing-masing pergerakan *quadcopter* sebagai berikut:

1. Pergerakan gaz.
   Pengujian *gaz* yang dilakukan memiliki dua jenis pergerakan, yaitu *gaz* ke atas dan *gaz* ke bawah. Pengujian pergerakan gaz ke atas dapat dilihat pada gambar 6.17. Pada gambar 7.17 bagian (a) tersebut diperlihatkan bahwa pada detik ke-17.26 pengguna melakukan posisi untuk melakukan gerakan gaz ke atas. Kemudian pada gambar 7.17 bagian (b), diperlihatkan bahwa *quadcopter* mulai berpindah posisi ke arah atas pada detik ke-18.09.



Gambar 6.17 Pengujian *delay* gerakan *gaz* ke depan

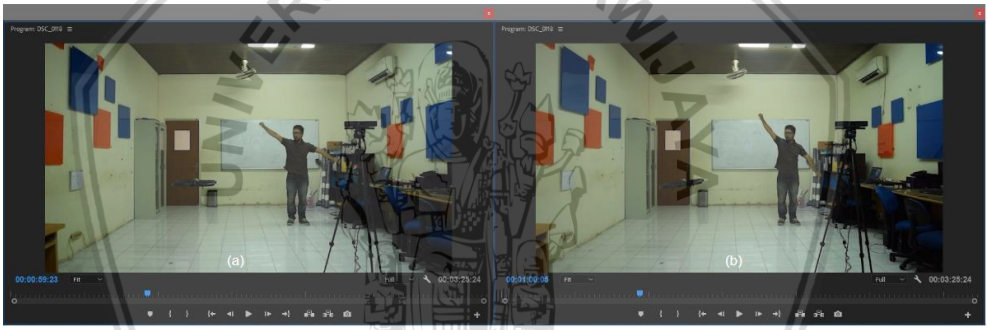
Lalu pengujian pergerakan gaz ke bawah dapat dilihat pada gambar 6.18. Pada gambar 6.18 bagian (a) tersebut diperlihatkan bahwa pada detik ke-21.57 pengguna melakukan posisi untuk melakukan gerakan gaz ke atas. Kemudian pada gambar 6.18 bagian (b), diperlihatkan bahwa *quadcopter* mulai berpindah posisi ke arah bawah pada detik ke-22.50 .



Gambar 6.18 Pengujian *delay* gerakan *gaz* ke bawah

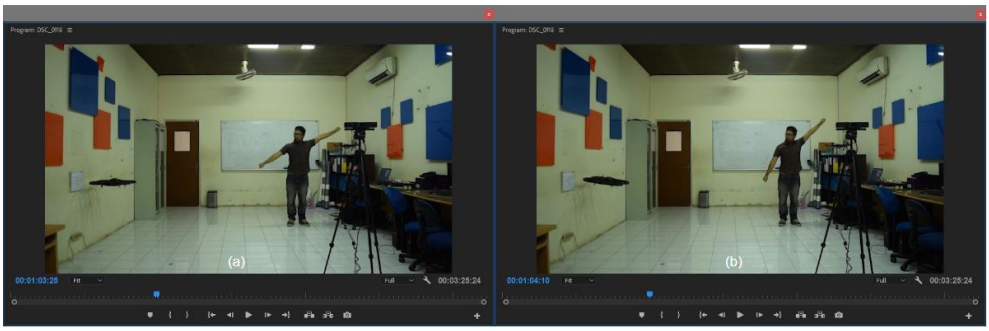
2. Pergerakan roll.

   Pengujian roll yang dilakukan memiliki dua jenis pergerakan, yaitu roll ke kanan dan roll ke bawah. Pengujian roll ke kanan dapat dilihat pada gambar 6.19. Pada gambar 6.19 bagian (a) tersebut diperlihatkan bahwa pada detik ke-59.23 pengguna melakukan posisi untuk melakukan gerakan roll ke kekanan. Kemudian pada gambar 6.19 bagian (b), diperlihatkan bahwa *quadcopter* mulai berpindah posisi ke arah kanan pada detik ke-00.05



Gambar 6.19 Pengujian *delay* gerakan *roll* ke kanan

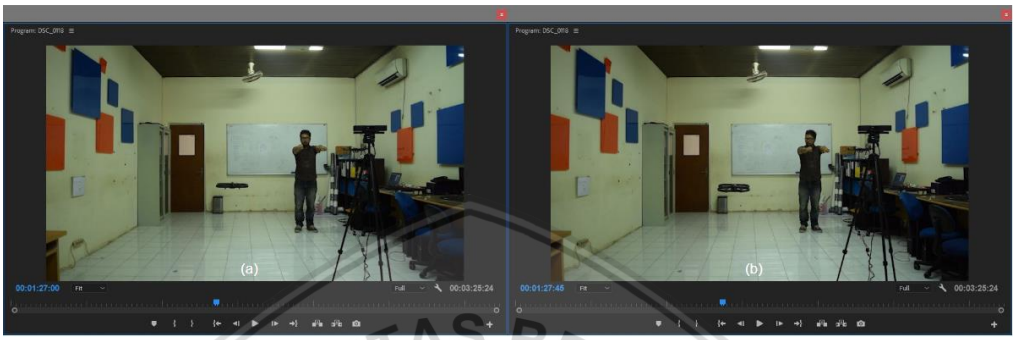
Lalu pengujian pergerakan roll ke kiri dapat dilihat pada gambar 6.20. Pada gambar 6.20 bagian (a) tersebut diperlihatkan bahwa pada detik ke-03.25 pengguna melakukan posisi untuk melakukan gerakan roll ke kiri. Kemudian pada gambar 6.20 bagian (b), diperlihatkan bahwa *quadcopter* mulai berpindah posisi ke arah kiri pada detik ke-04.10.



Gambar 6.20 Pengujian *delay* gerakan *roll* ke kiri

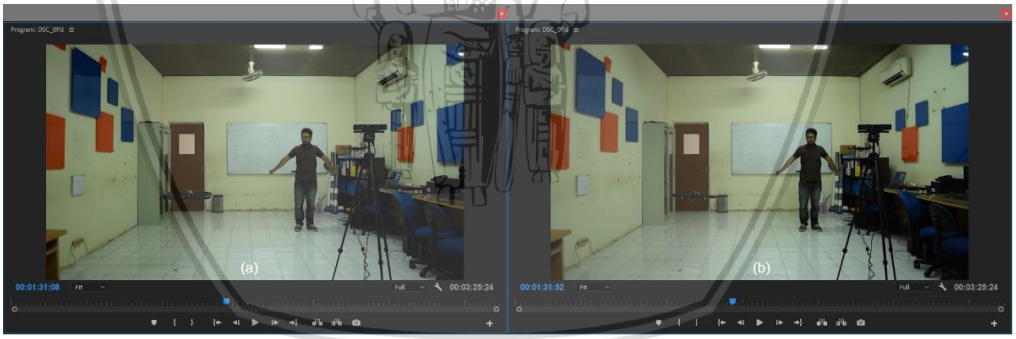
3. Pergerakan pitch.

   Pengujian *pitch* yang dilakukan memiliki dua jenis pergerakan, yaitu *pitch* ke depan dan *pitch* ke belakang. Pengujian *pitch* ke depan dapat dilihat pada gambar 6.21. Pada gambar 6.21 bagian (a) tersebut diperlihatkan bahwa pada detik ke-27.00 pengguna melakukan posisi untuk melakukan gerakan pitch ke depan. Kemudian pada gambar 6.21 bagian (b), diperlihatkan bahwa *quadcopter* mulai berpindah posisi ke arah depan pada detik ke-27.45



Gambar 6.21 Pengujian *delay* gerakan *pitch* ke depan

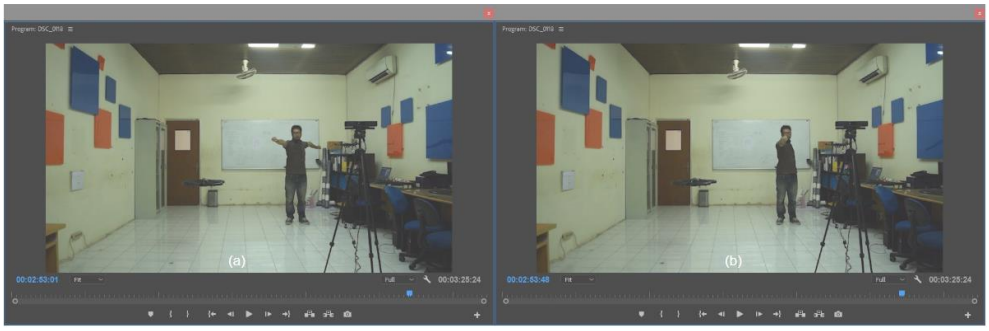
Lalu pengujian pergerakan pitch ke belakang dapat dilihat pada gambar 6.22. Pada gambar 6.22 bagian (a) tersebut diperlihatkan bahwa pada detik ke-31.08 pengguna melakukan posisi untuk melakukan gerakan *pitch* ke belakang. Kemudian pada gambar 6.22 bagian (b), diperlihatkan bahwa *quadcopter* mulai berpindah posisi ke arah belakang pada detik ke-31.52.



Gambar 6.22 Pengujian *delay* gerakan *pitch* ke belakang

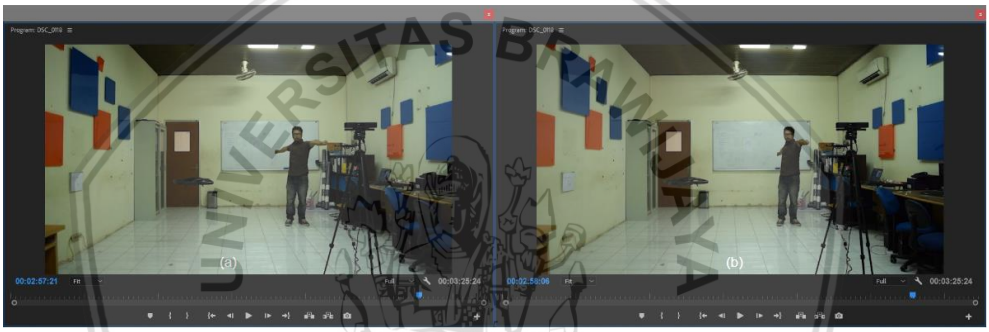
4. Pergerakan yaw.

   Pengujian *yaw* yang dilakukan memiliki dua jenis pergerakan, yaitu *yaw* ke arah kanan dan *yaw* ke arah kiri. Pengujian *yaw* ke arah kanan dapat dilihat pada gambar 6.23. Pada gambar 6.23 bagian (a) tersebut diperlihatkan bahwa pada detik ke-23.01 pengguna melakukan posisi untuk melakukan gerakan yaw ke arah kanan. Kemudian pada gambar 6.23 bagian (b), diperlihatkan bahwa *quadcopter* mulai berpindah posisi ke arah searah jarum jam pada detik ke-53.48.



Gambar 6.23 Pengujian *delay* gerakan *yaw* ke kanan

Lalu pengujian pergerakan yaw ke arah kiri dapat dilihat pada gambar 6.24. Pada gambar 6.24 bagian (a) tersebut diperlihatkan bahwa pada detik ke-57.21 pengguna melakukan posisi untuk melakukan gerakan yaw arah ke arah kiri. Kemudian pada gambar 6.24 bagian (b), diperlihatkan bahwa *quadcopter* mulai berpindah posisi ke arah berlawanan jarum jam pada detik ke-58.06.



Gambar 6.24 Pengujian *delay* gerakan *yaw* ke kiri

6.3.5 Analisis Pengujian

Setelah pengujian delay waktu sistem dilakukan , maka didapatkan hasil delay pada sistem pengendalian quadcopter dengan menggunakan sensor *kinect* yang menggunakan metode *finite state machine* yang ditampilkan pada tabel 6.7. Pengujian total *delay* diatas dilakukan menggunakan kamera yang diatur untuk merekam video. Kamera tersebut diatur dengan pengambilan *frame* per detik sebesar 60. Sehingga untuk dapat mengetahui total delay yang terjadi secara akurat, maka digunakan rumus pada persamaan 6.2.

$$\text{Nilai } delay \text{ sistem} = \frac{1000}{\text{Total } frame \text{ per detik pada kamera}} X \text{ Total frame} \quad \textbf{(6.2)}$$

Total frame yang pakai dalam perhitungan *delay* adalah *frame* antara pengguna melakukan instruksi pergerakan secara sempurna hingga *quadcopter* menerima instruksi dari pengguna tersebut dan mulai melakukan pergerakan. Dengan menggunakan rumus tersebut, *delay* waktu sistem dapat diketahui dalam satuan waktu *millisecond* yang kemudian diubah kedalam satuan waktu *second*. Untuk perhitungan nilai delay total dipeloreh dari nilai rata-rata pada keseluruhan percobaan yang telah dilakukan. Sehingga nilai rata-rata waktu delay sistem pada keseluruhan pergerakan dipeloreh sebesar 0.74 detik.

Tabel 6.7 Data *delay* pergerakan pada sistem dengan metode FSM

Pergerakan tubuh	Nilai *delay* pengujian ke- (detik)					Rata-rata (Detik)
	1	2	3	4	5	
Pitch maju	0.63	0.72	0.81	0.63	0.79	0.72
Pitch mundur	0.76	0.78	0.76	0.88	0.62	0.76
Roll kanan	0.67	0.86	0.62	0.75	0.71	0.72
Roll kiri	0.78	0.73	0.84	0.61	0.69	0.73
Gaz atas	0.79	0.63	0.77	0.86	0.78	0.77
Gaz bawah	0.68	0.74	0.75	0.63	0.87	0.73
Yaw kanan	0.76	0.82	0.71	0.66	0.86	0.76
Yaw kiri	0.64	0.71	0.69	0.76	0.83	0.73
Total						0.74

Sedangkan data *delay* pada penelitian yang telah dilakukan oleh Hadi et al. (2017) yaitu pengendalian *quadcopter* dengan menggunakan sensor *kinect* yang dilakukan tanpa menggunakan metode *finite state machine* dapat dilihat pada tabel 6.8. Nilai rata-rata waktu *delay* sistem pada keseluruhan pergerakan dipeloreh sebesar 0.17 detik.

Tabel 6.8 Data *delay* pergerakan pada sistem tanpa metode FSM

Pergerakan tubuh	Nilai *delay* pengujian ke- (detik)					Rata-rata (Detik)
	1	2	3	4	5	
Pitch maju	0.17	0.20	0.20	0.13	0.17	0.17
Pitch mundur	0.17	0.17	0.17	0.20	0.17	0.18
Roll kanan	0.17	0.17	0.13	0.17	0.20	0.17
Roll kiri	0.10	0.17	0.17	0.20	0.17	0.16
Gaz atas	0.13	0.20	0.17	0.17	0.20	0.17
Gaz bawah	0.13	0.17	0.17	0.20	0.17	0.17
Yaw kanan	0.13	0.17	0.20	0.17	0.17	0.17
Yaw kiri	0.17	0.13	0.17	0.17	0.13	0.15
Total						0.17

Setelah diketahui total *delay* pada sistem yang menggunakan metode *finite state machine* dan sistem yang tidak menggunakan metode *finite state machine*, maka dilakukan perbandingan rata-rata *delay* pada setiap gerakan untuk mengetahui total selisih *delay* pada kedua sistem tersebut yang ditampilkan pada tabel 6.9. Total selisih

delay antara sistem kendali *quadcopter* dengan menggunakan metode *finite state machine* dengan sistem kendali quadcopter tanpa menggunakan metode *finite state* machine yang dilakukan dengan menggunakan sensor *kinect* bernilai sebesar 0.57 detik.

Tabel 6.9 Data perbandingan delay antara sistem dengan FSM dan tanpa FSM

Pergerakan tubuh	Rata-rata *delay* dengan FSM (Detik)	Rata-rata *delay* tanpa FSM (Detik)	Total selisih *delay* (Detik)
Pitch maju	0.72	0.17	0.55
Pitch mundur	0.76	0.18	0.58
Roll kanan	0.72	0.17	0.55
Roll kiri	0.73	0.16	0.57
Gaz atas	0.77	0.17	0.60
Gaz bawah	0.73	0.17	0.56
Yaw kanan	0.76	0.17	0.59
Yaw kiri	0.73	0.15	0.58
Total			0.57

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan implementasi dan pengujian yang telah dilakukan pada penelitian ini, maka dapat disimpulkan sebagai berikut:

1. Berdasarkan pengujian metode *finite state machine* yang telah dilakukan pada *quadcopter*, didapatkan hasil pengendalian pergerakan *quadcopter* yang dimana tidak terjadi tabrakan terhadap objek yang berada pada sekitar *quadcopter* yang menyebabkan *quadcopter* tersebut menjadi nonaktif sehingga tidak dapat dikendalikan lebih lanjut ketika sistem menggunakan metode *finite state machine*. Kemudian pada pengujian gerakan navigasi yang dilakukan sebanyak empat belas kali dengan menggunakan metode *finite state machine*, didapatkan hasil yaitu kondisi *quadcopter* cenderung memasuki keadaan *hover* yang disebabkan oleh keterbatasan gerakan navigasi pada *quadcopter* yang dirancang pada metode *finite state machine* tersebut. Sehingga dapat disimpulkan bahwa penerapan metode *finite state machine* telah meningkatkan keamanan pada sistem pengendalian dan navigasi pada *quadcopter* yang dikendalikan oleh pengguna menggunakan *microsoft kinect*.
2. Berdasarkan pengujian ketepatan gerakan yang telah dilakukan, didapatkan hasil nilai persentase ketepatan gerakan sebesar 100 %. Sehingga dapat disimpulkan bawah sistem telah mampu berjalan sesuai yang diharapkan.
3. Berdasarkan pengujian *delay* yang dilakukan pada sistem kendali *quadcopter* dengan menggunakan metode *finite state machine* dan sistem kendali *quadcopter* tanpa menggunakan metode *finite state machine* yang dilakukan dengan menggunakan sensor *kinect*, maka dapat disimpulkan bahwa sistem kendali *quadcopter* yang menggunakan metode finite state machine memiliki *delay* yang lebih tinggi daripada sistem kendali *quadcopter* yang tidak menggunakan metode *finite state machine* dengan total selisih *delay* antara kedua sistem tersebut sebesar 0.57 detik yang dimana pada kedua sistem tersebut memiliki performa yang baik.

7.2 Saran

Terdapat beberapa saran agar sistem ini dapat dikembangkan lebih lanjut yakni sebagai berikut:

1. Untuk penelitian selanjutnya, dilakukan pengembangan terhadap rancangan metode *finite state machine* yang diterapkan pada sistem kendali dan navigasi *quadcopter* dengan *kinect* yang dimana pengguna dapat melakukan gerakan navigasi dengan aman tanpa mengurangi keleluasaan pengguna tersebut dalam melakukan kendali dan navigasi *quadcopter* menggunakan *kinect*.
2. Pada penelitian ini, jumlah pengguna yang dapat menggunakan sensor *kinect* untuk dapat mengendalikan *quadcopter* hanya berjumlah satu pengguna. Saran untuk penelitian selanjutnya, terdapat dua atau lebih pengguna yang melakukan

pergerakan tubuh didepan sensor kinect dalam rangka mengendalikan *quadcopter*.

3. Data pergerakan tubuh yang digunakan untuk mengendalikan *quadcopter* hanya menggunakan perubahan nilai *joint* tubuh bagian atas. Saran untuk penelitian selanjutnya, data *joint* tubuh bagian bawah dapat digunakan sebagai input dalam rangka pengendalian *quadcopter* dengan menggunakan sensor *kinect*.
4. Pada penelitian ini, metode *automata* yang digunakan dalam kendali dan navigasi pada *quadcopter* adalah *finite state machin*e. Untuk penelitian selanjutnya, penelitian dapat menggunakan metode *automata* yang lainnya seperti *pushdown automata*, *linear-bounded automata*, ataupun *turing machine*.

DAFTAR PUSTAKA

Aron, J., 2011. *microsoft explain the tech behind Kinect*. [Online] Available at: http://www.newscientst.com/blogs/onpercent/2011/03/micro [DIakses 15 Agustus 2017].

Blake, J., 2013. *Natural User Interface in .NET.* New York: Manning Publications.

Catuhe, D., 2012. *Programming with the Kinect for Windows Software Development Kit.* United State of America: Microsoft.

Computer Sciene Stanford Education, 2004. *Automata Theory*. [Online] Available at: https://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/automata-theory/basics.html. [Diakses 28 April 2018].

Embedded Micro, 2017. *Finite State Machine*. [Online] Available at: https://embeddedmicro.com/tutorials/mojo/finite-state-machines. [DIakses 13 Agustus 2017].

Hadi, S W., Setyawan, G. E. & Maulana, R., 2017. Sistem Kendali Navigasi *AR.Drone Quadcopter* Dengan Prinsip *Natural User Interface* menggunakan *Microsoft Kinect*. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer (JPTIIK)*, pp. 380-386.

Kugler, M. & Holzapfel N., 2016. *Designing a Safe and Robust Automatic Take-off Maneuver for a Fixed-Wing UAV*. Thailand, IEEE.

Kusuma, W., AK, R. E. & Iskandar, E., 2012. Perancangan dan Implementasi Kontrol Fuzzy-PID pada Pengendalian Auto Take-Off Quadcopter UAV. *JURNAL TEKNIK POMITS,* 1(1), pp. 1-6.

Linux Information Project, 2005. AT Command Set Definition. [Online] Available at: www.linfo.org/at_command_set.html [Diakses 29 Mei 2017].

Mashood, A., Noura, H., Jawhar, I. & Mohamed, N., 2015. *A Gesture Based Kinect for Quadrotor Control*. Abu Dhabi, IEEE.

Mayer, J., Kuderer, M., Muller, J. & Burgard, W., 2014. *Online Marker labeling for fully Automatic Skeleton Tracking in Optical Motion Capture*. Hongkong, IEEE.

Microsoft, 2017. *Microsoft Developer.* [Online] Available at: msdn.microsoft.com/en-us/library/ii131025.aspx. [Diakses 14 Agustus 2017].

Mihn, N., Jongwoo, Lee., Yonghwan, O. 2017. *A finite state machine and walking control algorithm of the biped robot with flat foot*. Japan, IEEE.

Rosado, J., Silva. F. & Santos, V., 2013. Using Kinect for Robot Gesture Imitaion. Procedia Technologi, 17(2014), pp. 423-430.

Sanna, A., Lamberti, F., Paravati, G. & Manuri, F., 2013. A Kinect-based natural interface for quadrotor control. *Elsevier,* pp. 179-186.

Setyawan, G. E., Setiawan, E. & Kurniawan, W., 2015. Sistem Kendali Ketinggian Quadcopter Menggunakan PID. *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK),* 2(2), pp. 125-131.

Vitor, D., Ana, P., Almeida., Ciro, T & Flavio, D., 2014. *A Parallel hierarchial finite state machine approach to UAV control for search and rescue task*. Austria, IEEE.

Zeng, W., 2012. Microsoft Kinect Sensor and its effect. *IEEE Multimedia*, 19(2), pp. 4-10.