TUGAS AKHIR – TE141599

IMPLEMENTASI INVERSE KINEMATICS PADA PERGERAKAN ROBOT QUADRUPED

Wahyu Tri Wibowo
NRP 2213100107

Dosen Pembimbing
Dr. Ir. Djoko Purwanto, M.Eng.
Fajar Budiman, ST. M.Sc.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2017

Institut
Teknologi
Sepuluh Nopember

FINAL PROJECT – TE141599

IMPLEMENTATION OF INVERSE KINEMATICS ON QUADRUPED ROBOT MOVEMENT

Wahyu Tri Wibowo
NRP 2213100107

Advisor
Dr. Ir. Djoko Purwanto, M.Eng.
Fajar Budiman, ST. M.Sc.

DEPARTMENT OF ELECTRICAL ENGINEERING
Faculty of Electrical Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2017

PERNYATAAN KEASLIAN
TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan **"Implementasi *Inverse Kinematics* Pada Pergerakan Robot *Quadruped*"** adalah benar benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka.

Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, 24 Juli 2017

Wahyu Tri Wibowo
NRP. 2213100107

IMPLEMENTASI INVERSE KINEMATICS PADA PERGERAKAN ROBOT QUADRUPED

TUGAS AKHIR

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

Menyetujui

Dosen Pembimbing I	Dosen Pembimbing II
Dr. Ir. Djoko Purwanto, M.Eng.	Fajar Budiman, ST., M.Sc.
NIP. 196512111990021002	NIP. 198607072014041001

KEMENTERIAN RISET, TEKNOLOGI DAN PENDIDIKAN TINGGI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
FAKULTAS TEKNOLOGI ELEKTRO
DEPARTEMEN

SURABAYA
JULI, 2017

IMPLEMENTASI INVERSE KINEMATICS PADA PERGERAKAN ROBOT QUADRUPED

Nama : Wahyu Tri Wibowo
Pembimbing I : Dr. Ir. Djoko Purwanto, M.Eng.
Pembimbing II : Fajar Budiman, ST. M.Sc.

ABSTRAK

Robot *quadruped* merupakan robot berkaki empat yang mampu berjalan dengan mengatur pergerakan pada masing-masing kakinya. Pada tugas akhir ini setiap kaki robot terdapat tiga sendi (3 DoF) yang memungkinkan *end-effector* mampu bergerak pada sumbu x, y, z. Permasalahan yang sering dihadapi dalam perancangan robot berkaki adalah perumusan pergerakan kaki (kinematika).

Pada tugas akhir ini dibahas mengenai perancangan *inverse kinematics* pada robot berkaki empat (*quadruped*) mulai dari perancangan *hardware* hingga perancangan gerak robot dengan menggunakan servo dynamixel RX-28. Untuk mewujudkan pergerakan kaki yang lebih halus maka digunakan trayektori ujung kaki berbasis kartesian. Pola langkah (*gait*) yang diterapkan pada robot adalah *dualpod.* Manuver gerakan robot dirumuskan dengan menggunakan transformasi geometri *translasi* dan *rotasi*.

Dengan menggunakan rumus *inverse kinematics* dan transformasi geometri (*translasi* dan *rotasi*) yang disertai dengan trayektori berbasis kecepatan mampu mewujudkan pergerakan pada robot *quadruped*. Kecepatan maksimal robot dalam berjalan lurus yang mampu ditempuh robot yaitu 0,33 m/s. Tingkat rata-rata kesalahan posisi pada gerak satu kaki robot adalah 11%. Tingkat kesalahan rata-rata jarak langkah robot kurang dari 0,9%. Tingkat kesalahan rata-rata sudut belok pada gerak jalan lurus robot sejauh 1,5 m adalah sebesar 11,1° belok ke arah kiri. Tingkat kesalahan rata-rata sudut dari pergerakan ditempat adalah 0,62°.

Kata kunci: robot *quadruped*, *inverse kinematics,* kartesian trayektori, *dualpod gait*

Halaman ini sengaja dikosongkan

IMPLEMENTATION OF INVERSE KINEMATICS ON QUADRUPED ROBOT MOVEMENT

Name	**: Wahyu Tri Wibowo**
Advisor	**: Dr. Ir. Djoko Purwanto, M.Eng.**
Co-Advisor	**: Fajar Budiman, ST. M.Sc.**

ABSTRACT

Quadruped robot is a four legged robot that is capable of walking by arranging movement on each of its legs. In this final project each robot leg has three joints (3 DoF) allowing the end-effector to move in the direction of x, y, z axes. The problem often faced in the design of a legged robot is the formulation of leg movement (kinematics).

This final project discuss the design of inverse kinematics on quadruped robots from the hardware design to the design of robot motion using servo dynamixel RX-28. To realize the more subtle movement of the leg it used a leg-end based on catesian trajectory planning. The gait algorithm applied to the robot is dualpod. Maneuvering robot movements are formulated using translational and rotational geometry transformations.

Using the inverse kinematics and geometric transformation (translational and rotational) formulas accompanied by a speed-based trajectory has been capable of realizing the movement of quadruped robot. The maximum speed of robot in a straight walk is 0.33 m / s. The average rate of position error on the motion of a robot leg is 11%. The average error rate of the robot step distance is less than 0.9%. The average turn angle error rate on the robot's straight motion in 1.5 m is 11.1 ° turning to the left. The average angular error rate of the motion in place is 0.62 °.

Keywords : quadruped robot, inverse kinematics, cartesian trajectory, dualpod gait

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji syukur kepada Tuhan Yang Maha Esa atas kasih dan rahmat-Nya penulis dapat menyelesaikan Tugas Akhir ini dengan judul :

Implementasi *Inverse Kinematics* pada Pergerakan Robot *Quadruped*

Tugas Akhir ini merupakan persyaratan dalam menyelesaikan pendidikan program Strata-Satu di Departemen Teknik Elektro, Fakultas Teknologi Elektro, Institut Teknologi Sepuluh Nopember, Surabaya.

Tugas Akhir ini dibuat berdasarkan teori-teori yang didapat selama mengikuti perkuliahan, berbagai literatur penunjang dan pengarahan dosen pembimbing dari awal hingga akhir pengerjaan Tugas Akhir ini.

Pada kesempatan ini, penulis ingin berterima kasih kepada pihak-pihak yang membantu pembuatan tugas akhir ini, khususnya kepada:

1. Bapak, Ibu, Kakak serta seluruh keluarga yang memberikan dukungan baik moril maupun materiil.
2. Dr. Ir. Djoko Purwanto, M.Eng. selaku dosen pembimbing 1 atas bimbingan dan arahan selama penulis mengerjakan tugas akhir ini.
3. Fajar Budiman, ST. M.Sc. selaku dosen pembimbing 2 atas bimbingan dan arahan selama penulis mengerjakan tugas akhir ini.
4. Ir. Tasripan, M.T. selaku Koordinator Bidang Studi Elektronika.
5. Dr.Eng. Ardyono Priyadi, S.T., M.Eng. selaku Kepala Departemen Teknik Elektro ITS Surabaya.
6. Seluruh dosen bidang studi Elektronika.
7. Teman-teman tim robot ITS dan tim KRPAI berkaki ITS ABINARA-1 yang telah membantu proses pengerjaan tugas akhir ini.

Penulis sadar bahwa Tugas Akhir ini belum sempurna dan masih banyak hal yang dapat diperbaiki. Saran, kritik dan masukan dari semua pihak sangat membantu penulis untuk pengembangan lebih lanjut.

Terakhir, penulis berharap Tugas Akhir ini dapat memberikan manfaat bagi banyak pihak. Penulis juga berharap Tugas Akhir ini dapat membantu pengembangan robot *quadruped*.

Surabaya, 24 Juli 2017

Penulis

Halaman ini sengaja dikosongkan

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Halaman ini sengaja dikosongkan

BAB I
PENDAHULUAN

1.1. Latar Belakang

Robot *quadruped* merupakan sebuah robot berkaki dengan kaki berjumlah empat buah yang mampu berjalan dengan mengatur pergerakan pada masing-masing kakinya. Pada tugas akhir ini setiap kaki pada robot *quadruped* dirancang dengan terdapat tiga sendi (3 DoF) yang memungkinkan *end-effector* kaki robot mampu bergerak pada sumbu x, y, dan z. Permasalahan yang sering dihadapi dalam perancangan robot berkaki adalah perumusan pergerakan kaki (kinematika) yang meliputi trayektori ujung kaki, pola langkah (*gait*) dan manuver gerakan sehingga memungkinkan robot untuk dapat bergerak berjalan maupun bergerak di tempat. Perancangan yang tepat akan menghasilkan pergerakan robot yang rapi, cepat dan terarah.

Robot *quadruped* saat ini banyak dikembangkan dalam berbagai kegiatan diantaranya yaitu Kontes Robot Pemadam Api Indonesia (KRPAI) yang merupakan *event* perlombaan robot pemadam api yang tiap tahunnya dilaksanakan di Indonesia dan *Trinity College Fire-Fighting Home Robot Contest* yang merupakan perlombaan robot pemadam api tingkat internasional. Kedepannya robot pemadam api dipersiapkan untuk membantu manusia dalam memadamkan api pada musibah kebakaran.

Sistem kinematika digunakan untuk menghitung hubungan antara posisi *joint* dan posisi lengan robot dalam koordinat *cartesian*. Kinematik pada robot berkaki dibagi menjadi dua yaitu *forward kinematics* dan *inverse kinematics*. Perencanaan gerak (trayektori) digunakan untuk merencanakan pergerakan tiap kaki robot. Trayektori dibagi menjadi dua metode yaitu : perencanaan gerak berbasis *joint* (*joint trajectory planning*) dan perencanaan gerak berbasis *cartesian* (*cartesian trajectory planning*). Pola langkah digunakan untuk menentukan *fase support* dan *fase transfer* pada tiap-tiap kaki. Sedangkan manuver gerakan robot digunakan untuk merumuskan pergerakan robot.

Robot yang akan dirancang pada tugas akhir ini disesuaikan dengan peraturan pada Kontes Robot Pemadam Api Indonesia (KRPAI) divisi berkaki dan disesuaikan untuk menunjang misi robot untuk memadamkan api. Pada divisi berkaki dimensi maksimal robot adalah 31 x 31 x 27 (panjang x lebar x tinggi). Pada tugas akhir ini akan dilakukan pengembangan *inverse kinematics*, perencanaan gerak (trayektori), pola langkah dan manuver gerakan pada robot *quadruped*. Robot *quadruped* berjalan dengan sistem *dualpod gait* yaitu dua kaki menyangga dan dua kaki bergerak. Sendi pada setiap kaki robot disusun dari servo. Motor servo adalah motor DC yang dilengkapi kontrol posisi sudut putaran dengan masukan setpoint sudut. Diharapkan robot *quadruped* akan memiliki pola pergerakan kaki yang stabil, cepat, dan efektif. Penelitian ini dilakukan menggunakan robot *quadruped* hasil dari perancangan sendiri yang terdiri dari 3 DoF pada tiap kaki dengan menggunakan servo tipe dynamixel RX-28 dan mikrokontroller STM32F4 discovery.

1.2. Perumusan Masalah

Permasalahan yang dibahas dalam tugas akhir ini adalah:

1. Bagaimana merumuskan *inverse kinematic* pada robot *quadruped*?
2. Bagaimana merumuskan perencanaan gerak (trayektori) pada robot *quadruped*?
3. Bagaimana merumuskan pola langkah pada robot *quadruped*?
4. Bagaimana merumuskan manuver pergerakan pada robot *quadruped*?

1.3. Batasan Masalah

Batasan masalah dalam tugas akhir ini adalah

1. Kinematika ini hanya dapat diterapkan pada robot berkaki empat (*quadruped*)
2. Robot ini diperuntukkan khusus untuk mengikuti Kontes Robot Pemadam Api (KRPAI) divisi berkaki.
3. Pada bagian atas robot ini terdapat part dan komponen untuk mendukung pergerakan robot dalam menelusuri ruang dan memadamkan api.

1.4. Tujuan

Tujuan dari pelaksanaan tugas akhir ini adalah

1. Didapatkannya rumusan pada sistem *inverse kinematics*, perencanaan gerak (*trajectory*), pola langkah dan manuver pergerakan pada robot *quadruped*.
2. Terciptanya pergerakan robot *quadruped* yang stabil, cepat, dan efektif.

1.5. Metodologi

Penulisan tugas akhir ini akan digunakan metodologi penelitian sebagai berikut:

1. Studi Literatur

   Tahap ini meliputi pengumpulan dasar teori yang dapat menjadi acuan tugas akhir. Dasar teori akan diambil dari buku – buku, jurnal yang telah dipublikasi, *proceeding* dan artikel dari internet.
2. Perancangan Hardware

   Pada tahap ini dilakukan proses desain dan pembuatan mekanik robot *quadruped* sesuai dengan karakteristik servo dynamixel RX-28. Selain itu diperhitungkan juga dimensi robot agar sesuai dengan peraturan pada KRPAI divisi berkaki tahun 2017. Proses pembuatan mekanik robot akan dilakukan dengan menggunakan mesin CNC dengan bahan alumunium pada bagian-bagian yang butuh penopang yang kuat. Serta digunakan acrilic pada body robot.
3. Perancangan Sistem Elektronis

   Pada tahap ini dilakukan proses desain dan pembuatan perangkat elektronik pada robot. Meliputi rangkaian regulator, shield STM32F4 discovery dan driver servo RS-485.
4. Perancangan Sistem Software

   Setelah proses membuatan body robot dan elektronis selesai maka tahap selanjutnya adalah proses pengaksesan serta setting ID pada masing-masing servo. Setelah itu dilakukan perhitungan *inverse kinematic*, dan perencanaan gerak yaitu *joint trajectory planning* dan *cartesian trajectory planning* pada robot.

5. Pengujian Sistem
   Tahapan pengujian sistem akan dilakukan dengan menggunakan bluetooth yang terkoneksi ke handphone android dan akan dianalisis pergerakannya apakah sudah sesuai dengan target atau belum, jika belum sesuai dengan targer maka akan dilakukan proses pembetulan pada perhitungan *inverse kinematic*.
6. Pembandingan Perencanaan Gerak
   Setelah didapatkan pergerakan yang sesuai pada masing-masing perencanaan gerak maka tahap selanjutnya adalah dilakukan pembandingan antara masing-masing perencanaan gerak. Hasil yang didapatkan akan digunakan sebagai kesimpulan dari kerja praktek ini.
7. Penulisan Laporan Tugas Akhir
   Setelah dilakukan segala percobaan dan pengambilan data, maka akan dilakukan penulisan laporan tugas akhir yang final.

1.6. Sistematika Penulisan

Dalam buku tugas akhir ini, pembahasan mengenai sistem yang dibuat terbagi menjadi lima bab dengan sistematika penulisan sebagai berikut:

- BAB I : PENDAHULUAN
  Bab ini meliputi penjelasan latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi, sistematika penulisan, dan relevansi.
- BAB II : TINJAUAN PUSTAKA DAN TEORI PENUNJANG
  Bab ini menjelaskan tentang teori penunjang dan *literature* yang dibutuhkan dalam pengerjakan tugas akhir ini. Dasar teori yang menunjang meliputi *inverse kinematics*, *cartesian trajectory planning*, transformasi geometri translasi dan rotasi, servo RX-28, RS-485, regulator, dan mikrokontroler STM32F4. Bagian ini memaparkan mengenai beberapa teori penunjang dan beberapa literatur yang berguna bagi pembuatan Tugas Akhir ini.
- BAB III : PERANCANGAN SISTEM
  Bab ini menjelasakan tentang perencanaan sistem baik perangkat keras (*hardware*) yang terdiri dari mekanik dan elektrik maupun perangkat lunak (*software*) untuk sistem perumusan kinematika untuk pergerakan robot.

- BAB IV : PENGUJIAN

  Pada bab ini akan menjelaskan hasil uji coba sistem beserta analisanya.
- BAB V : PENUTUP

  Bagian ini merupakan bagian akhir yang berisikan kesimpulan yang diperoleh dari pembuatan Tugas Akhir ini, serta saran-saran untuk pengembangan lebih lanjut.

1.7. Relevansi dan Manfaat

Hasil yang diharapkan dari tugas akhir ini diharapkan mampu berkontribusi terhadap perkembangan riset robot pemadam api berkaki di ITS dalam Kontes Robot Indonesia. Serta diharapkan kedepannya robot *quadruped* dapat diaplikasikan secara langsung untuk memudahkan dalam mengatasi bencana kebakaran.

Halaman ini sengaja dikosongkan

BAB II
TINJAUAN PUSTAKA DAN TEORI PENUNJANG

Tinjauan pustaka dan teori penunjang berisi tentang kumpulan informasi penunjang yang berhubungan dengan alat maupun sistem yang akan dibuat pada tugas akhir ini yang berasal dari berbagai sumber termasuk penelitian yang sudah pernah diimplementasikan oleh penulis-penulis sebelumnya. Kajian teori yang dibahas pada bagian ini terdiri dari kinematika robot, trayektori ujung kaki, pola langkah robot (*gait*), transformasi geometri, mikrokontroller STM32F4 discovery, dan dynamixel RX-28.

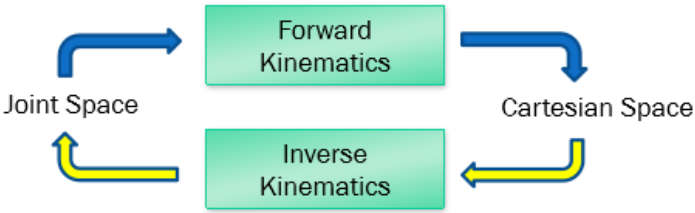
2.1. Kinematika Robot

Pengertian kinematika adalah studi tentang gerak tanpa memperhatikan faktor-faktor yang menyebabkan gerak tersebut. Sedangkan pengertian kinematika robot adalah kegiatan menghitung hubungan antara posisi *joint* dan posisi lengan robot dalam koordinat *cartesian*. [3]

Pada tugas akhir ini, kinematika merupakan sebuah cara untuk menggerakan tiap-tiap servo pada robot sehingga robot *quadruped* mampu bergerak sesuai yang diinginkan. Dalam kasus robot berkaki *quadruped* ini kinematika digunakan untuk menghitung hubungan antara posisi *joint* dan posisi lengan robot dalam koordinat *cartesian*.

Pada sebuah servo, data yang dapat diinputkan adalah berupa sudut derajat servo. Sedangkan yang kita inginkan adalah berupa inputan dalam sudut kartesian yaitu (x,y,z). Maka dari itu dibutuhkan sebuah kinematika robot yang dapat mengubah besaran *cartesian* menjadi besaran sudut pada tiap lengan robot.

Kinematika Pada lengan robot dibagi menjadi dua metode yaitu *forward kinematiks* dan *inverse kinematics*. Masing-masing metode mempunyai fungsi yang berkebalikan.

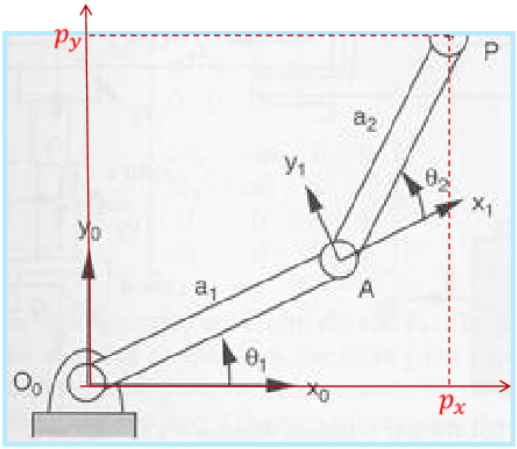


Gambar 2. 1 Analogi Kinematika *[3]*

Pada robot *quadruped* terdapat empat buah kaki yang masing-masing kaki terdiri dari tiga buah *joint* sehingga masing-masing kaki terdapat tiga tingkat kebebasan (*Degree of Freedom*/DoF). Pada kasus ini kinematika digunakan untuk mengendalikan posisi *end-effector* yaitu posisi kaki paling luar yang bersentuhan dengan lantai pada robot.

2.1.1. *Forward Kinematic*

Forward Kinematics merupakan sebuah metode untuk mengubah besaran sudut tiap *joint* menjadi besaran kartesian pada ujung *end-effector*. Pada metode Forward Kinematics, posisi dan orientasi ujung lengan (*end-effector*) robot (yang dikenal dengan istilah robot pose) dapat ditentukan berdasarkan posisi sudut-sudut *joint* dan struktur mekanik robot. [3]



Gambar 2. 2 Contoh Lengan Robot 2 Dof [3]

Dengan menggunakan metode *forward kinematic* maka bisa didapatkan posisi ujung lengan (*end-effector*) dengan menggunakan sudut-sudut tiap lengan. Pada contoh Gambar 2. 2 diatas maka dapat dihitung koordinat *end-effector* dengan persamaan :

$$Px = a1 . \cos\theta1 + a2. \cos(\theta1 + \theta2) \quad (2.\ 1)$$

$$Px = a1 . \sin\theta1 + a2. \sin(\theta1 + \theta2) \quad (2.\ 2)$$

Metode *forward kinematic* biasa dikenal dengan metode kinematik yang konvensional karena dengan *forward kinematic* untuk menentukan posisi koordinat ujung lengan (*end-effector*) harus dengan cara menghitung dengan menggunakan data tiap sudut pada lengan robot.

2.1.2. *Inverse Kinematics*

Inverse kinematics merupakan sebuah metode untuk mengubah besaran *cartesian* pada *end-effector* kaki robot menjadi besaran sudut pada tiap *joint*/servo dengan menggunakan rumus trigonometri. Pada metode *inverse kinematics*, sudut pada tiap-tiap *joint*/servo ditentukan oleh posisi ujung lengan kaki (*end-effector*) robot. [3]

Dengan menggunakan metode *inverse kinematics* maka bisa didapatkan sudut-sudut tiap lengan dengan menggunakan data posisi koordinat ujung lengan (*end-effector*) dengan menggunakan rumus trigonometri. Pada contoh Gambar 2. 2 diatas maka dapat dihitung sudut-sudut pada tiap lengan dengan persamaan :

$$Px^2 + Py^2 = a1^2 + a2^2 - 2.a1.a2\,\mathbf{cos}(\pi - \theta2)$$

$$Px^2 + Py^2 = a1^2 + a2^2 + 2.a1.a2\,\mathbf{cos}\,\theta2$$

$$\theta2 = \mathbf{cos^{-1}}\frac{Py^2 + Px^2 - a1^2 - a2^2}{2.a1.a2} \quad (2.\ 3)$$

$$\theta1 = \mathbf{tan^{-1}}\frac{Py}{Px} - \mathbf{tan^{-1}}\frac{a2.\mathbf{sin}\,\theta2}{a1 + a2.\mathbf{cos}\,\theta2} \quad (2.\ 4)$$

Inverse kinematics biasanya digunakan untuk memudahkan dalam menggerakkan lengan robot. Hal ini dikarenakan dengan metode *inverse kinematics* hanya perlu memasukkan nilai kartersian koordinat ujung lengan (*end-effector*), dan sudut-sudut tiap servo akan didapatkan dengan perhitungan *inverse kinematics*.

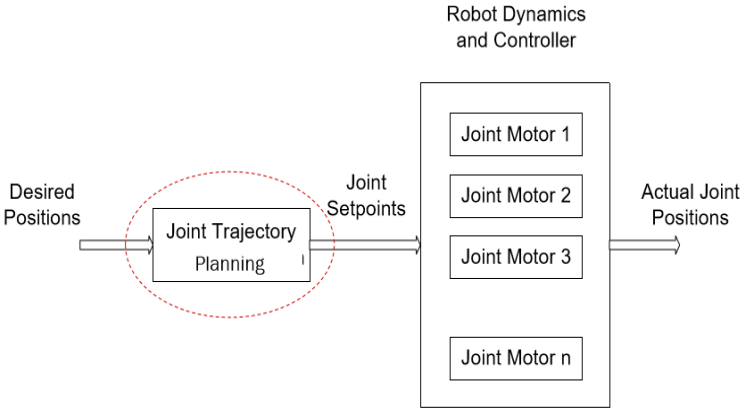
Metode *inverse kinematics* sangat dibutuhkan karena nilai masukan pada masing-masing servo adalah dalam derajat putar.

2.2. Trayektori Ujung Kaki

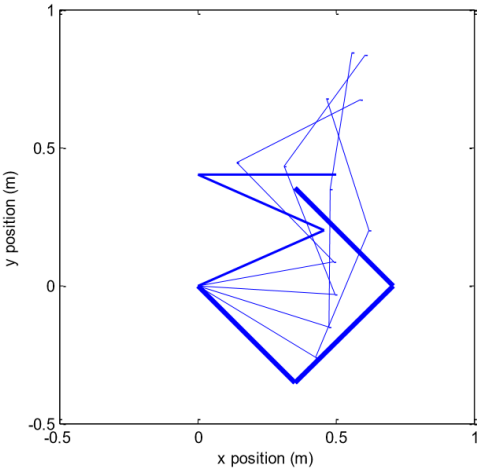
Postur tiap kaki pada robot *quadruped* dapat direpresentasikan dalam bentuk *joint space* maupun *cartesian space*. *Joint space* merupakan representasi dalam besaran sudut, sedangkan *cartesian space* merupakan representasi dalam besaran kartesian x,y,z. Trayektori ujung kaki merupakan sebuah perencanaan gerak pada ujung kaki robot. Algoritma trayektori menentukan arah gerak dari masing-masing kaki. Trayektori dibagi menjadi dua yaitu trayektori berbasis *joint* dan trayektori berbasis kartesian. Masing-masing trayektori mempunyai karakteristik yang saling berlawanan. Penggunaan trayektori menentukan performa dari masing-masing kaki robot sehingga berpengaruh terhadap performa jalan robot secara keseluruhan. [4]

2.2.1. Trayektori Ujung Kaki Berbasis *Joint*

Trayektori ujung kaki berbasis *joint* merupakan perencanaan gerak kaki yang menyebabkan pergerakannya sesuai dengan perubahan sudut pada masing masing servo.



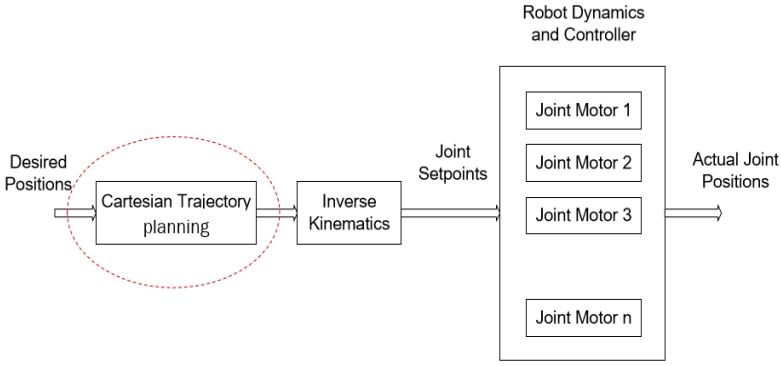
Gambar 2. 3 Alur *Joint Trajectory Planning [4]*



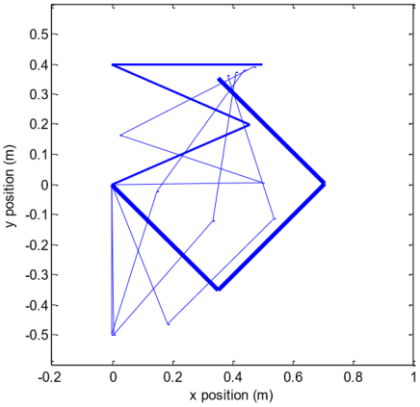
Gambar 2. 4 Ilustrasi *Joint Trajectory Planning* [4]

2.2.2. Trayektori Ujung Kaki Berbasis Kartesian

Trayektori ujung kaki berbasis kartesian merupakan perencanaan gerak kaki yang menyebabkan pergerakannya sesuai dengan perubahan dalam bidang kartesian.



Gambar 2. 5 Alur *Cartesian Trajectory Planning* [4]



Gambar 2. 6 Ilustrasi *Cartesian Trajectory Planning* [4]

2.3. Pola Langkah Robot (*Gait*)

Algoritma pola langkah (*gait*) diperlukan untuk mengatur waktu kapan sebuah kaki berada pada *fase support* dan kapan berada pada *fase transfer*. Prinsip dari perancangan algoritma *gait* adalah menentukan waktu yang tepat untuk masing-masing kaki berada dalam *fase support* maupun *fase transfer*. Ketepatan dalam menentukan pola langkah akan sangat mempengaruhi pergerakan robot. Fase pergerakan lengan robot dibagi menjadi dua yaitu :

- *Fase Support (stance)*
  Yaitu posisi dimana kaki robot diam untuk menyangga badan robot.
- *Fase Transfer (swing)*
  Yaitu posisi dimana kaki robot bergerak dari satu titik ke titik lainnya. [5] [6]

Terdapat beberapa algoritma *gait* yang dapat diterapkan pada robot berkaki sesuai dengan jumlah kakinya. Contohnya pada robot berkaki enam, algoritma *gait* yang dapat diterapkan antara lain yaitu pentapod, tetrapod, dan tripod. Sedangkan pada robot berkaki empat, algoritma *gait* yang dapat diterapkan antara lain yaitu tripod dan *dualpod*.

Yang dimaksud dengan algoritma *gait* adalah jumlah kaki yang digunakan sebagai penopang pada setiap pergerakan langkah kaki. Maka pada robot berkaki empat maka algoritma *gait* yang dapat diterapkan yaitu sebagai berikut :

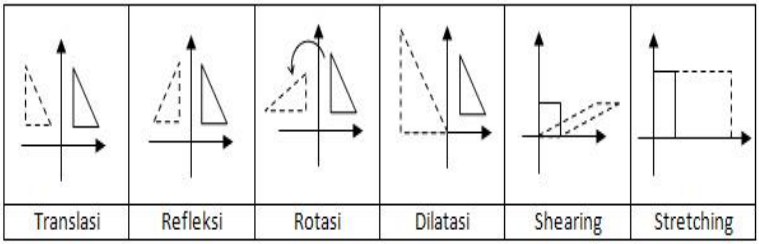
- *Tripod*
  Jumlah kaki penopang (*fase support*) 3 buah, jumlah kaki bergerak (*fase transfer*) 1 buah.
- *Dualpod*
  Jumlah kai penopang (*fase support*) 2 buah, jumlah kaki bergerak (*fase transfer*) 2 buah.

2.4. Transformasi Geometri

Transformasi Geometri berasal dari kata transformasi yang berarti perubahan dan geometri yang berarti ilmu yang membahas mengenai bangun. Sehingga, transformasi geometri adalah sebuah proses penentuan titik koordinat baru dari sebuah bangun pada sebuah bidang.

Transformasi geometri diperkenalkan oleh ilmuan bernama Felix Klein yang dijelaskan pada sebuah paper berjudul Erlangen Program. Beliau mengatakan bahwa geometri merupakan ilmu yang mempelajari mengenai bangun yang bisa ditransformasikan ke dalam bentuk yang berbeda dan sifat-sifat bangun tidak terpengaruh karena perubahan yang dilakukan.

Transformasi geometri dibagi menjadi empat jenis yaitu transformasi pergeseran (translasi), pencerminan (refleksi), perputaran (rotasi), dan perbesaran (dilatasi). Pada tugas akhir ini, transformasi geometri digunakan untuk menyusun kinematika manuver gerakan robot seperti jalan lurus dan belok. [7]



Gambar 2. 7 Jenis Transformasi Geometri [7]

2.4.1. Translasi (Pergeseran)

Transformasi translasi atau pergeseran merupakan sebuah proses yang dilakukan untuk menggeser sebuah bangun pada koordinat kartesian sebanyak nilai pergeseran yang diinginkan dalam sumbu x/y/z. Pada transformasi translasi, ukuran dan bentuk serta arah bangun tidak berubah, yang berubah hanya posisinya terhadap posisi awal.

Pada Gambar 2. 8 segitiga ABC berwarna hitam mempunyai koordinat A(3,9), B(3,3), C(6,3), apabila ditransformasikan translasi menjadi segitiga ABC warna oranye, hijau dan ungu perhitungannya adalah sebagai berikut :

- Segitiga ABC oranye
  Koordinat segitiga ABC oranye A2(-7,9), B2(-7,3),C2(-4,3). Maka pergeseran translasi sebanyak x,y (-10,0).
- Segitiga ABC ungu
  Koordinat segitiga ABC ungu A3(3,-4), B3(3,-10), C3(6,-10). Maka pergeseran translasi sebanyak x,y (0,-13).
- Segitiga ABC hijau
  Koordinat segitiga ABC hijau A4(3,-4), B4(3,-10), C4(6,-10). Maka pergeseran translasi sebanyak x,y (-10,-13).

Dari perhitungan di atas maka transformasi pergeseran / translasi dapat dirumuskan sebagai berikut :

$$P_{(x,y)} \xrightarrow{T\binom{a}{b}} P'_{(x+a,\ y+b)} \qquad (2.\ 5)$$

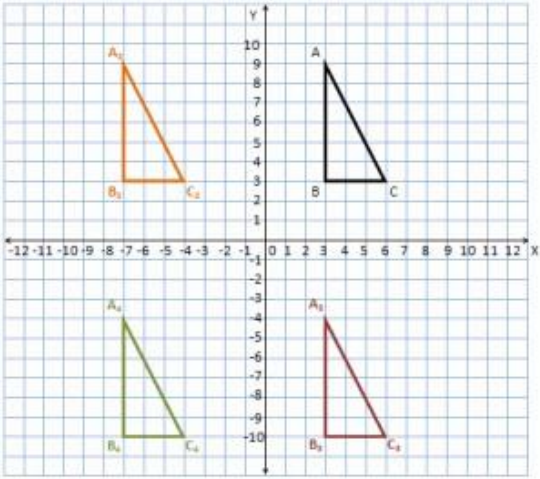
Dimana :

P = koordinat awal

P’ = koordinat akhir (setelah di transformasi translasi)

a = pergeseran horisontal (+ ke kanan, - ke kiri)

b = pergeseran vertikal (+ ke atas, - ke bawah)



Gambar 2. 8 Contoh Transformasi Translasi *[7]*

2.4.2. Rotasi (Perputaran)

Transformasi rotasi atau perputaran merupakan sebuah proses yang dilakukan untuk memutar sebuah bangun pada koordinat kartesian terhadap titik pusat tertentu sebanyak sudut tertentu tanpa merubah bentuk dan ukuran dari bangun tersebut. Pada tranformasi rotasi, bentuk dan ukuran bangun tidak berubah, yang berubah adalah arah dan posisi terhadap keadaan awal.

Pada Gambar 2. 9 segitiga ABC berwarna hitam mempunyai koordinat A(3,9), B(3,3), C(6,3), apabila ditransformasikan rotasi mejadi segitiga ABC warna oranye, hijau dan ungu perhitungannya adalah sebagai berikut :

- Segitiga ABC oranye
  Koordinat segitiga ABC oranye A2(-9,3), B2(-3,3),C2(-3,6). Maka pergeseran rotasi dengan sudut +90° atau -270° dengan pusat rotasi pada O(0,0).
- Segitiga ABC ungu
  Koordinat segitiga ABC ungu A3(9,-3), B3(3,-3), C3(3,-6). Maka pergeseran rotasi dengan sudut +270° atau -90° dengan pusat rotasi pada O(0,0).

- Segitiga ABC hijau
  Koordinat segitiga ABC hijau A4(3,-9), B4(-3,-3), C4(-6,-3). Maka pergeseran rotasi dengan sudut +180° atau -180° dengan pusat rotasi pada O(0,0).

Dari perhitungan di atas maka transformasi rotasi / perputaran dapat dirumuskan sebagai berikut :

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x-a \\ y-b \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} \qquad (2.6)$$

Dimana :

x = koordinat x awal

y = koordinat y awal

x' = koordinat x akhir (setelah ditransformasi rotasi)

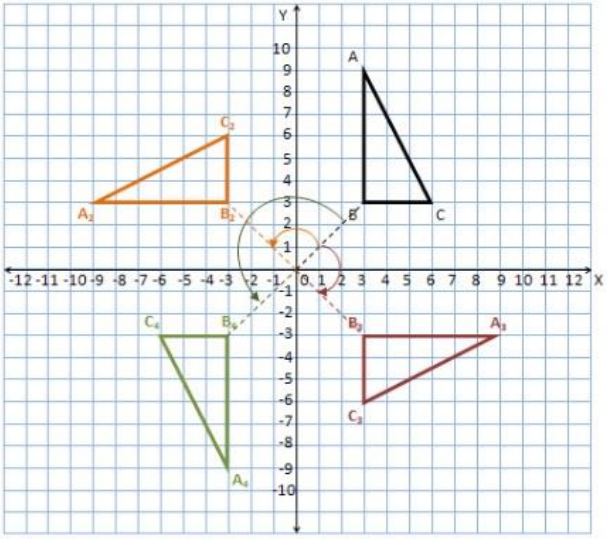
y' = koordinat y akhir (setelah ditransformasi rotasi)

θ = sudut putar

a = sumbu x pusat putar

b = sumbu y pusat putar

Pada transformasi rotasi, sudut putar (θ) bernilai positif (+) akan menyebabkan bangun berputar berlawanan arah jarum jam (counter clockwise) sedangkan apabila bernilai negatif (-) akan menyebabkan bangun berputar searah jarum jam (clockwise).



Gambar 2. 9 Contoh Transformasi Rotasi *[7]*

2.5. Mikrokontroler STM32F4 Discovery

STM32F4 discovery adalah sebuah modul mikrokontroler yang di dalamnya menggunakan IC STM32F407VGT6 ARM Cortex-M4 yang mempunyai kecepatan sampai dengan 168 MHz, serta mampu mengeksekusi perintah hingga 210 MIPS (Million Instruction per Second). STM32F4 discovery merupakan mikrokontroler 32 bit dengan arsitektur ARM. Mikrokontroller ini memiliki kapasitas 1 MByte Flash PEROM (Flash Programmable and Eraseble Read Only Memory), 192 Kbyte SRAM. Dilengkapi dengan 100 buah pin input output yang mempunyai karakteristik masing-masing yaitu USART, TIMER, ADC dan I2C.

Fitur dari STM32F4 Discovery antara lain yaitu :

- *On-board* ST-LINK/V2 yang digunakan untuk proses upload file hex ke dalam mikrokontroler.
- ST MEMS *motion sensor*, 3-axis accelerometer.
- Audio DAC dengan driver speaker kelas D
- Dua push button (user dan reset)
- USB OTG

Bahasa pemrograman pada mikrokontroller ini adalah bahaca C. Media pemrograman yang dapat digunakan untuk memprogram STM32F4 Discovery yaitu antara lain :

- Altium, TASKING VX-Toolset
- Atlollic TrueSTUDIO
- IAR Embedded Workbench for ARM (EWARM)
- Keil, MDK-ARM
- Coocox

Gambar 2. 10 Board STM32F4-Discovery *[8]*

2.6. Dynamixel RX-28

Dynamixel RX-28 merupakan sebuah servo yang mengintegrasikan *speed reducer*, *controller*, *driver*, dan *network function* menjadi satu modul. Servo RX-28 dapat dikoneksikan secara mudah menjadi bentuk-bentuk robot yang bermacam-macam. Setiap servo mempunyai ID yang dapat ditentukan dan diubah-ubah. Servo ini dikendalikan dengan paket komunikasi pada sebuah *bus* dengan menggunakan RS485. [9]

2.6.1. Spesifikasi Dynamixel RX-28

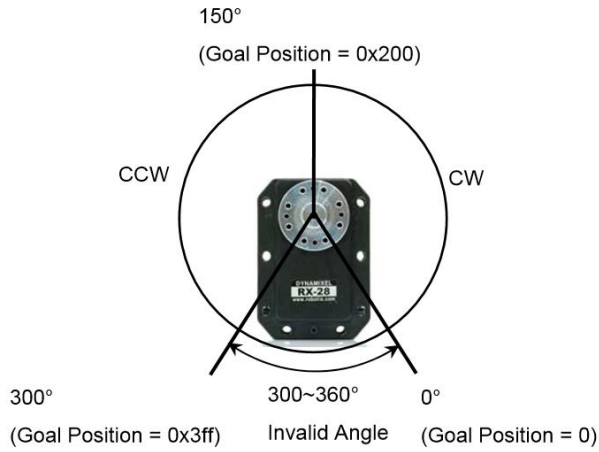
Spesifikasi dari servo RX-28 yaitu :

- Tegangan supply 12 ~ 16 V (tegangan rekomendasi 14.4V)
- Sudut putar bebas 300° dengan resolusi sudut 0.29° (0 ~ 1024)
- Kekuatan torsi 28,3 ~ 37,7 kgf.cm.
- Kecepatan 0.167 ~ 0.126 detik per 60°.
- Arus maksimal 1200mA, arus keadaan *standby* 50mA

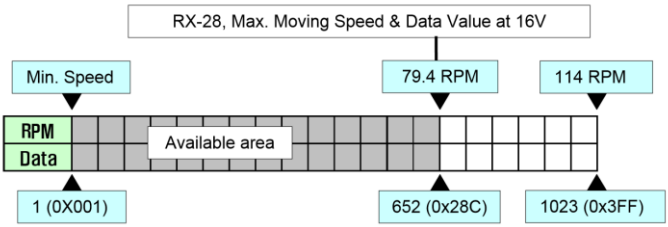
- Protokol pengiriman data RS485 Asynchronous Serial Communication (8bit, 1stop, No Parity) dengan kecepatan komunikasi 7343bps ~ 1 Mbps.
- ID servo 0~253
- Motor Maxon RE-MAX
- *Sensing* dan *measuring* : posisi, temperatur, beban, tegangan input, dll.
- Material gear *full metal*

Pada servo ini dapat mengendalikan posisi dan kecepatan dengan resolusi 1024 serta dapat pula membaca posisi dan kecepatan servo. Mikrokontroller dpat mengontrol kecepatan, posisi, torsi, dll dengan satu paket data, serta dapat mengontrol banyak servo dengan satu paket data. Terdapat penanda dalam bentuk status LED saat temperatur, torsi, tegangan supply menyimpang dari yang telah di tetapkan oleh *user*, serta terdapat fasilitas *shutdown* dimana servo akan otomatis mati sendiri pada keadaan tertentu sesuai yang *user* tetapkan.

Rentang *goal position* pada servo dynamixel RX-28 adalah 0 ~ 300°, sehingga servo tidak dapat digerakkan pada sudut servo 300 ~ 360° atau pada sudut ini disebut sebagai *invalid angle*. Untuk menentukan *goal position* digunakan skala 0 sampai 1023 (10 bit) dengan 0 (0x00) = 0° dan 1023 (0x3FF) = 360°.



Gambar 2. 11 Rentang *Goal Position [9]*

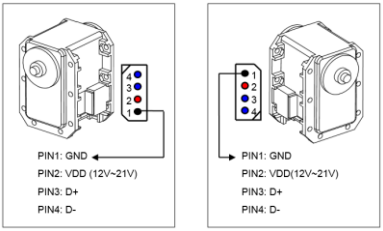


Gambar 2. 12 Hubungan Antara Nilai Data dan Kecepatan RPM [9]

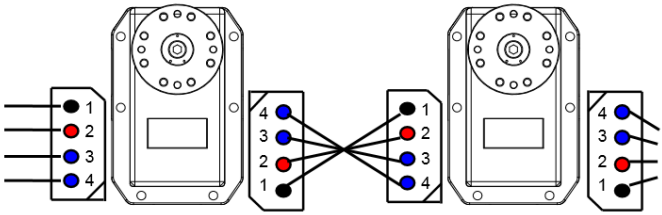
Rentang kecepatan pada servo RX-28 adalah 0 sampai 1023 (0x3FF). Kecepatan pada servo RX-28 dapat dikonversikan menjadi besaran RPM dengan mengalikan data kecepatan dengan 0.111. Contohnya jika data kecepatan yang dikirimkan adalah 1023 maka kecepatan dalam RPM adalah 1023 x 0.111 = 113.6. Tetapi kecepatan riil pada servo ditentukan oleh banyak faktor luar. Kecepatan maksimal dari servo RX-28 berbanding lurus dengan ukuran tegangan suplai. Semakin besar tegangan suplai maka semakin besar area kecepatan yang dapat dikendalikan. Contohnya ketika RX-28 disuplai dengan 16V maka dapat mengendalikan kecepatan 0 sampai 79.4 RPM, sedangkan apabila disuplai dengan tegangan 12V maka kecepatan maksimal akan berkurang menjadi 59.9RPM. Hubungan dari nilai data dan kecepatan dapat direprsentasikan pada gambar berikut :

2.6.2. Pengkabelan Dynamixel RX-28

Pada servo RX-28 terdapat empat buah port yang terdiri dari GND, VDD, D+ dan D-. Menghubungkan beberapa RX-28 dapat dilakukan melalui BUS.



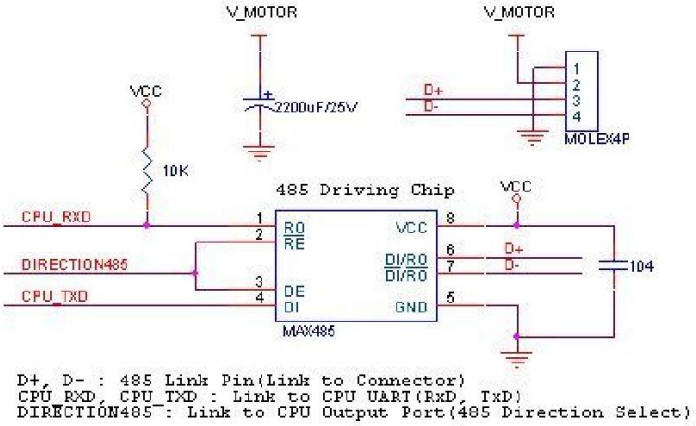
(a)



(b)

Gambar 2. 13 Susunan Pin RX-28 (a), Susunan Koneksi (b) [9]

2.6.3. RS-485

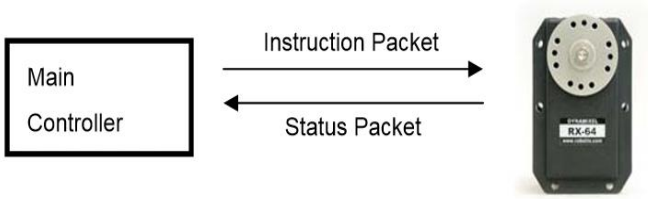


Gambar 2. 14 Rangkaian RS485 [9]

Untuk mengontrol RX-28 dengan mikrokontroller, data pada UART (*Universal Asynchronous Receiver/Transmitter*) harus di konversikan menggunakan RS485 kemudian menuju ke servo. Rangkaian RS485 menurut datasheet sebagai berikut :

Supply dari RX-28 lewat Pin1(-) dan Pin2 (+). Port DIRECTION485 akan menentukan arah dari sinyal data pada TxD dan RxD. Jika DIRECTION485 bernilai 1 maka sinyal pada TxD akan keluar pada D+ dan D-, sedangkan apabila DIRECTION485 bernilai 0 maka sinyal D+ dan D- akan keluar pada RxD, sehingga port DIRECTION485 akan menentukan arah sinyal yaitu sebagai transmitter atau receiver.

2.6.4. Pengiriman Data Dynamixel RX-28



Gambar 2. 15 Analogi *Instruction Packet* Dan *Status Packet* [9]

Untuk mengontrol pergerakan RX-28, data yang dikirimkan harus sesuai dengan protokol RX-28. RX-28 digerakaan dengan menerima data biner. Mikrokontroller dan RX-28 berkomunkasi satu sama lain (dua arah) dengan mengirimkan dan meneruma data yang disebut paket. Paket terbagi menjadi dua yaitu :

- *Instruction packet*, dimana mikrokontroller mengirimkan data kontrol ke RX-28
- *Status packet*, dimana RX-28 merespon data ke Mikrokontroler.

ID merupakan sebuah angka spesifik untuk membedakan masing-masing RX-28 ketika beberapa RX-28 dikoneksikan menjadi satu pada sebuah bus. Dengan menggunakan ID pada Instuction dan Status Packet, mikrokontroler dapat mengendalikan RX-28 yang ingin dikendalikan. RX-28 menggunakan *Asynchronous Serial Communication Receiver/ Transmitter* (UART) dengan 8 bit, 1 Stop bit, dan None parity. Pada sebuah bus apabila RX-28 dengan ID yang sama dikoneksikan maka akan terjadi masalah pada pergerakan servo, maka pada sebuah bus tidak boleh ada ID yang sama.

Paket instruksi yang dikirimkan dari mikrokontroler ke RX-28 mempunyai struktur sebagai berikut :

OXFF	0XFF	ID	LENGTH	INSTRUCTION	PARAMETER1	…	PARAMETER N	CHECK SUM

Dimana maksud dari masing-masing byte paket adalah sebagai berikut :

- 0XFF 0XFF
  Data ini mengindikasikan awal sebuah paket.
- ID
  Merupakan ID pada sebuah RX-28 dimana akan menerima *instruction packet.* Dapat digunakan 254 macam ID mulai dari

0-253 (0X00 ~ 0XFD). Dimana 254 (0XFE) merupakan *broadcasting* ID, broadcasting ID digunakan untuk mengendalikan semua RX-28 yang terkoneksi menjadi satu, tetapi *status packet* tidak dikirimkan.

- LENGTH
  Merupakan pangjang dari sebuah paket data. Length dihitung berdasarkan nilai parameter (N) + 2.
- INSTRUCTION
  Instruction merupakan sebuah perintah yang diberikan ke RX-28. *Intruction* memiliki banyak tipenya yaitu seperti Tabel 2. 1.
- PARAMETER0...N
  Merupakan sebuah data parameter yang dikirimkan berdasarkan jenis instruksi yang digunakan.

Tabel 2. 1 Tipe Instruksi Dynamixel RX-28

Value	Nama	Fungsi	No Parameter
0x01	PING	Tidak ada eksekusi. Hanya digunakan saat mikrokontroller siap untuk menerima *Status Packet*	0
0x02	READ DATA	Untuk membaca data dari RX-28	2
0x03	WRITE DATA	Untuk menulis data pada RX-28	2 atau lebih
0x04	REG WRITE	Mirip dengan WRITE DATA namun tidak akan dieksekusi hingga perintah ACTION diterima.	2 atau lebih
0x05	ACTION	Untuk mengeksekusi perintah dari REG WRITE	0
0x06	RESET	Untuk mengembalikan setting dari RX-28 menjadi standar pabrikan	0
0x83	SYNC WRITE	Untuk menggerakkan beberapa RX-28 secara bersamaan dalam satu waktu	4 atau lebih

- CHECK SUM
  Digunakan untuk memastikan apakah paket data rusak pada saat pengiriman. *Checksum* dihitung berdasarkan rumus :
  Checksum=
  ~(ID+*length*+*instruction*+*parameter1*+..*parameterN*).
  Dimana "~" merupakan operator Not Bit.
  Ketika nilai perhitungan dari *Checksum* lebih dari 255 (0xFF) maka yang digunakan hanya bagian *lower bytes*.

Pada setiap instruksi terdapat parameter, dan setiap parameter mempunyai peraturan masing-masing untuk pengirimannya. Berikut ini merupakan tabel kontrol :

Tabel 2. 2 Tabel Kontrol *[9]*

	Address (hexadecimal)	Name	Description	Access	Initial Value (Hexadecimal)
EEPROM Area	0 (0X00)	Model Number(L)	Lowest byte of model number	R	28 (0X1C)
	1 (0X01)	Model Number(H)	Highest byte of model number	R	0 (0X00)
	2 (0X02)	Version of Firmware	Information on the version of firmware	R	-
	3 (0X03)	ID	ID of Dynamixel	RW	1 (0X01)
	4 (0X04)	Baud Rate	Baud Rate of Dynamixel	RW	34 (0X22)
	5 (0X05)	Return Delay Time	Return Delay Time	RW	250 (0XFA)
	6 (0X06)	CW Angle Limit(L)	Lowest byte of clockwise Angle Limit	RW	0 (0X00)
	7 (0X07)	CW Angle Limit(H)	Highest byte of clockwise Angle Limit	RW	0 (0X00)
	8 (0X08)	CCW Angle Limit(L)	Lowest byte of counterclockwise Angle Limit	RW	255 (0XFF)
	9 (0X09)	CCW Angle Limit(H)	Highest byte of counterclockwise Angle Limit	RW	3 (0X03)
	11 (0X0B)	the Highest Limit Temperature	Internal Limit Temperature	RW	80 (0X50)
	12 (0X0C)	the Lowest Limit Voltage	Lowest Limit Voltage	RW	60 (0X3C)
	13 (0X0D)	the Highest Limit Voltage	Highest Limit Voltage	RW	240 (0XF0)
	14 (0X0E)	Max Torque(L)	Lowest byte of Max. Torque	RW	255 (0XFF)
	15 (0X0F)	Max Torque(H)	Highest byte of Max. Torque	RW	3 (0X03)
	16 (0X10)	Status Return Level	Status Return Level	RW	2 (0X02)
	17 (0X11)	Alarm LED	LED for Alarm	RW	36 (0X24)
	18 (0X12)	Alarm Shutdown	Shutdown for Alarm	RW	36 (0X24)
RAM Area	24 (0X18)	Torque Enable	Torque On/Off	RW	0 (0X00)
	25 (0X19)	LED	LED On/Off	RW	0 (0X00)
	26 (0X1A)	CW Compliance Margin	CW Compliance margin	RW	0 (0X00)
	27 (0X1B)	CCW Compliance Margin	CCW Compliance margin	RW	0 (0X00)
	28 (0X1C)	CW Compliance Slope	CW Compliance slope	RW	32 (0X20)
	29 (0X1D)	CCW Compliance Slope	CCW Comilance slope	RW	32 (0X20)
	30 (0X1E)	Goal Position(L)	Lowest byte of Goal Position	RW	-
	31 (0X1F)	Goal Position(H)	Highest byte of Goal Position	RW	-
	32 (0X20)	Moving Speed(L)	Lowest byte of Moving Speed	RW	-
	33 (0X21)	Moving Speed(H)	Highest byte of Moving Speed	RW	-
	34 (0X22)	Torque Limit(L)	Lowest byte of Torque Limit	RW	ADD14
	35 (0X23)	Torque Limit(H)	Highest byte of Torque Limit	RW	ADD15
	36 (0X24)	Present Position(L)	Lowest byte of Current Position	R	-
	37 (0X25)	Present Position(H)	Highest byte of Current Position	R	-
	38 (0X26)	Present Speed(L)	Lowest byte of Current Speed	R	-
	39 (0X27)	Present Speed(H)	Highest byte of Current Speed	R	-
	40 (0X28)	Present Load(L)	Lowest byte of Current Load	R	-
	41 (0X29)	Present Load(H)	Highest byte of Current Load	R	-
	42 (0X2A)	Present Voltage	Current Voltage	R	-
	43 (0X2B)	Present Temperature	Current Temperature	R	-
	44 (0X2C)	Registered Instruction	Means if instruction is registered	RW	0 (0X00)
	46 (0X2E)	Moving	Means if there is any movement	R	0 (0X00)
	47 (0X2F)	Lock	Locking EEPROM	RW	0 (0X00)
	48 (0X30)	Punch(L)	Lowest byte of Punch	RW	32 (0X20)
	49 (0X31)	Punch(H)	Highest byte of Punch	RW	0 (0X00)

2.6.5. *Sync Write*

Pada RX-28 terdapat sebuah *instruction* yang dapat mengendalikan beberapa RX-28 secara bersamaan dengan satu buah paket pengiriman data yang bernama *SYNC WRITE*. Dengan menggunakan perintah *SYNC WRITE*, beberapa perintah dapat dikirimkan pada satu waktu, sehingga waktu yang dibutuhkan untuk komunikasi akan berkurang ketika mengendalikan banyak RX-28. Tetapi, pada perintah *SYNC WRITE* hanya dapat dipakai jika alamat dan panjang data yang digunakan untuk mengontrol robot adalah sama. ID yang dikirimkan haruslah menggunakan *broadcasting* ID. Pada perintah ini panjang paketnya tidak boleh lebih dari 143 bytes.

Format pengiriman data pada metode *sync write* adalah sebagai berikut :

- ID
  0xFE
- Length
  (L+1) x N + 4 (L : Data length setiap RX-28, N : nomor terakhir / jumlah RX-28)
- Parameter1
  Start Address untuk mengirim data
- Parameter2
  Panjang data untuk yang dikirim
- Parameter3
  ID RX-28 pertama
- Parameter4
  Data pertama RX-28
- Parameter5
  Data kedua RX-28
  .....
- Parameter L+3
  Panjang data servo RX-28 pertama
- Parameter L+4
  ID RX-28 Kedua
- Parameter L+5
  Data pertama RX-28 Kedua
- Parameter L+6
  Data kedua RX-28 Kedua

.....

- Parameter 2L+4
  Panjang data RX-28 Kedua

Contoh penggunakan metode *sync write* adalah sebagai berikut :

- ID 0 : posisi 0x010 kecepatan 0x150
- ID 1 : posisi 0x220 kecepatan 0x360
- ID 2 : posisi 0x030 kecepatan 0x170
- ID 3 : posisi 0x220 kecepatan 0x380

Maka *instruction packet* yang dikirimkan adalah sebagai berikut :

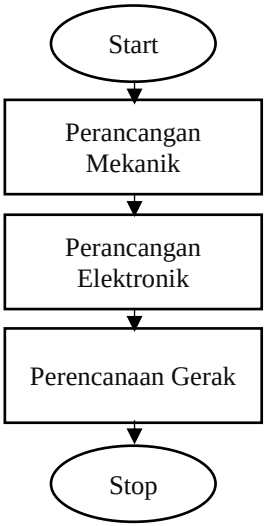
0XFF 0XFF 0XFE 0X18 0X83 0X1E 0X04 0X00 0X10 0X00 0X50 0X01 0X01 0X20 0X02 0X60 0X03 0X02 0X30 0X00 0X70 0X01 0X03 0X20 0X02 0X02 0X80 0X03 0X12.

BAB III
PERANCANGAN SISTEM

Pada bab ini akan dijelaskan mengenai perancangan *inverse kinematics* yang diterapkan pada *quadruped* robot. Pada proses perancangan sistem terdiri dari beberapa bagian, yaitu perancangan mekanik, perancangan elektronik dan perancangan gerak.

Perancangan mekanik dilakukan terlebih dahulu kemudian dilanjutkan dengan perancangan elektronik. Perancangan mekanik dan elektronik ini dilakukan pada proses pertama kali dikarenakan perhitungan-perhitungan pada perancangan software mengacu pada perancangan mekanik dan elektronik, sehingga perancangan elektronik dan mekanik dilakukan pertama kali.

Perancangan mekanik terdiri dari desain *base* robot dan susunan kaki, serta desain kepala robot yang digunakan untuk navigasi robot dalam menemukan api serta memadamkan api. Pada proses desain robot mengacu pada peraturan Kontes Robot Pemadam Api Indonesia (KRPAI) kategori berkaki yaitu ukuran maksimal robot adalah 31x31x27 cm, pada bagian kepala terdapat panel interface seperti *kill plug*, *sound activation*, dll, serta terdapat sistem pemadaman api menggunakan cairan.



Gambar 3. 1 Diagram Alur Perancangan Sistem

Pada perancangan elektronik terdiri dari desain shield pcb untuk STM32F4 discovery yang didalamnya terdapat port-port yang digunakan untuk sistem komunikasi dengan servo RX-28 serta untuk mendukung robot dapat menyelesaikan misinya yaitu untuk proses navigasi dan proses mendeteksi serta memadamkan api.

Perancangan gerak robot dibagi menjadi beberapa langkah, yaitu pengiriman paket data ke servo, kalibrasi posisi *default* kaki, perumusan *inverse kinematics*, perumusan algoritma *gait*, perumusan manuver gerakan robot, dan yang terakhir adalah perumusan trayektori ujung kaki. Semua proses pada perancangan software dilakukan satu-persatu secara berurutan karena masing-masing sangat berpengaruh terhadap proses yang berikutnya.

Proses pertama pada tahap perancangan gerak robot yaitu proses pengiriman paket data dari mikrokontroler menuju ke servo, pada proses ini dilakukan pengecekan apakah paket data yang dikirimkan sudah benar. Apabila paket data yang dikirimkan sudah benar ditunjukkan dengan semua servo yang dapat bergerak dengan posisi dan kecepatan yang diinginkan maka tahap selanjutnya adalah kalibrasi posisi *default* kaki.

Langkah kedua pada tahap perancangan gerak robot adalah dilakukan kalibrasi posisi servo *default*, posisi *default* merupakan sebuah posisi dimana masing-masing lengan/kaki tegak lurus dari badan. Hal ini dilakukan agar mudah dalam merumuskan *inverse kinematics* pada robot. Nilai-nilai yang didapatkan pada tahap kalibrasi kemaudian dijadikan nilai dasar servo sebelum diolah.

Langkah ketiga dalam perancangan gerak robot adalah perumusan *inverse kinematics*. Pada langkah ini *inverse kinematics* menjadi dasar dari nilai-nilai tiap servo pada saat digerakkan. *Inverse kinematics* merupakan sebuah metode untuk mengubah besaran kartesian posisi *end-effector* menjadi besaran sudut pada tiap-tiap *joint*. Setelah dianggap posisi *end-effector* pada tiap-tiap kaki sesuai dengan apa yang kita inginkan, maka langkah selanjutnya adalah menentukan algoritma *gait* pada pergerakan robot.

Algoritma *gait* adalah sebuah algoritma yang merumuskan masing-masing kaki kapan berada pada *fase support* atau pada *fase transfer*. Algoritma *gait* dilakukan setelah rumusan *inverse kinematics* sudah sesuai dengan yang diinginkan. Penerapan algoritma *gait* yang tepat akan menghasilkan pergerakan robot yang baik, sebaliknya apabila algoritma *gait* kurang tepat maka akan menghasilkan pergerakan robot yang lambat.

Langkah kelima dalam perancangan gerak robot adalah perumusan manuver gerakan robot. Pada proses ini dilakukan perumusan transformasi pada masing-masing kaki robot sehingga robot dapat melakukan manuver berjalan seperti belok, putar ditempat, dll. Setelah didapatkan perumusan manuver gerakan robot maka langkah selanjutnya adalah perumusan trayektori ujung kaki.

Tayektori ujung kaki dilakukan agar pergerakan masing-masing kaki menjadi lebih terarah dan halus. Semakin terarah pergerakan masing-masing ujung kaki maka semakin bagus pergerakan dari robot. Namun semakin halus pergerakan ujung kaki juga akan menyebabkan pergerakan kaki robot jadi lebih lambat.

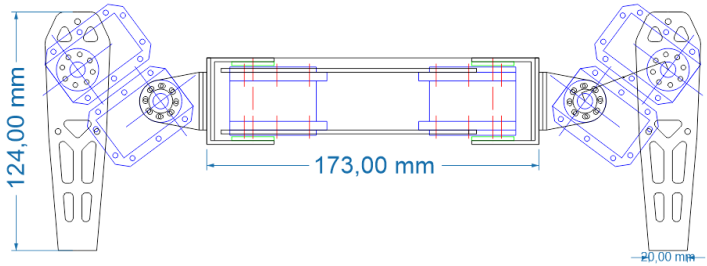
3.1. Mekanik

Proses pembuatan mekanik dibagi menjadi dua jenis yaitu pembuatan mekanik bagian bawah dan mekanik bagian atas. Pada proses ini pendesainan dilakukan dengan menggunakan software Corel Draw untuk desain 2D dan SolidWorks untuk desain 3D. Penggunaan SolidWorks didasari dengan tujuan agar didapatkan model 3D sehingga dapat dianalisa lebih lanjut sebelum dieksekusi ke bentuk yang riil.

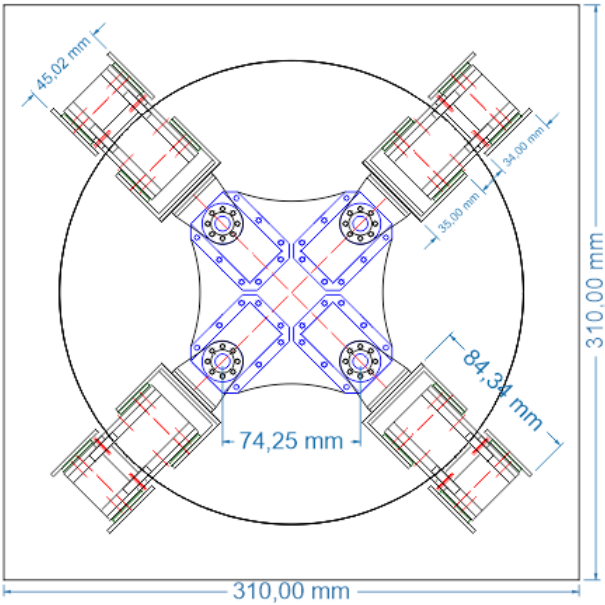
Gambar 3. 2 Desain 3D Keseluruhan Robot

3.1.1. Bagian Bawah

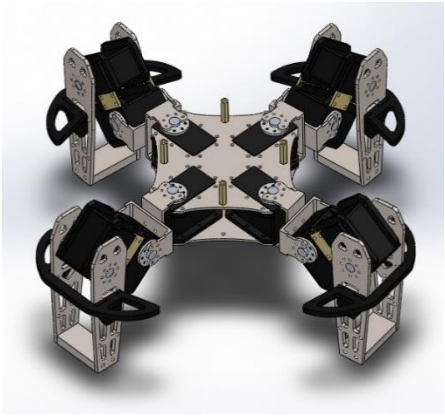
Proses pembuatan mekanik bagian bawah ini meliputi proses pembuatan desain kaki dan *base*. Peletakan servo RX-28 dibuat seperti Gambar 3. 4 *Base* Tampak Atas dikarenakan kecilnya area yang diperbolehkan yaitu maksimal 31x31x27. Pada mekanik bagian bawah semua bagian terbuat dari alumunium agar bagian bawah robot lebih kuat dan meminimalisir terjadinya patah.



Gambar 3. 3 *Base* Tampak Samping



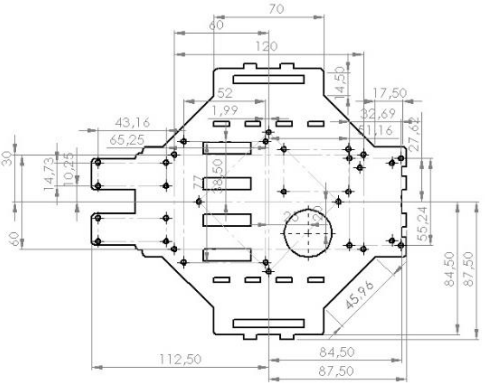
Gambar 3. 4 *Base* Tampak Atas



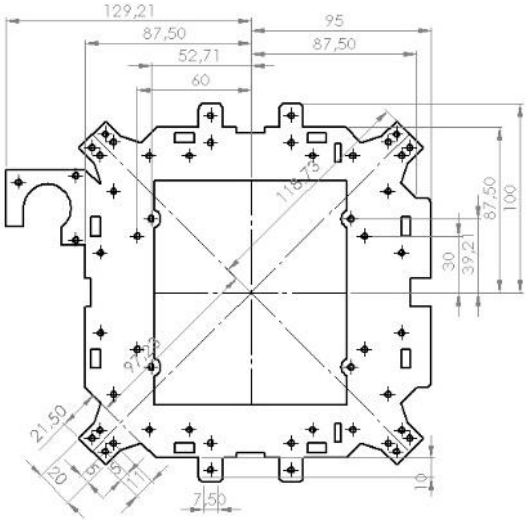
Gambar 3. 5 *Base* 3D

3.1.2. Bagian Atas

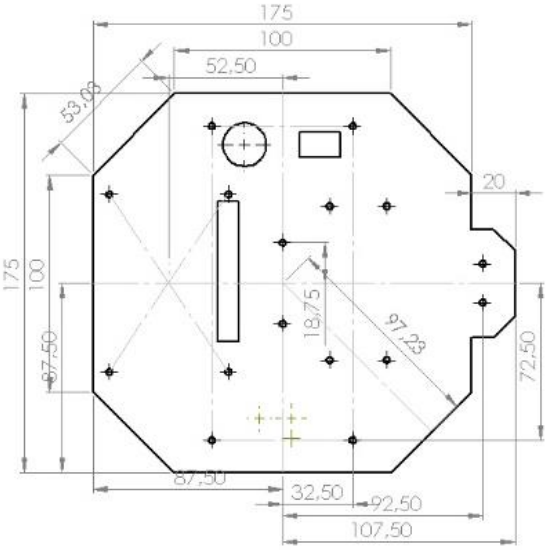
Komponen robot bagian atas didesain berdasarkan kebutuhan robot untuk menunjang dalam menelusuri ruangan dan memadamkan api. Dalam menelusuri ruang digunakan sensor SRF-04 dan Sharp sebagai sensor jarak. Sensor infared dan Uv tron untuk mendeteksi api. Pada mekanik bagian atas, semua bagian terbuat dari *acrilic* dengan tujuan agar ringan dan mudah dilakukan perbaikan apabila ada kesalahan.



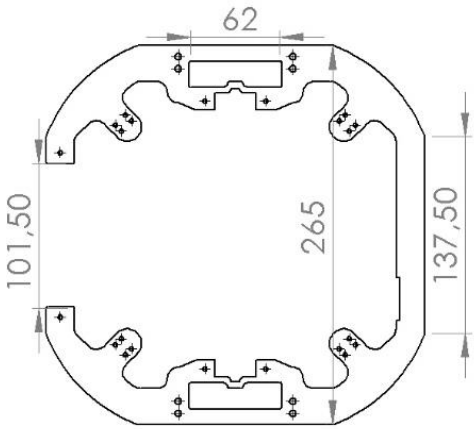
Gambar 3. 6 Kepala Bawah



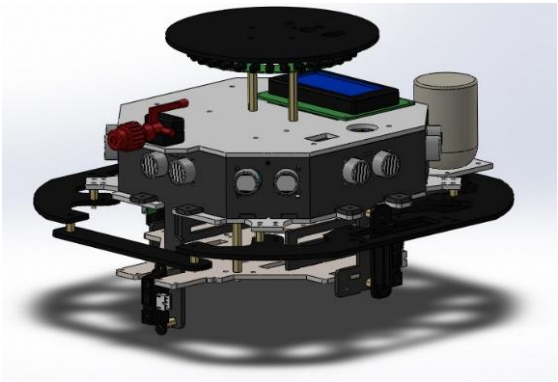
Gambar 3. 7 Kepala Tengah



Gambar 3. 8 Kepala Atas



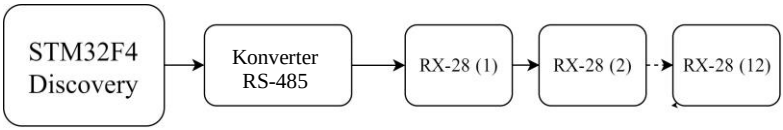
Gambar 3. 9 Bumper



Gambar 3. 10 Kepala Robot 3D

3.2. Elektronik

Proses perancangan elektronik ini merupakan sebuah proses mendesain rangkaian PCB dengan menggunakan *software* eagle. Pada perancangan rangkaian elektronik ini disesuaikan dengan kebutuhan robot terutama kebutuhan robot dalam menelusuri ruang dan memadamkan api. Pada proses ini diperhitungkan juga dimensi dari PCB agar tidak terlalu besar karena dimensi maksimal robot yang disediakan terbatas.

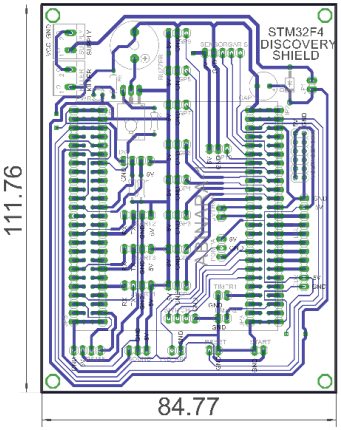


Gambar 3. 11 Diagram Rangkaian Elektronik

STM32F4 discovery merupakan mikrokontroler sebagai pusat pemrosesan yang bertugas untuk mengolah kinematika meliputi *inverse kinematics*, trayektori, algoritma *gait* dan manuver pergerakan robot. RS-485 merupakan protocol komunikasi yang digunakan oleh dynamixel RX-28 untuk berkomunikasi dua arah dengan mikrokontroler. Sedangkan servo RX-28 merupakan aktuator pada robot *quadruped* yang berfungsi untuk pergerakan robot, RX-28 disusun secara paralel sebanyak dua belas buah.

3.2.1. STM32F4 Discovery PCB Shield

STM32F4 Discovery PCB Shield merupakan sebuah PCB yang digunakan untuk meng-ekspansi pin-pin yang terdapat pada mikrokontroler STM32F4 Discovery agar mudah diakses / dihubungkan dengan komponen-komponen atau modul-modul yang terdapat pada robot. Pada robot ini STM32F4 discovery merupakan sebuah mikrokontroller master yang bertugas untuk mengendalikan seluruh pergerakan robot serta membaca sensor jarak dengan menggunakan *sharp*, sensor api UV-tron, sensor suara, dan sensor garis.



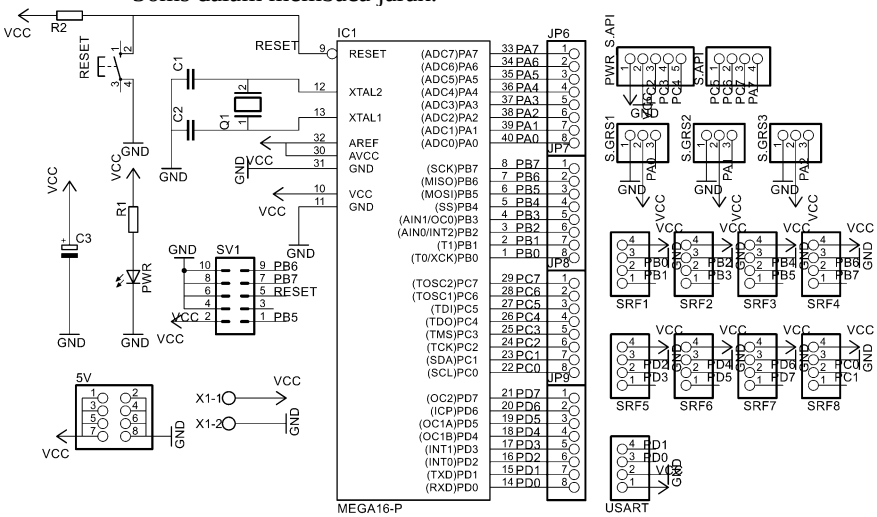
Gambar 3. 12 STM32F4 Discovery PCB *Shield*

Komponen penyusun pada *shield* ini terdiri dari :

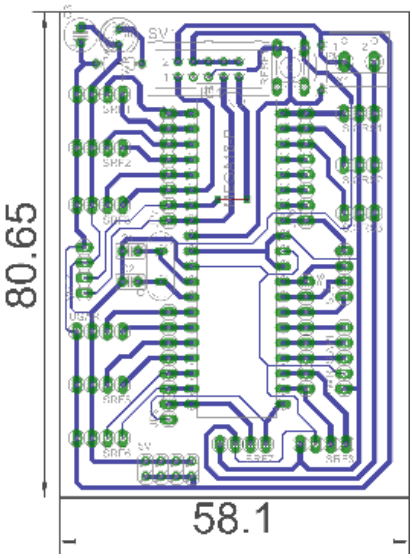
- Led (3mm)
- Kapasitor (10uF)
- Dioda (1N4001)
- Pin Header
- Resistor (10K, 560)
- Transistor BD139
- Buzzer
- Pin Screw

3.2.2. Mikrokontroler Atmega16

Pada robot ini, mikrokontroller atmega16 berperan sebagai slave yang bertugas untuk membaca nilai sensor jarak menggunakan SRF-04 dan sensor api menggunakan infrared dan kemudian mengirimkan data tersebut kepada master yaitu STM32F4 discovery. Alasan menggunakan atmega16 sebagai slave untuk membaca sensor jarak SRF-04 karena dalam proses membaca sensor SRF-04 diperlukan waktu maksimal sebanyak 36ms dalam membaca jarak.



Gambar 3. 13 *Schematic* Mikrokontroler ATMEGA16 (*Slave*)



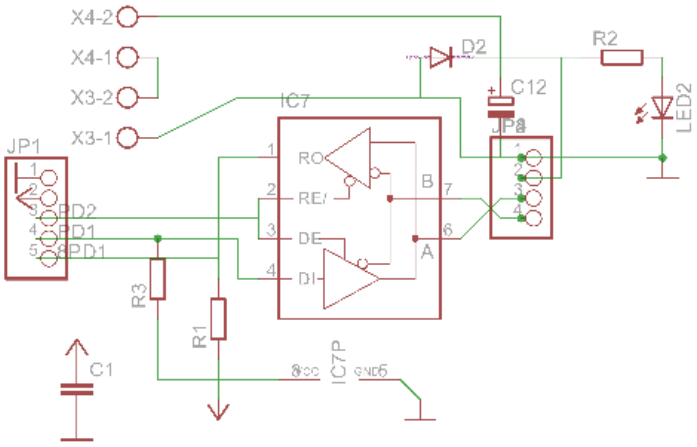
Gambar 3. 14 Board Mikrokontroler ATMEGA16 (Slave)

Komponen penyusun pada mikrokontroler atmega16 ini terdiri dari :

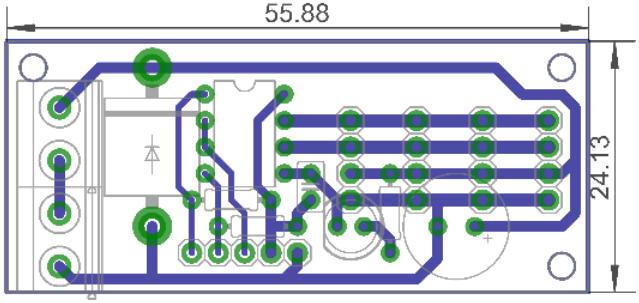
- Atmega16
- Crystal 11.0592 MHz
- Kapasitor (10uF)
- Push Button
- Resistor (10K, 560)
- Pin Header
- Led (3mm)

3.2.3. RS-485

RS-485 merupakan sebuah protocol komunikasi yang digunakan oleh dynamixel RX-28 untuk berkomunikasi dua arah dengan mikrokontroler. RS-485 merupakan protokol komunikasi secara half duplex yang mempu mengirimkan data hingga 1,2 km. Kelebihan dari RS-485 yaitu mampu berkomunikasi dari 32 master dengan 32 slave.



Gambar 3. 15 Schematic RS-485

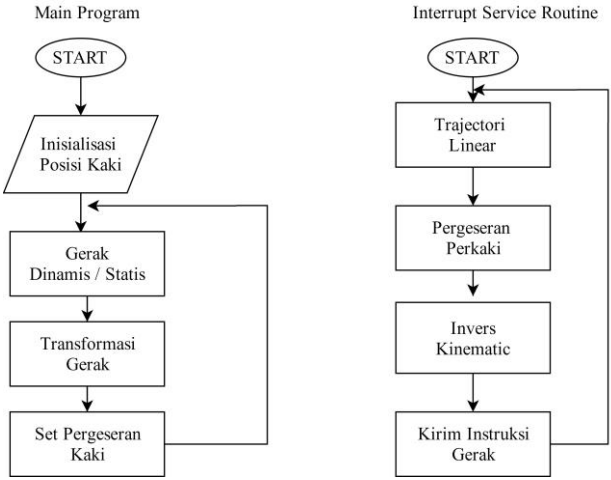


Gambar 3. 16 Board RS-485

Komponen penyusun pada rangkaian RS-485 ini terdiri dari :

- IC Max 485
- Dioda (1N4007)
- Led (5mm)
- Resistor (10K, 560)
- Kapasitor (100uF)
- Pin Header
- Pin Screw

3.3. Perancangan Gerak



Gambar 3. 17 *Flowchart* Sistem Gerak Robot

Pada perancangan gerak ini, proses dibagi menjadi enam bagian yaitu : pengiriman paket data ke servo, kalibrasi *default* posisi kaki, perumusan *inverse kinematics*, perumusan algoritma *gait*, perumusan manuver gerakan robot, dan perumusan trayektori ujung kaki. Semua proses dalam perancangan *software* ini terkait satu persatu sehingga apabila ada satu bagian yang salah maka akan mempengaruhi pada performa jalan robot secara keseluruhan.

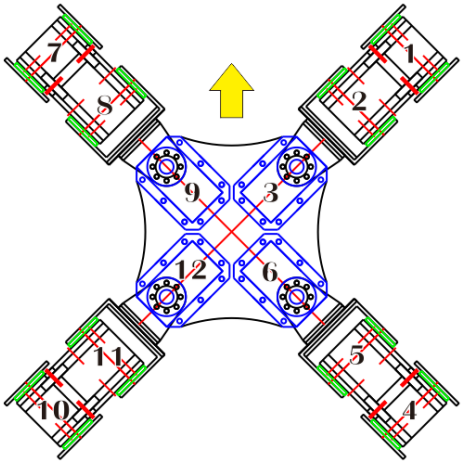
3.3.1. Pengiriman Paket Data ke Servo

Besaran pada servo yang akan dikendalikan oleh mikrokontroler adalah berupa posisi dan kecepakan sehingga data yang akan dikirimkan oleh mikrokontroler ke servo RX-28 yaitu berupa instruksi untuk pengaturan *goal position* dan *speed*. Untuk mempercepat proses pengendalian posisi dan kecepatan pada servo maka digunakan metode *SYNC WRITE*. *SYNC WRITE* adalah sebuah metode yang dapat mengendalikan banyak servo RX-28 dengan menggunakan satu paket data pengiriman secara simultan.

Pada robot ini servo disusun dengan urutan ID sebagai berikut :

Tabel 3. 1 Susunan ID

No	Kaki	ID
1	Kanan Depan Luar	1
2	Kanan Depan Tengah	2
3	Kanan Depan Dalam	3
4	Kanan Belakang Luar	4
5	Kanan Belakang Tengah	5
6	Kanan Belakang Dalam	6
7	Kiri Depan Luar	7
8	Kiri Depan Tengah	8
9	Kiri Depan Dalam	9
10	Kiri Belakang Luar	10
11	Kiri Belakang Tengah	11
12	Kiri Belakang Dalam	12



Gambar 3. 18 Susunan ID Servo

Sesuai dengan susunan ID yang ditetapkan yaitu dari 1 sampai 12 maka paket data yang dikirimkan dengan metode *SYNC WRITE* untuk mengendalikan posisi dan kecepatan tiap servo adalah sebagai berikut :

Data[0] = 0xFF //header
Data[1] = 0XFF //header
Data[2] = 0xFE //broadcast instruction
Data[3] = 64 //length=((lengthdata+1)*jumlah servo + 1)
Data[4] = 0x83 //sync write
Data[5] = 30 //start address to write data
Data[6] = 4 // lengt of data to write
Data[7] = 1 // ID first servo
Data[8] = goal position (L) first servo
Data[9] = goal position (H) first servo
Data[10] = speed (L) first servo
Data[11] = speed (H) first servo
Data[12] = 2 // ID second servo
Data[13] = goal position (L) second servo
Data[14] = goal position (H) second servo
Data[15] = speed (L) second servo
Data[16] = speed (H) second servo
.
.
.
.
.
Data[62] = 12 // ID 12th servo
Data[63] = goal position (L) 12th servo
Data[64] = goal position (H) 12th servo
Data[65] = speed (L) 12th servo
Data[66] = speed (H) 12th servo
Data[67] = checksum

Cheksum merupakan sebuah nilai yang digunakan untuk memeriksa apakah terdapat data yang rusak selama komunikasi berjalan. Rumus checksum = ~(ID+Length+Instruction+Parameter 1+...Parameter N).

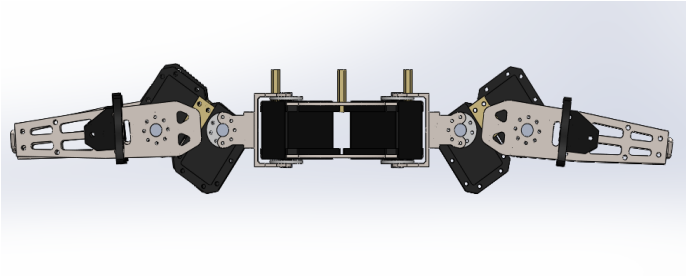
Pengiriman paket data ke servo menggunakan fungsi *interrupt* USART bertujuan agar data yang dikirimkan merupakan data yang *real-time*. Pada robot ini pengiriman data menggunakan

USART6. Baudrate yang digunakan adalah 57600, sedangkan data yang dikirimkan tiap paket berjumlah 67sehingga waktu yang dibutuhkan dalam sekali pengiriman adalah 67*8/57600 = 0.009305556 detik.

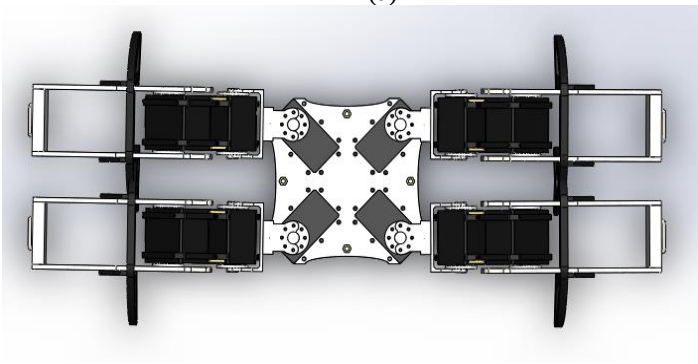
3.3.2. Kalibrasi Posisi *Default* Kaki

Posisi *default* kaki merupakan sebuah posisi standar dimana masing-masing servo belum mendapatkan perhitungan *inverse kinematics*. Posisi *default* ini digunakan untuk memudahkan dalam perumusan *inverse kinematics*. Pada tahap kalibrasi ini dicari nilai masing masing servo yang kemudian dijadikan sebagai acuan nilai *default* masing-masing servo sebelum digerakkan dengan *inverse kinematics*.

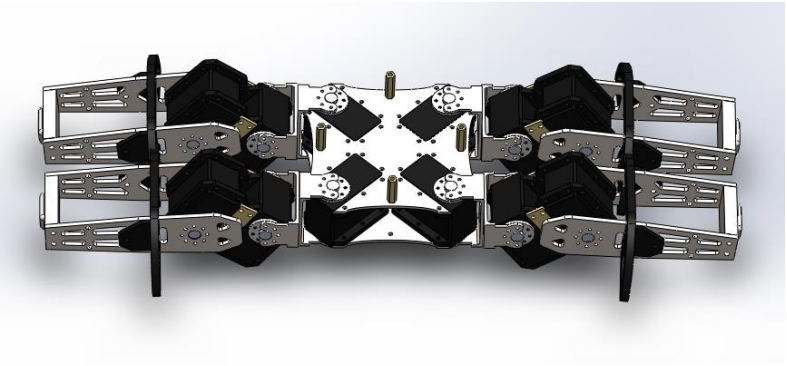
Posisi *defaut* masing-masing kaki pada robot adalah sebagai berikut :



(a)



(b)



(c)

Gambar 3. 19 Posisi *Default* Kaki

Dari posisi *default* tersebut didapatkan nilai sudut tiap servo sebagai berikut :

Tabel 3. 2 Nilai *Default* Tiap Servo

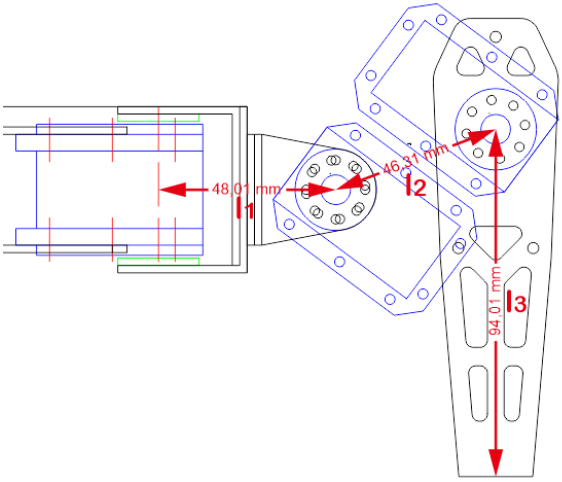
No	Kaki	ID	Nilai *Default*
1	Kanan Depan Luar	1	650
2	Kanan Depan Tengah	2	393
3	Kanan Depan Dalam	3	427
4	Kanan Belakang Luar	4	700
5	Kanan Belakang Tengah	5	415
6	Kanan Belakang Dalam	6	670
7	Kiri Depan Luar	7	700
8	Kiri Depan Tengah	8	420
9	Kiri Depan Dalam	9	350
10	Kiri Belakang Luar	10	720
11	Kiri Belakang Tengah	11	470
12	Kiri Belakang Dalam	12	650

3.3.3. Perumusan *Inverse Kinematics*

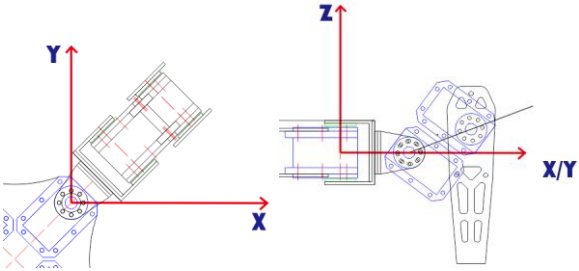
Kinematika merupakan sebuah studi tentang gerak tanpa memperhatikan faktor-faktor yang menyebabkan gerak tersebut. Kinematika robot menghitung hubungan antara posisi *joint* dan posisi lengan robot dalam koordinat kartesian. *inverse kinematics* merupakan sebuah metode untuk megubah besaran kartesian pada masing-masing kaki menjadi besaran sudut pada tiap servo. Posisi dan orientasi ujung lengan (*end-effector*) pada tiap kaki robot dapat ditentukan berdasarkan posisi suut-sudut *joint* dan struktur mekanik robot. Tiap kaki robot mempunyai tiga buah lengan dengan panjang yang berbeda-beda yaitu l1, l2, l3 seperti pada Gambar 3. 20.

Pada robot *quarduped* masing-masing kaki mempunyai tiga derajat kebebasan (*Degree of Freedom*) dengan tipe lengan *elbow up*. Postur kaki robot dapat direpresentasikan dalam bentuk *joint space* dan *cartesian space*.

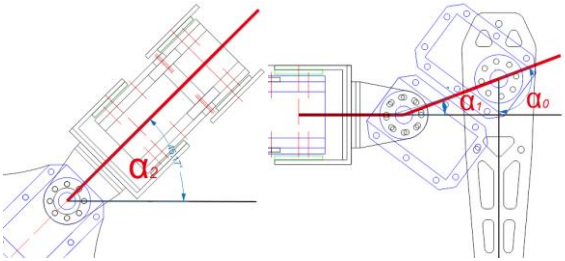
Joint Space : $[\alpha_1\ \alpha_2\ \alpha_3]$
Cartesian Space : $[X\ Y\ Z]$



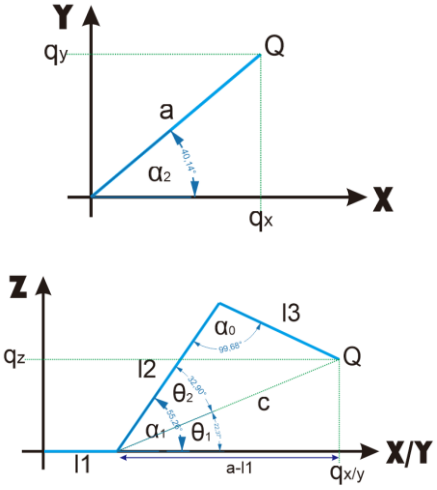
Gambar 3. 20 Susunan Lengan Tiap Kaki



Gambar 3. 21 *Cartesian* Space



Gambar 3. 22 *Joint Space*



Gambar 3. 23 Sketsa Kaki

Dari spesifikasi bentuk mekanik kaki yang telah disusun maka dengan rumus trigonometri bisa didapatkan rumus *inverse kinematics* sebagai berikut :

$$a = \sqrt{{q_x}^2 + {q_y}^2}$$
$$c = \sqrt{(a - l_1)^2 + {q_z}^2}$$
$$\alpha_2 = \tan^{-1}\frac{q_y}{q_x} \quad (3.\ 1)$$

$${l_3} = c^2 + {l_2}^2 - 2.c^2.{l_2}^2.\cos\theta_2$$
$$\theta_2 = -\cos^{-1}\frac{{l_3}^2 - c^2 - {l_2}^2}{2.c^2.{l_2}^2}$$
$$\theta_2 = \cos^{-1}\frac{c^2 + {l_2}^2 - {l_3}^2}{2.c^2.{l_2}^2}$$
$$\theta_1 = \tan^{-1}\frac{q_z}{(a - l_1)}$$
$$\alpha_1 = \theta_1 + \theta_2$$
$$\alpha_1 = \tan^{-1}\frac{q_z}{(a-l_1)} + \cos^{-1}\frac{c^2+{l_2}^2-{l_3}^2}{2.c^2.{l_2}^2} \quad (3.\ 2)$$

$$(a - l_1)^2 + {q_z}^2 = {l_2}^2 + {l_3}^3 - 2.l_2.l_3.\cos(\pi - \alpha_0)$$
$$c^2 = {l_2}^2 + {l_3}^3 + 2.l_2.l_3.\cos\alpha_0$$
$$\alpha_0 = \cos^{-1}\frac{c^2-{l_2}^2-{l_3}^2}{2.l_2.l_3} \quad (3.\ 3)$$

Dari persamaan *inverse kinematics* yang telah didapatkan, maka dapat dicari sudut pada masing-masing servo tiap kaki yaitu α_0,α_1,α_2 dengan memasukkan koordinat *end-effector* tiap kaki robot pada sumbu X,Y,Z. Sehingga dengan meggunakan metode *inverse kinematics* akan memudahkan dan mempresisikan posisi *end-effector* dalam menggerakkan kaki.

Pada perancangan kinematika robot ini, setiap kaki mempunyai koordinat awal yang menjadi posisi *default* kaki robot sebelum mengalami pergeseran karena perumusan manuver gerakan robot dan algoritma *gait*. Berikut merupakan koordinat awal posisi tiap kaki robot sebelum mengalami pergerakan :

Tabel 3. 3 Koordinat Awal tiap Kaki

Kaki	x	y	z
0	6,4	6,4	-9,3
1	6,4	-6,4	-9,3
2	-6,4	-6,4	-9,3
3	-6,4	6,4	-9,3

3.3.4. Perumusan Algoritma *Gait*

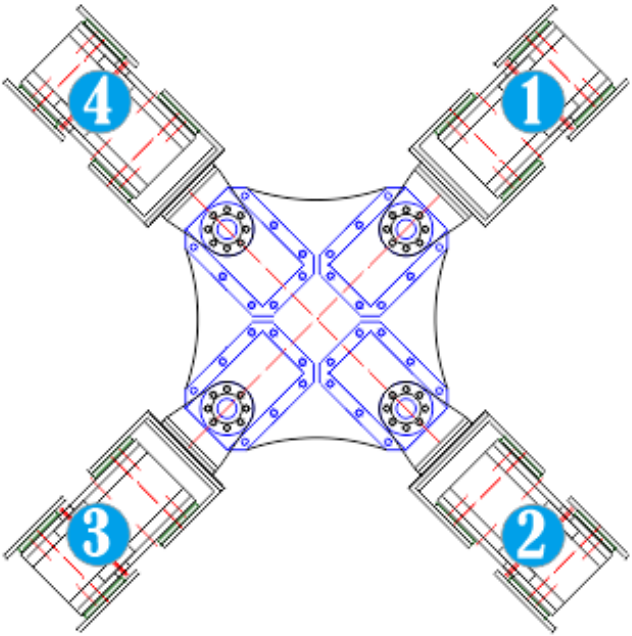
Algoritma pola langkah (*gait*) diperlukan untuk mengatur waktu kapan sebuah kaki berada pada *fase support* dan kapan berada pada *fase transfer*. Prinsip dari perancangan algoritma *gait* adalah menentukan waktu yang tepat untuk masing-masing kaki berada dalam *fase support* maupun *fase transfer*.

Pada robot *quadruped*, mempunyai empat buah kaki. Dari jumlah kaki tersebut maka algoritma *gait* yang dapat diterapkan adalah *dualpod* dan tripod. Algoritma *dualpod* adalah cara berjalan dimana dua kaki berada pada *fase transfer* dan dua kaki yang lain berada pada *fase support*. Sedangkan algoritma tripod adalah cara berjalan dimana satu kaki berada pada *fase transfer* dan tiga kaki berada pada *fase support*.

Untuk mewujudkan pergerakan yang cepat maka pada tugas akhir ini algoritma *gait* yang diterapkan pada robot *quadruped* adalah algoritma *dualpod*. Dengan algoritma ini maka dua kaki akan berada pada *fase support* dan dua kaki lainnya berada pada *fase transfer* seperti pada Gambar 3. 243 dimana garis hitam merupakan *fase transfer* dan garis biru putus-putus merupakan *fase support*.

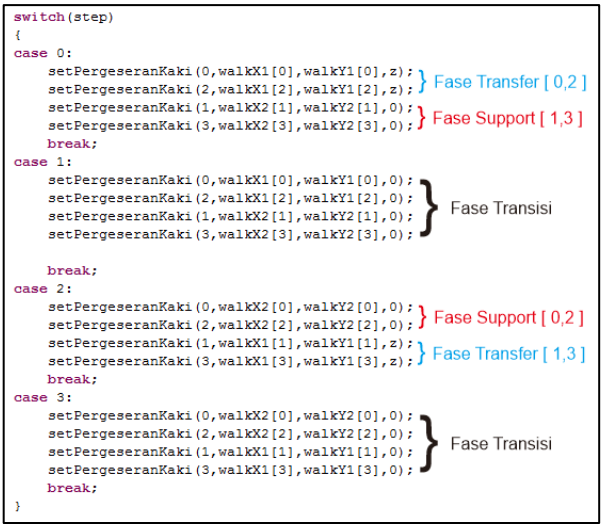
Pada saat kaki 1 dan 3 berada pada *fase support* maka kaki 2 dan 4 berada pada *fase transfer*, sedangkan pada saat kaki 1 dan 3 berada pada *fase transfer* maka kaki 2 dan 4 berada pada *fase support*, begitu terus-menerus secara bergantian.

Gambar 3. 24 Diagram *Dualpod Gait*



Gambar 3. 25 Susunan Kaki *Quadruped*

Implementasi dari algoritma *dualpod gait* pada gerak berjalan robot *quadruped* adalah seperti berikut ini :



Dimana terdapat empat tahap dalam setiap satu *sequence* langkah. Tahap pertama adalah kaki 0 dan 2 berada pada fase *transfer* sedangkan kaki 1 dan 3 berada pada *fase support*. Tahap kedua adalah berupa fase transisi dimana semua kaki menempel pada lantai. Tahap ketiga adalah kaki 0 dan 2 berada pada *fase support* sedangkan kaki 1 dan 3 berada pada *fase transfer*. Tahap keempat adalah berupa fase transisi dimana semua kaki menempel pada lantai. Begitu seterusnya pada pergerakan berjalan.

3.3.5. Perumusan Manuver Gerakan Robot

Pada bagian ini dilakukan perumusan posisi kaki sehingga robot mampu untuk bermanuver gerak dinamis seperti gerakan belok, putar ditempat, dan melakukan gerakan statis seperti *yaw*, *pitch*, *roll*. Perumusan manuver gerakan menggunakan rumus transformasi geometri. Transformasi geometri yang digunakan yaitu transformasi translasi dan rotasi.

3.3.5.1. Gerak Berjalan

Gerak berjalan merupakan sebuah gerak pada robot yang memungkinkan posisi kaki berpindah dari pijakan awal dan menyebabkan robot bergerak maju, belok, dan putar di tempat.

Perumusan manuver gerak dinamis robot didapatkan dari transformasi rotasi dan transformasi translasi terhadap posisi *default* pada masing-masing kaki. Proses rotasi dilakukan terlebih dahulu kemudian dilakukan proses translasi. Proses rotasi menentukan sudut putar badan robot, sedangkan proses translasi menentukan lebar langkah robot.

Masukan pada gerak dinamis adalah berupa ω (sudut putar), x dan y (lebar langkah), serta z (tinggi langkah). Nilai ω dimasukkan ke perhitungan transformasi rotasi sehingga dapat menentukan sudut putar robot, omega bernilai positif akan menyebabkan robot berputar ke kiri, sedangkan apabila bernilai negatif akan berputar ke kanan.

Nilai x dan y menentukan lebar langkah dan arah pergerakan robot. Nilai x positif kan menyebabkan robot bergerak ke depan, sedangkan nilai x negatif akan menyebabkan robot bergerak ke belakang. Nilai y positif akan menyebabkan robot bergerak ke kanan, sedangkan nilai y negatif menyebabkan robot bergerak ke kiri. Apabila nilai x dan y dimasukkan maka akan menyebabkan pergerakan robot menjadi serong.

Rumus transformasi rotasi pada robot adalah sebagai berikut :

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \mathbf{cos}\ \omega & -\mathbf{sin}\ \omega \\ \mathbf{sin}\ \omega & \mathbf{cos}\ \omega \end{bmatrix} \begin{bmatrix} x-a \\ y-b \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} \qquad (3.\ 4)$$

x = koordinat x awal
y = koordinat y awal
x’ = koordinat x akhir (setelah ditransformasi rotasi)
y’ = koordinat y akhir (setelah ditransformasi rotasi)
θ = sudut putar
a = sumbu x pusat putar
b = sumbu y pusat putar

Rumus transformasi translasi pada robot adalah sebagai berikut :

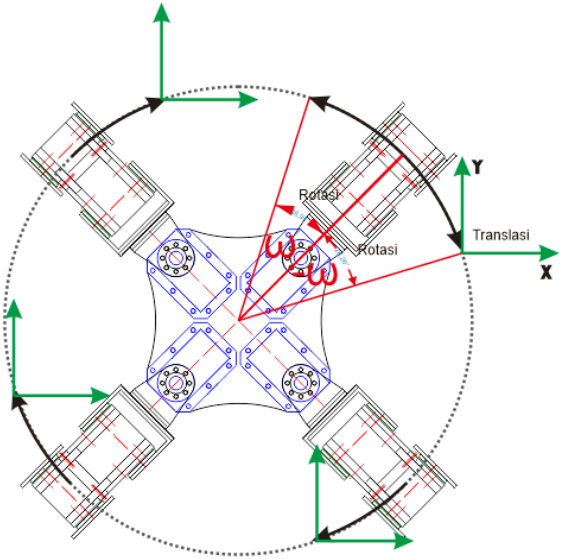
$$P_{(x,y)} \xrightarrow{T\binom{a}{b}} P'_{(x+a,\ y+b)} \quad (3.5)$$

P = koordinat awal

P' = koordinat akhir (setelah di transformasi translasi)

a = pergeseran horisontal (+ ke kanan, - ke kiri)

b = pergeseran vertikal (+ ke atas, - ke bawah)



Gambar 3. 26 Sketsa Gerak

3.3.5.2. Gerak di Tempat

Gerak statis merupakan sebuah gerakan pada robot yang memungkinkan badan robot bergerak tetapi posisi kaki tidak berubah (diam). Gerakan pada gerak di tempat meliputi *yaw*, *pitch*, dan *roll*.

Perumusan gerak statis robot didapatkan dari transformasi rotasi dan transformasi translasi. Hal yang membedakan antara gerak dinamis dan gerak statis adalah pada gerak statis tidak ada algoritma *gait*, sehingga posisi *end-effector* kaki diam pada tempatnya.

Tranformasi rotasi digunakan untuk pergerakan robot memutar badan baik *yaw, pitch* maupun *roll*. Sedangkan transformasi translasi digunakan untuk perpindahan badan robot.

- *Roll*
  Roll adalah gerakan berupa memutar badan ke depan atau belakang. Pada robot ini perumusan *roll* terdiri dari transformasi rotasi pada sudut y dan z serta transformasi linear pada sumbu x,y.
- *Pitch*
  Pitch adalah gerakan berupa memutar badan ke kanan atau kiri. Pada robot ini perumusan *roll* terdiri dari transformasi rotasi pada sudut x dan z serta transformasi linear pada sumbu x,y.
- *Yaw*
  Yaw adalah gerakan berupa memutar badan ke depan atau belakang. Pada robot ini perumusan *yaw* terdiri dari transformasi rotasi pada sudut x dan y serta transformasi linear pada sumbu x,y.

3.3.6. Perumusan Kartesian Trayektori Ujung Kaki

Trayektori ujung kaki merupakan sebuah perencanaan gerak pada ujung kaki robot. Algoritma trayektori menentukan arah gerak dari masing-masing kaki. Tujuan dari penggunaan trayectori pada masing-masing kaki adalah agar pergerakan masing-asing kaki menjadi persamaan garis lurus dari posisi awal hingga posisi akhir (*linear*). Selain itu trayektori juga dapat digunakan untuk menentukan kecepatan jalan robot. Maka dari itu pada robot *quadruped* ini digunakan kartesian trayektori berbasis kecepatan.

Cartesian trajectory tlanning merupakan perencanaan gerak kaki yang menyebabkan pergerakannya sesuai dengan perubahan pada bidang bidang kartesian. Perumusan *Cartesian trajectory planning* yang digunakan adalah persamaan trayektori berbasis kecepatan.

Rumus dari *cartesian trajectory planning* berbasis kecepatan adalah sebagai berikut :

$$q(n.T) = v.T + q\big((n-1).T\big) \quad (3.6)$$

q = Posisi

V = Kecepatan (cm/s)

T = Waktu sampling (s)

Dengan menggunakan rumus kartesian trayektori berbasis kecepatan ini dapat ditentukan posisi kaki pada waktu berikutnya berdasarkan kecepatan yang diinputkan, semakin besar nilai kecepatan maka semakin besar pula perubahan posisi.

Pada robot ini terdapat 3 sumbu pada masing-masing kaki yaitu x,y,z sehingga digunakan penskalaan pada kecepatan masing-masing sumbu sehingga didapatkan kecepatan diagonal yang diharapkan posisi kaki akan bergerak dan sampai pada posisi tujuan dengan waktu yang sama. Rumus dari kecepatan diagonal pada masing-masing sumbu adalah :

$$Vx = V \times \left(\frac{\Delta x}{S}\right) \quad (3.7)$$

$$Vy = V \times \left(\frac{\Delta y}{S}\right) \quad (3.8)$$

$$Vz = V \times \left(\frac{\Delta z}{S}\right) \quad (3.9)$$

$V_{(x\,/\,y/z)}$ = Kecepatan x/y/z

V = Kecepatan global (cm/s)

$\Delta_{(x\,/\,y\,/z\,)}$ = Posisi tujuan x/y/z − posisi sekarang x/y/z

S = Jarak posisi tujuan − posisi awal

Sebagai sampling waktu pada perumusan trayektori robot maka digunakan *interrupt timer*. *Interrupt timer* adalah sebuah fasilitas pada mikrokontroler yang akan mengeksekusi perintah yang terdapat di dalam fungsi *interrupt* pada saat tertentu secara periodik dan akan meninggalkan sementara fungsi *looping* utama, setelah fungsi pada *interrupt* telah selesai dilaksanakan maka program akan kembali lagi ke sistem *looping* utama. Adapun waktu periodik *interrupt* sesuai dengan periode yang telah ditetapkan *interrupt*. Pengunaan *interrupt* bertujuan agar posisi kaki robot akan diperbaharui dalam setiap kurun waktu yang telah ditetapkan pada *interrupt* secara periodik dan tidak mengganggu sistem *looping* utama.

Penetapan periode *interrupt* dilakukan pada nilai *timer prescaler* dan *timer period*. Adapun rumus periode *interrupt* adalah :

$$f = \frac{\frac{TIM\ APB}{tim\_prescaler+1}}{tim\_periode+1} \quad (3.\ 10)$$

$$a = \frac{1}{\frac{\frac{TIM\ APB}{tim\_prescaler+1}}{tim\_periode+1}} \quad (3.\ 11)$$

Pada robot ini periode *interrupt timer* di set pada nilai 0.04 detik sehingga didapatkan nilai *tim_prescaller* = 84-1, dan *tim_periode* = 40000-1.

Penggunaan periode *interrupt timer* 0.04s dikarenakan banyaknya pengiriman data dalam 1 paket adalah 67, sedangkan baudrate komunikasi serial dengan servo adalah 57600, sehingga dalam sekali pengiriman membutuhkan waktu 67*8/57600 = 0.009305556 detik. Nilai tersebut merupakan nilai minimal periode sampling *interrupt timer*.

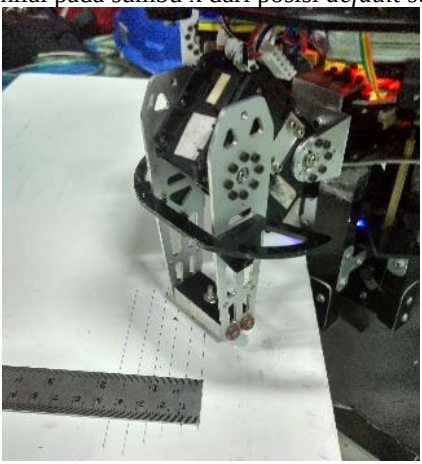
Halaman ini sengaja dikosongkan

BAB IV
PENGUJIAN

Pada bab pengujian ini akan dilakukan pengujian terhadap sistem yang telah dirancang dan dibuat pada bab perancangan. Tujuan dari bab ini adalah sebagai media untuk menguji kinerja dari pergerakan robot *quadruped* apakah sudah sesuai dengan tujuan atau belum. Apabila hasil dari pengujian belum sesuai dengan tujuan maka akan dilakukan perancangan sistem kembali. Pengujian terdiri dari pengujian gerak satu kaki, pengujian trayektori, pengukuran jarak gerak maju, pengukuran kecepatan robot dengan variabel kecepatan trayektori, pengukuran kecepatan robot dengan variabel lebar langkah dan pengujian performa robot berjalan lurus.

4.1. Pengujian Gerak Satu Kaki

Pada pengujian gerak satu kaki ini bertujuan untuk mengetahui tingkat presisi respon posisi dari *end-effector* kaki yang didapatkan dari perhitungan *inverse kinematics*. Pada pengujian ini satu kaki digerakkan pada sumbu kartesian dengan nilai tertentu, kemudian posisi kaki riil akan diukur dengan menggunakan penggaris dan perbedaan nilai *set point* dengan nilai aktual adalah *error*. Cara pengujian pada langkah ini adalah dengan menambah nilai pada sumbu x dari posisi *default* servo.



Gambar 4. 1 Pengujian Gerak Satu Kaki

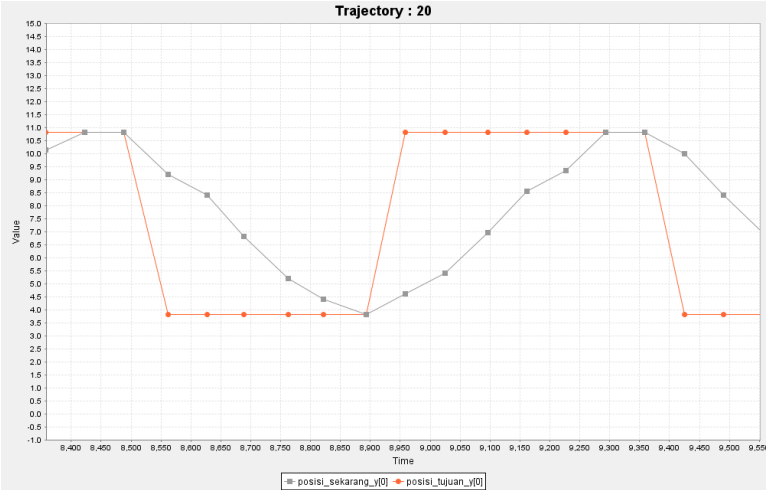
Tabel 4. 1 Posisi Aktual *End-Effector* terhadap *Set Point*

Posisi *Set Point* (cm)	Posisi Aktual (cm)	*Error* (cm)	*Error* (%)
1,00	0,90	0,10	10%
2,00	1,80	0,20	10%
3,00	2,65	0,35	12%
4,00	3,55	0,45	11%
5,00	4,50	0,50	10%
Rata Persentase *Error*			11%

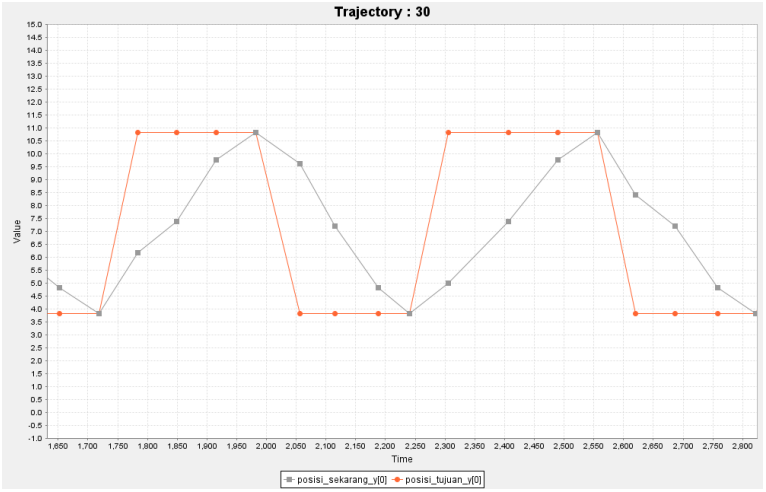
Pada saat uji coba, nilai maksimal dari posisi default adalah sejauh 5cm, apabila kaki digerakkan lebih dari 5cm dari posisi default maka pergerakan kaki menjadi kacau karena diluar jangkauan kaki. Dari data yang didapatkan persentase *error* posisi kaki dari perhitungan *inverse kinematics* adalah 11%. Nilai *error* ini kemungkinan disebabkan karena nilai tiap-tiap servo pada posisi *default* kaki robot tidak sepenuhnya tepat karena saat menentukan posisi *default* tidak ada alat ukur yang bisa digunakan maka hanya dengan menggunakan visual manusia. Nilai *error* ini juga bisa disebabkan karena faktor mekanik pengukuran panjang tiap lengan yang kurang sesuai. Selain dari dua faktor diatas, nilai *error* juga bisa disebabkan kurangnya kepresisian saat pengukuran.

4.2. Pengujian Trayektori

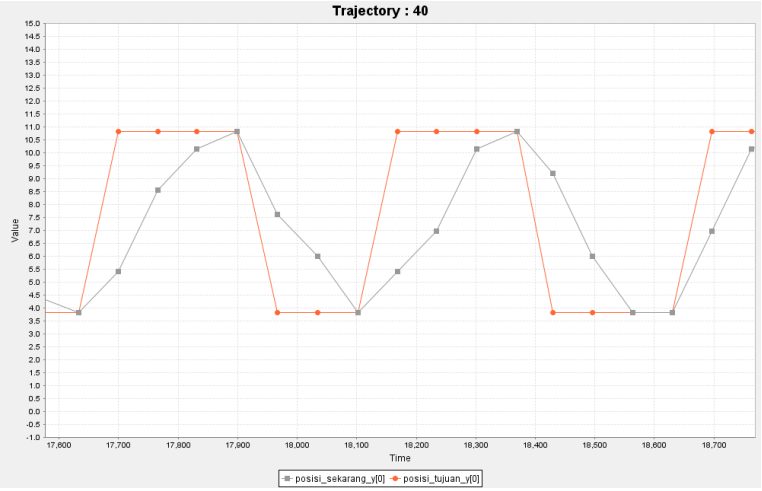
Pengujian trayektori merupakan sebuah langkah untuk menguji algoritma trayektori yang diterapkan pada robot. Algoritma trayektori yang diterapkan pada robot adalah trayektori berbasis kecepatan. Sehingga penentuan kecepatan jalan robot *quadruped* ini terdapat pada algoritma trayektori. Pada perngujian ini kecepatan trayektori yang diuji coba adalah dari 20 cm/s hingga 70 cm/s dengan selisih 10 cm/s. Pada grafik, variabel yang diuji adalah garis merah merupakan posisi tujuan salah satu kaki pada sumbu x sedangan garis abu-abu merupakan *update* posisi sekarang satu kaki pada sumbu x. Waktu sampling pada algoritma trayektori adalah 0,04 s. Waktu sampling pada grafik pengujian adalah 20 ms.



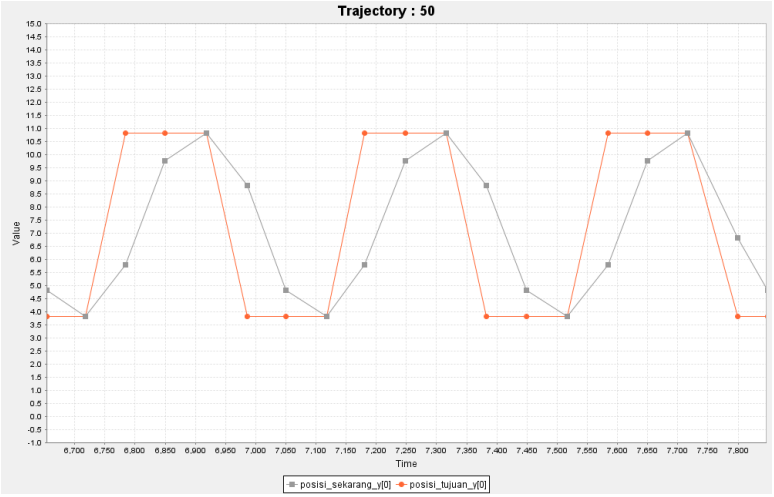
Gambar 4. 2 Grafik Kecepatan Trayektori 20



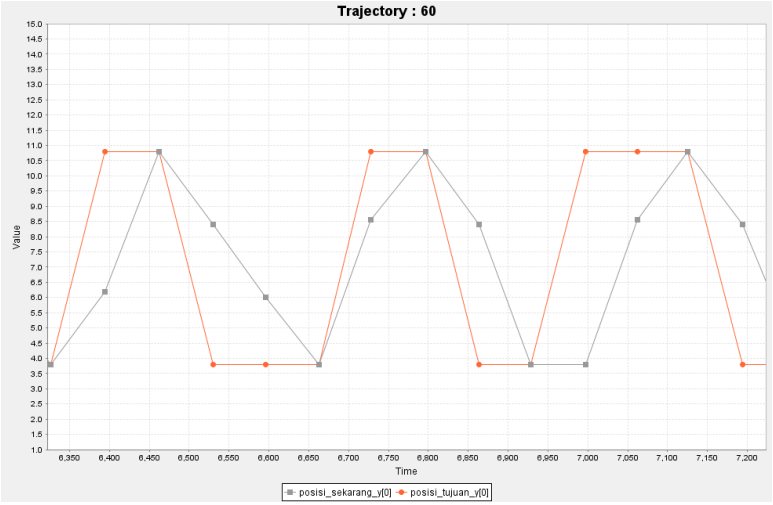
Gambar 4. 3 Grafik Kecepatan Trayektori 30



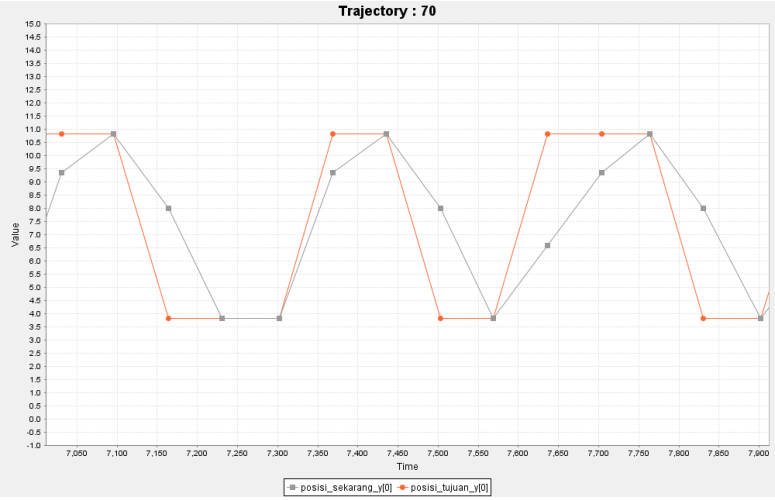
Gambar 4. 4 Grafik Kecepatan Trayektori 40



Gambar 4. 5 Grafik Kecepatan Trayektori 50



Gambar 4. 6 Grafik Kecepatan Trayektori 60



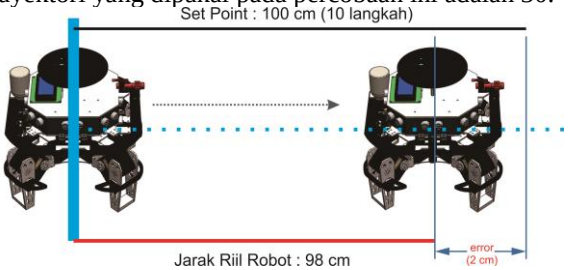
Gambar 4. 7 Grafik Kecepatan Trayektori 70

Dari Gambar 4. 2 hingga Gambar 4. 7 yang didapatkan diatas maka dapat disimpulkan semakin tinggi nilai kecepatan trayektori maka semakin cepat perubahan nilai posisi kaki robot. Semakin cepat kecepatan

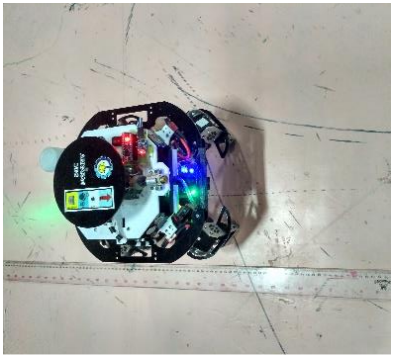
trayektori akan membuat kecepatan jalan robot semakin tinggi akan tetapi pergerakan *end-effector* tiap kaki akan semakin tidak linier dari posisi awal terhadap posisi tujuan.

4.3. Pengukuran Jarak Gerak Maju

Pengukuran jarak gerak maju adalah sebuah metode untuk mengetahui tingkat kepresisian jarak yang ditempuh robot secara aktual terhadap nilai koordinat yang diberikan. Pada pengujian ini dilakukan empat jarak pengujian yaitu 40cm, 60cm, 80cm, dan 100cm dengan masing-masing dilakukan lima percobaan kemudian nilai *error* yang didapatkan dirata-rata. Lebar langkah di atur pada jarak 2,5cm sehingga dalam satu langkah *(cycle)* jalan robot dapat menempuh jarak 10cm. Pada pengujian jarak 40cm digunakan 4 langkah, 60cm digunakan 6 langkah, 80cm digunakan 8 langkah, dan pada jarak 100cm digunakan 10 langkah. Kecepatan trayektori yang dipakai pada percobaan ini adalah 30.



Gambar 4. 8 Ilustrasi Pengukuran Jarak Gerak Maju



Gambar 4. 9 Pengujian Jarak Gerak Maju

Tabel 4. 2 Hasil Pengujian Gerak Maju 40cm (4 Langkah)

Percobaan Ke-	Hasil (cm)	*Error* (cm)
1	36,0	4
2	36,0	4
3	36,0	4
4	36,5	3,5
5	36,5	3,5
Rata *Error*		3,8

Tabel 4. 3 Hasil Pengujian Gerak Maju 60cm (6 Langkah)

Percobaan Ke-	Hasil (cm)	*Error* (cm)
1	57,0	3,0
2	57,0	3,0
3	58,0	2,0
4	57,5	2,5
5	57,7	2,3
Rata *Error*		2,6

Tabel 4. 4 Hasil Pengujian Gerak Maju 80cm (8 Langkah)

Percobaan Ke-	Hasil (cm)	*Error* (cm)
1	78,0	2,0
2	77,3	2,7
3	77,5	2,5
4	76,8	3,2
5	77,7	2,3
Rata *Error*		2,5

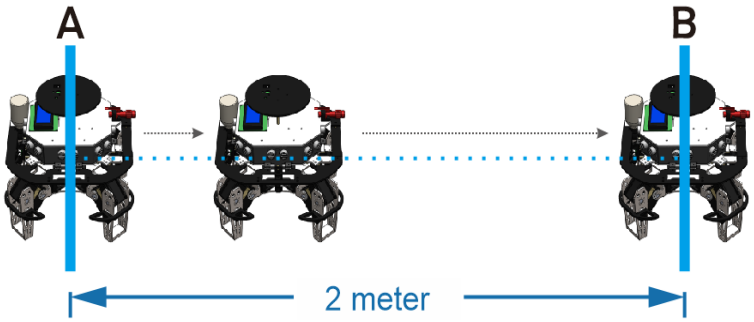
Tabel 4. 5 Hasil Pengujian Gerak Maju 100cm (10 Langkah)

Percobaan Ke-	Hasil (cm)	*Error* (cm)
1	98,0	2,0
2	98,8	1,2
3	98,5	1,5
4	97,9	2,1
5	98,4	1,6
Rata *Error*		1,7

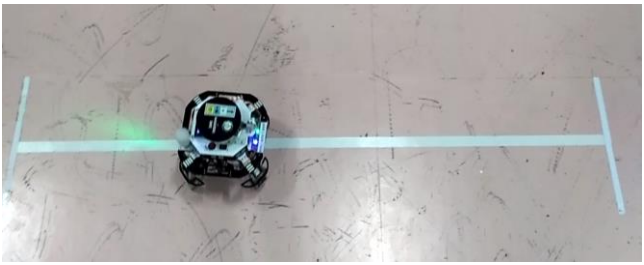
Dari data hasil percobaan yang telah dilakukan didapatkan rata-rata nilai *error* pada gerak maju 40 cm adalah 3,8 cm, pada 60 cm adalah 2,6 cm, pada 80 cm adalah 2,5 cm, dan pada 100 cm adalah 1,7 cm. Pada percobaan ini dapat disimpulkan bahwa semakin jauh jarak jalan robot maka semakin kecil nilai *error*, hal ini disebabkan karena kurang kepresisiannya kinematika robot yang saat ini masih terdapat *error* posisi sehingga mempengaruhi performa jarak jalan robot.

4.4. Pengukuran Kecepatan Robot dengan Variabel Kecepatan Trayektori

Pada pengujian yang pertama ini dilakukan pengukuran kecepatan riil robot dengan variabel jarak pada trayektori. Pengukuran dilakukan dengan cara mengukur waktu yang ditempuh robot dalam jarak dua meter dengan menggunakan *stopwatch* yang dioperasikan oleh manusia. Setelah didapatkan waktu tempuh, maka dapat di rumuskan kecepatan yaitu kecepatan = jarak/waktu. Pada pengujian ini digunakan lebar langkah 2,8 cm.



Gambar 4. 10 Ilustrasi Pengukuran Kecepatan Robot dengan Variabel Kecepatan Trayektori

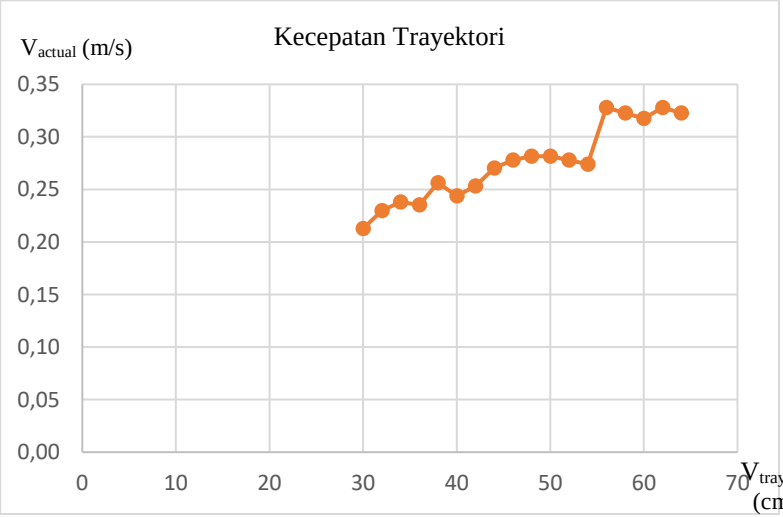


Gambar 4. 11 Pengukuran Kecepatan Robot dengan Variabel Jarak Trayektori

Tabel 4. 6 Kecepatan Robot dengan Variabel Jarak Trayektori

No	Kecepatan Trayektori (cm/s)	Waktu (s)	Kecepatan (m/s)
1	30	9,4	0,21
2	32	8,7	0,23
3	34	8,4	0,24
4	36	8,5	0,24
5	38	7,8	0,26
6	40	8,2	0,24

7	42	7,9	0,25
8	44	7,4	0,27
9	46	7,2	0,28
10	48	7,1	0,28
11	50	7,1	0,28
12	52	7,2	0,28
13	54	7,3	0,27
14	56	6,1	0,33
15	58	6,2	0,32
16	60	6,3	0,32
17	62	6,1	0,33
18	64	6,2	0,32

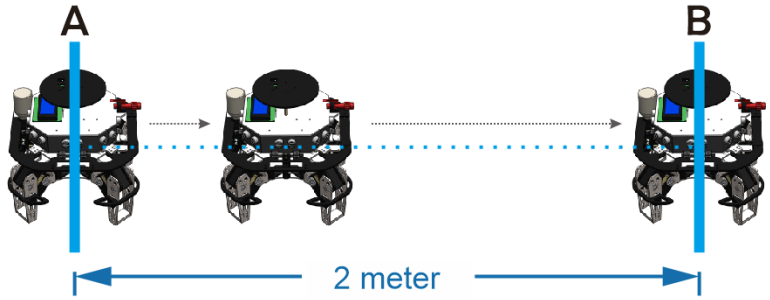


Gambar 4. 12 Grafik Kecepatan Trayektori Terhadap Kecepatan Riil Robot

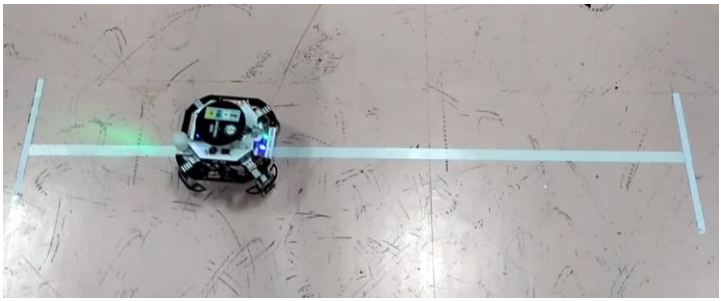
Dari data yang didapatkan dari pengujian kecepatan trayektori robot terhadap kecepatan robot dapat disimpulkan bahwa semakin tinggi nilai kecepatan trayektori maka semakin tinggi pula kecepatan robot. Namun kecepatan robot pada kecepatan trayektori diatas 54 sudah tidak lagi mengalami kenaikan hal tersebut dikarenakan *update* posisi trayektori sudah lebih besar dari nilai perpindahan dari posisi tujuan dengan posisi awal sehingga posisi tujuan dengan posisi sekarang sama. Hal ini juga disebabkan karena kecepatan servo RX-28 sudah maksimal dan mampu lagi untuk melakukan gerakan yang diperintahkan mikrokontroller. Data yang didapatkan memungkinkan terdapat ketidakpresisian waktu tempuh dikarenakan dioperasikan dengan cara menual menggunakan *stopwatch*.

4.5. Pengukuran Kecepatan Robot dengan Variabel Lebar Langkah

Pengujian yang kedua adalah pengukuran kecepatan robot dengan variabel lebar langkah. Pada tahap ini lebar langkah akan menentukan kecepatan robot dikarenakan tiap lebar langkah menentukan keefektifan pada saat berjalan. Pengukuran dilakukan dengan cara mengukur waktu yang ditempuh robot dalam jarak dua meter dengan menggunakan *stopwatch* yang dioperasikan oleh manusia. Setelah didapatkan waktu tempuh, maka dapat di rumuskan kecepatan yaitu kecepatan = jarak/waktu. Pada pengujian ini digunakan kecepatan trayektori 40 cm/s.



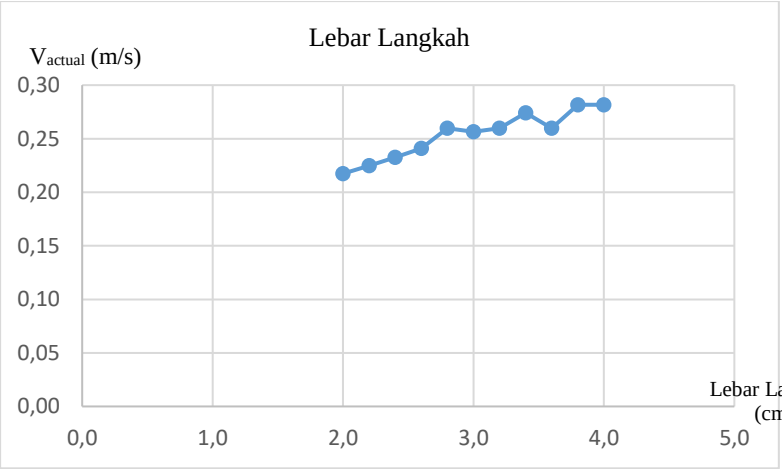
Gambar 4. 13 Ilustrasi Pengukuran Kecepatan Robot Dengan Variabel Lebar Langkah



Gambar 4. 14 Pengukuran Kecepatan Robot Dengan Variabel Lebar Langkah

Tabel 4. 7 Kecepatan Robot dengan Variabel Lebar Langkah

No	Lebar Langkah (cm)	Waktu (s)	Kecepatan (m/s)
1	2,0	9,2	0,22
2	2,2	8,9	0,22
3	2,4	8,6	0,23
4	2,6	8,3	0,24
5	2,8	7,7	0,26
6	3,0	7,8	0,26
7	3,2	7,7	0,26
8	3,4	7,3	0,27
9	3,6	7,7	0,26
10	3,8	7,1	0,28
11	4,0	7,1	0,28

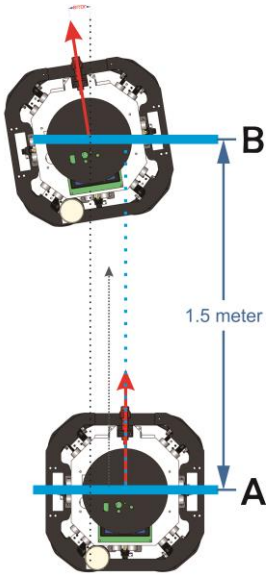


Gambar 4. 15 Grafik Lebar Langkah Terhadap Kecepatan Robot

Dari data yang didapatkan dari pengujian lebar langkah jalan robot terhadap kecepatan robot dapat disimpulkan bahwa semakin tinggi lebar langkah robot maka semakin tinggi pula kecepatan robot. Data yang didapatkan memungkinkan terdapat ketidakpresisian waktu tempuh dikarenakan dioperasikan dengan cara menual menggunakan *stopwatch*.

4.6. Pengujian Performa Robot Berjalan Lurus

Pada tahap ini dilakukan pengujian terhadap performa robot dalam berjalan lurus. Parameter yang diperhatikan disini adalah kemiringan robot dari posisi start dengan posisi finish dengan jarak tempuh 1.5m dengan menggunakan sensor *gyro accelero* mpu6050 yang mampu mengukur perubahan sudut putar. Variabel yang diubah pada pengujian ini adalah kecepatan trayektori. Dengan jarak langkah adalah 2,8cm.



Gambar 4. 16 Ilustrasi Pengujian Performa Robot Berjalan Lurus

Tabel 4. 8 Data Kemiringan Robot dengan Variabel Jarak Trayektori

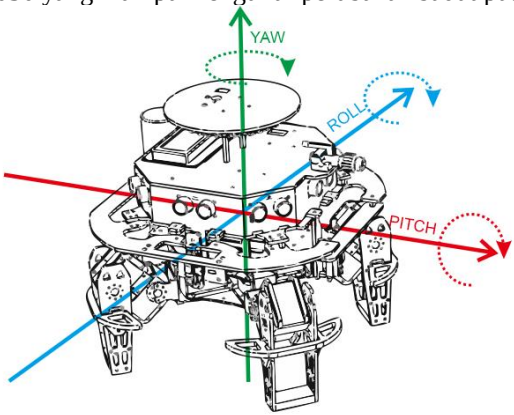
No	Kecepatan Trayektori	Kemiringan (°)
1	30	3
2	32	5
3	34	11
4	36	15
5	38	19
6	40	19
7	42	10
8	44	17
9	46	16
10	48	12
11	50	13

12	52	8
13	54	13
14	56	4
15	58	11
16	60	12
17	62	2
18	64	10

Dari data yang didapatkan tersebut maka nilai rata-rata kesalahan sudut belok pada gerak jalan lurus robot adalah sebesar 11,1° belok ke arah kiri pada jarak tempuh 1,5m. Hal ini disebabkan karena kurang tepatnya posisi *default* robot dikarenakan telah dilakukan banyak penggantian servo sehingga posisi *default* robot masih perlu untuk dikalibrasi ulang.

4.7. Pengujian Sudut *Yaw, Pitch, dan Roll*

Pada tahap ini dilakukan pengujian performa robot dalam melakukan pergerakan di tempat yaitu *yaw, pitch, dan roll*. Parameter yang diuji adalah kesalahan sudut masukan terhadap sudut aktual robot. Untuk menghitung sudut aktual pada robot digunakan sensor *gyro accelero* mpu6050 yang mampu mengukur perubahan sudut putar.



Gambar 4. 17 Ilustrasi *Yaw, Pitch,* Dan *Roll.*

Tabel 4. 9 Data Sudut *Yaw*

Yaw			
No	Sudut Masukan (°)	Sudut Aktual (°)	*Error* (°)
1	-20	-19,12	0,88
2	-16	-15,00	1,00
3	-12	-11,34	0,66
4	-8	-7,65	0,35
5	-4	-3,75	0,25
6	0	0,00	0,00
7	4	3,80	0,20
8	8	7,72	0,28
9	12	11,60	0,40
10	16	15,60	0,40
11	20	19,37	0,63
Rata *Error*			0,46

Tabel 4. 10 Data Sudut *Pitch*

Pitch			
No	Sudut Masukan (°)	Sudut Aktual (°)	*Error* (°)
1	-8	-6,41	1,59
2	-6	-5,20	0,80
3	-4	-3,87	0,13
4	-2	-1,79	0,21
5	0	0,00	0,00
6	2	1,44	0,56
7	4	3,01	0,99
8	6	5,17	0,83

9	8	6,11	1,89
Rata *Error*			0,78

Tabel 4. 11 Data Sudut *Roll*

Roll			
No	Sudut Masukan (°)	Sudut Aktual (°)	*Error* (°)
1	-8	-7,40	0,60
2	-6	-5,30	0,70
3	-4	-3,20	0,80
4	-2	-1,50	0,50
5	0	0,00	0,00
6	2	1,33	0,67
7	4	3,20	0,80
8	6	5,20	0,80
9	8	7,20	0,80
Rata *Error*			0,63

Dari data yang telah didapatkan, rata-rata nilai *error* dari pergerakan *yaw* adalah 0,46°, pergerakan *pitch* adalah 0,78°, dan pergerakan *roll* adalah 0,63°. Kesalahan sudut terkecil adalah pada pergerakan *yaw*. Hal ini disebabkan karena kurang tepatnya perhitungan dan pembuatan mekanik sehingga dapat memunculkan terjadinya kesalahan sudut pada pergerakan di tempat ini. Serta perlunya dilakukan kalibrasi ulang terhadap tiap servo setiap dilakukan pergantian servo.

Halaman ini sengaja dikosongkan

BAB V
PENUTUP

5.1. Kesimpulan

Kesimpulan yang bisa diambil dari pelaksanaan tugas akhir ini adalah :

1. Dengan menggunakan rumus *inverse kinematics* dan transformasi geometri (*translasi* dan *rotasi*) yang disertai dengan trayektori berbasis kecepatan mampu mewujudkan pergerakan pada robot *quadruped* dalam melakukan gerakan berjalan yaitu jalan lurus, belok, serong, putar di tempat, serta gerakan di tempat yaitu *roll*, *pitch*, dan *yaw*.
2. Kecepatan maksimal robot dalam berjalan lurus yang mampu ditempuh robot yaitu 0,33 m/s.
3. Tingkat rata-rata kesalahan posisi pada gerak satu kaki robot adalah 11%.
4. Tingkat kesalahan rata-rata jarak langkah robot kurang dari 0,9% (pada jarak 40 cm : 3,8 cm, pada 60 cm : 2,6 cm, pada 80 cm : 2,5cm, dan pada 100 cm: 1,7 cm).
5. Tingkat kesalahan rata-rata sudut belok pada gerak jalan lurus robot sejauh 1,5m adalah sebesar 11,1° belok ke arah kiri.
6. Tingkat kesalahan rata-rata sudut dari pergerakan ditempat adalah 0,62° (*yaw* : 0,46°, *pitch* : 0,78°, dan *roll* : 0,63°).

5.2. Saran

Saran yang dapat penulis sampaikan pada tugas akhir ini adalah perlu adanya teknik kalibrasi yang lebih presisi pada *setting* posisi *default* masing-masing kaki robot sehingga dapat meminimalisasi terjadinya *error* posisi pada pergerakan *end-effector* kaki robot.

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

[1] Ristekdikti, "Kontes Robot Pemadam Api Indonesia (KRPAI) - 2017", Jakarta: Direktorat Kemahasiswaan , 2017.

[2] T. College, "Trinity College Fire-Fighting Home Robot Contest 2017 Rules V1.0", Trinity College, 2016.

[3] D. Purwanto, "Bahan Kuliah Robot Industri : Kinematika Robot", Surabaya.

[4] D. Purwanto, "Bahan Kuliah Robot Industri : Perencanaan Gerak Robot", Surabaya.

[5] D. Y. Habibi, "Penerapan *Inverse Kinematics* Pada Pengendalian Gerak Robot," *Publikasi Jurnal Skripsi,* 2012.

[6] O. Sorin dan N. Mircea, "The Modeling of the Hexapod Mobile Robot Leg and Associated Interpolated Movements While Stepping," *System Theory, Control and Computing (ICSTCC), 2012 16th International Conference on,* 2012.

[7] "Rumus Matematika," 23 September 2013. http://rumus-matematika.com/lebih-mengenal-transformasi-geometri/. Diakses 12 Januari 2017.

[8] STMicroelectronics, UM1472 User manual Discovery kit for STM32F407/417 lines, 2014.

[9] Robotis, "User's Manual Dynamixel Rx-28", Robotis Co.,Ltd.

[10] E. Prasetya, "Implementasi *Inverse Kinematics* Pada Pergerakan Mobile Robot KRPAI Divisi Berkaki," *Publikasi Jurnal Skripsi,* 2014.

[11] S. Cubero, "Industrial Robotics Theory, Modelling and Control", 2007.

Halaman ini sengaja dikosongkan

LAMPIRAN

Listing Program

```
void gerak(double x,double y,double w, double z, double speed)
{
	static int step = 0;

	setSpeed(speed);

	transformasiGerak(x,y,w);

	if(!statusGerak[0] && !statusGerak[1] && !statusGerak[2] &&
!statusGerak[3])
	{
		switch(step)
		{
		case 0:
			setPergeseranKaki(0,walkX1[0],walkY1[0],z);
			setPergeseranKaki(2,walkX1[2],walkY1[2],z);
			setPergeseranKaki(1,walkX2[1],walkY2[1],0);
			setPergeseranKaki(3,walkX2[3],walkY2[3],0);
			break;
		case 1:
			setPergeseranKaki(0,walkX1[0],walkY1[0],0);
			setPergeseranKaki(2,walkX1[2],walkY1[2],0);
			setPergeseranKaki(1,walkX2[1],walkY2[1],0);
			setPergeseranKaki(3,walkX2[3],walkY2[3],0);

			break;
		case 2:
			setPergeseranKaki(0,walkX2[0],walkY2[0],0);
			setPergeseranKaki(2,walkX2[2],walkY2[2],0);
			setPergeseranKaki(1,walkX1[1],walkY1[1],z);
			setPergeseranKaki(3,walkX1[3],walkY1[3],z);
			break;
		case 3:
			setPergeseranKaki(0,walkX2[0],walkY2[0],0);
			setPergeseranKaki(2,walkX2[2],walkY2[2],0);
			setPergeseranKaki(1,walkX1[1],walkY1[1],0);
			setPergeseranKaki(3,walkX1[3],walkY1[3],0);
			break;
		}

		step++;
		if(step>3){step=0;hitung_langkah++;hitung_zigzag++;}
	}
}


void gerakStatic(double x,double y, double z,double yaw, double
pitch, double roll,double speed)
{
	int i;

	setSpeed(speed);
```

```
        yaw = yaw *PI/180;
        pitch = pitch *PI/180;
        roll = roll *PI/180;

        for(i=0;i<jumlahKaki;i++)
        {
                offset_x_awal[i]=koordinat_awal_x[i];
                offset_y_awal[i]=koordinat_awal_y[i];
                offset_z_awal[i]=koordinat_awal_z[i];

                //transformasi rotasi yaw
                if (i==0)
                {
                        //YAW
                        offset_x_awal[i] = (offset_x_awal[i]-(-
center_a)) * cos(yaw) - (offset_y_awal[i]-(-center_b)) * sin(yaw) +
(-center_a);
                        offset_y_awal[i] = (offset_x_awal[i]-(-
center_a)) * sin(yaw) + (offset_y_awal[i]-(-center_b)) * cos(yaw) +
(-center_b);

                        //PITCH
                        offset_x_awal[i] = (offset_x_awal[i]-(-
center_a)) * cos(pitch) - (offset_z_awal[i] - koordinat_awal_z[i]) *
sin(pitch) + (-center_a);
                        offset_z_awal[i] = (offset_x_awal[i]-(-
center_a)) * sin(pitch) + (offset_z_awal[i] - koordinat_awal_z[i]) *
cos(pitch) + koordinat_awal_z[i];

                        //ROLL
                        offset_y_awal[i] = (offset_y_awal[i]-(-
center_a)) * cos(roll) - (offset_z_awal[i] - koordinat_awal_z[i]) *
sin(roll) + (-center_a);
                        offset_z_awal[i] = (offset_y_awal[i]-(-
center_a)) * sin(roll) + (offset_z_awal[i] - koordinat_awal_z[i]) *
cos(roll) + koordinat_awal_z[i];
                }
                else if (i==1)
                {
                        //YAW
                        offset_x_awal[i] = (offset_x_awal[i]-(-
center_a)) * cos(yaw) - (offset_y_awal[i]-(center_b)) * sin(yaw) + (-
center_a);
                        offset_y_awal[i] = (offset_x_awal[i]-(-
center_a)) * sin(yaw) + (offset_y_awal[i]-(center_b)) * cos(yaw) +
(center_b);

                        //PITCH
                        offset_x_awal[i] = (offset_x_awal[i]-(-
center_a)) * cos(pitch) - (offset_z_awal[i] - koordinat_awal_z[i]) *
sin(pitch) + (-center_a);
                        offset_z_awal[i] = (offset_x_awal[i]-(-
center_a)) * sin(pitch) + (offset_z_awal[i] - koordinat_awal_z[i]) *
cos(pitch) + koordinat_awal_z[i];

                        //ROLL
```

```
                        offset_y_awal[i] = (offset_y_awal[i]-
(center_a)) * cos(roll) - (offset_z_awal[i] - koordinat_awal_z[i]) *
sin(roll) + (center_a);
                        offset_z_awal[i] = (offset_y_awal[i]-
(center_a)) * sin(roll) + (offset_z_awal[i] - koordinat_awal_z[i]) *
cos(roll) + koordinat_awal_z[i];
                }
                else if (i==2)
                {
                        //YAW
                        offset_x_awal[i] = (offset_x_awal[i]-
(center_a)) * cos(yaw) - (offset_y_awal[i]-(center_b)) * sin(yaw) +
(center_a);
                        offset_y_awal[i] = (offset_x_awal[i]-
(center_a)) * sin(yaw) + (offset_y_awal[i]-(center_b)) * cos(yaw) +
(center_b);

                        //PITCH
                        offset_x_awal[i] = (offset_x_awal[i]-
(center_a)) * cos(pitch) - (offset_z_awal[i] - koordinat_awal_z[i]) *
sin(pitch) + (center_a);
                        offset_z_awal[i] = (offset_x_awal[i]-
(center_a)) * sin(pitch) + (offset_z_awal[i] - koordinat_awal_z[i]) *
cos(pitch) + koordinat_awal_z[i];

                        //ROLL
                        offset_y_awal[i] = (offset_y_awal[i]-
(center_a)) * cos(roll) - (offset_z_awal[i] - koordinat_awal_z[i]) *
sin(roll) + (center_a);
                        offset_z_awal[i] = (offset_y_awal[i]-
(center_a)) * sin(roll) + (offset_z_awal[i] - koordinat_awal_z[i]) *
cos(roll) + koordinat_awal_z[i];
                }
                else if (i==3)
                {
                        //YAW
                        offset_x_awal[i] = (offset_x_awal[i]-
(center_a)) * cos(yaw) - (offset_y_awal[i]-(-center_b)) * sin(yaw) +
(center_a);
                        offset_y_awal[i] = (offset_x_awal[i]-
(center_a)) * sin(yaw) + (offset_y_awal[i]-(-center_b)) * cos(yaw) +
(-center_b);

                        //PITCH
                        offset_x_awal[i] = (offset_x_awal[i]-
(center_a)) * cos(pitch) - (offset_z_awal[i] - koordinat_awal_z[i]) *
sin(pitch) + (center_a);
                        offset_z_awal[i] = (offset_x_awal[i]-
(center_a)) * sin(pitch) + (offset_z_awal[i] - koordinat_awal_z[i]) *
cos(pitch) + koordinat_awal_z[i];

                        //ROLL
                        offset_y_awal[i] = (offset_y_awal[i]-(-
center_a)) * cos(roll) - (offset_z_awal[i] - koordinat_awal_z[i]) *
sin(roll) + (-center_a);
```

```
                        offset_z_awal[i] = (offset_y_awal[i]-(-
center_a)) * sin(roll) + (offset_z_awal[i] - koordinat_awal_z[i]) *
cos(roll) + koordinat_awal_z[i];
                }

                //transformasi linear
                offset_x_awal[i]+=x;
                offset_y_awal[i]+=y;
                offset_z_awal[i]+=z;

                //jadikan pergeseran
                offset_x_awal[i] -=koordinat_awal_x[i];
                offset_y_awal[i] -=koordinat_awal_y[i];
                offset_z_awal[i] -=koordinat_awal_z[i];

                setPergeseranKaki(i,0,0,0);
        }
}

void transformasiGerak(double x,double y,double w)
{
        int i;

        w = w *PI/180;

        for(i=0;i<jumlahKaki;i++)
        {
                //transformasi rotasi
                if (i==0)
                {
        walkX1[i] = (koordinat_awal_x[i]-(-center_a)) * cos(w) -
(koordinat_awal_y[i]-(-center_b)) * sin(w) + (-center_a);
        walkX2[i] = (koordinat_awal_x[i]-(-center_a)) * cos(-w) -
(koordinat_awal_y[i]-(-center_b)) * sin(-w) + (-center_a);
        walkY1[i] = (koordinat_awal_x[i]-(-center_a)) * sin(w) +
(koordinat_awal_y[i]-(-center_b)) * cos(w) + (-center_b);
        walkY2[i] = (koordinat_awal_x[i]-(-center_a)) * sin(-w) +
(koordinat_awal_y[i]-(-center_b)) * cos(-w) + (-center_b);
                }
                else if (i==1)
                {
        walkX1[i] = (koordinat_awal_x[i]-(-center_a)) * cos(w) -
(koordinat_awal_y[i]-(center_b)) * sin(w) + (-center_a);
        walkX2[i] = (koordinat_awal_x[i]-(-center_a)) * cos(-w) -
(koordinat_awal_y[i]-(center_b)) * sin(-w) + (-center_a);
        walkY1[i] = (koordinat_awal_x[i]-(-center_a)) * sin(w) +
(koordinat_awal_y[i]-(center_b)) * cos(w) + (center_b);
        walkY2[i] = (koordinat_awal_x[i]-(-center_a)) * sin(-w) +
(koordinat_awal_y[i]-(center_b)) * cos(-w) + (center_b);
                }
                else if (i==2)
                {
        walkX1[i] = (koordinat_awal_x[i]-(center_a)) * cos(w) -
(koordinat_awal_y[i]-(center_b)) * sin(w) + (center_a);
        walkX2[i] = (koordinat_awal_x[i]-(center_a)) * cos(-w) -
(koordinat_awal_y[i]-(center_b)) * sin(-w) + (center_a);
```

```
        walkY1[i] = (koordinat_awal_x[i]-(center_a)) * sin(w) +
(koordinat_awal_y[i]-(center_b)) * cos(w) + (center_b);
        walkY2[i] = (koordinat_awal_x[i]-(center_a)) * sin(-w) +
(koordinat_awal_y[i]-(center_b)) * cos(-w) + (center_b);
                }
                else if (i==3)
                {
        walkX1[i] = (koordinat_awal_x[i]-(center_a)) * cos(w) -
(koordinat_awal_y[i]-(-center_b)) * sin(w) + (center_a);
        walkX2[i] = (koordinat_awal_x[i]-(center_a)) * cos(-w) -
(koordinat_awal_y[i]-(-center_b)) * sin(-w) + (center_a);
        walkY1[i] = (koordinat_awal_x[i]-(center_a)) * sin(w) +
(koordinat_awal_y[i]-(-center_b)) * cos(w) + (-center_b);
        walkY2[i] = (koordinat_awal_x[i]-(center_a)) * sin(-w) +
(koordinat_awal_y[i]-(-center_b)) * cos(-w) + (-center_b);
                }


                //transformasi linear
                walkX1[i] +=x;
                walkX2[i] -=x;
                walkY1[i] +=y;
                walkY2[i] -=y;

                //jadikan pergeseran
                walkX1[i] -=koordinat_awal_x[i];
                walkX2[i] -=koordinat_awal_x[i];
                walkY1[i] -=koordinat_awal_y[i];
                walkY2[i] -=koordinat_awal_y[i];
        }
}

void setPergeseranKaki(int noKaki,double x, double y, double z)
{
        statusGerak[noKaki]=1;

        posisi_tujuan_x[noKaki] = koordinat_awal_x[noKaki] + x +
offset_x_awal[noKaki];
        posisi_tujuan_y[noKaki] = koordinat_awal_y[noKaki] + y +
offset_y_awal[noKaki];
        posisi_tujuan_z[noKaki] = koordinat_awal_z[noKaki] + z +
offset_z_awal[noKaki];
}

void TIM6_DAC_IRQHandler()
{
        trajectoriLinier(speedServo);
        kirimInstruksiGerak(0);

        //      GPIO_ToggleBits(GPIOD,GPIO_Pin_12);
        if (penanda_buzzer==0)(GPIO_SetBits(GPIOE,GPIO_Pin_4));
        else if
(penanda_buzzer>=5)(GPIO_ResetBits(GPIOE,GPIO_Pin_4));
        penanda_ganti_muka++;
        pewaktu++;
        penanda_buzzer++;
```

```
        TIM_ClearFlag(TIM6,TIM_FLAG_Update);
}

void trajectoriLinier(double speed)
{
        static double delta_x[jumlahKaki];
        static double delta_y[jumlahKaki];
        static double delta_z[jumlahKaki];

        static double speed_x[jumlahKaki];
        static double speed_y[jumlahKaki];
        static double speed_z[jumlahKaki];

        static double s[jumlahKaki];

        double ts = TIME_SAMPLING;

        int i;

        for(i=0;i<jumlahKaki;i++)
        {
                delta_x[i] = posisi_tujuan_x[i] -
posisi_sekarang_x[i];
                delta_y[i] = posisi_tujuan_y[i] -
posisi_sekarang_y[i];
                delta_z[i] = posisi_tujuan_z[i] -
posisi_sekarang_z[i];

                s[i] = sqrt(delta_x[i]*delta_x[i] +
delta_y[i]*delta_y[i] + delta_z[i]*delta_z[i]);

                if(fabs(s[i])>fabs(speed*ts))
                {
                        speed_x[i] = speed*(delta_x[i]/s[i]);
                        speed_y[i] = speed*(delta_y[i]/s[i]);
                        speed_z[i] = speed*(delta_z[i]/s[i]);

                        posisi_sekarang_x[i] = speed_x[i]*ts +
posisi_sekarang_x[i];
                        posisi_sekarang_y[i] = speed_y[i]*ts +
posisi_sekarang_y[i];
                        posisi_sekarang_z[i] = speed_z[i]*ts +
posisi_sekarang_z[i];
                }

                else if(fabs(s[i])<=fabs(speed*ts) && fabs(s[i]>0))
                {
                        posisi_sekarang_x[i] = posisi_tujuan_x[i];
                        posisi_sekarang_y[i] = posisi_tujuan_y[i];
                        posisi_sekarang_z[i] = posisi_tujuan_z[i];

                        statusGerak[i]=0;
                }

                else if(fabs(s[i]==0))statusGerak[i]=0;
```

```
                pergeseranPerkaki(i,posisi_sekarang_x[i]-
koordinat_awal_x[i],

        posisi_sekarang_y[i]-koordinat_awal_y[i],

        posisi_sekarang_z[i]-koordinat_awal_z[i]);

        }
}

void kirimInstruksiGerak(int speed)
{
         unsigned char position_H = 0;
         unsigned char position_L = 0;
         unsigned char speed_H = 0;
         unsigned char speed_L = 0;
         unsigned char id;
         unsigned char panjangData = 4;
         unsigned char jumlahServo = jumlahKaki * servoPerkaki;
         unsigned char LENGTH = (panjangData + 1) * jumlahServo +
4;//19; // (panjang data +1) x jumlah servo + 4 ===>>> lighat manual
rx 28 halaman 37

//       tentukanKecepatanServo(speed);

         speed_H = speed>>8;  //high adresses
         speed_L = speed&0xff; //low addresses karena pengiriman
data harus 16bit tetapi dipecah menjadi 2 yaitu masing2 8bit

         checksum =0xFE + LENGTH + 0X83 + GOAL_POSITION_L +
panjangData;

         //############## HEADER ############//
         bufferDataTx[0]=0xFF;
         bufferDataTx[1]=0xFF;

         //######### BROADCAST INTRUCTION ############//
         bufferDataTx[2]=0xFE;

         //######### LENGTH ############//
         bufferDataTx[3]=LENGTH;

         //######### SYNC WRITE ############//
         bufferDataTx[4]=0x83;

         //######### FISRT ADDRESS ############//
         bufferDataTx[5]=GOAL_POSITION_L;

         //######### LENGTH OF DATA ############//
         bufferDataTx[6]=panjangData;

         //######### SERVO 00 ##################//
         position_H = outServo[0][0]>>8;
         position_L = outServo[0][0]&0xff;

         id =ID00;
```

```
checksum += position_L +position_H +speed_L +speed_H + id;

bufferDataTx[7] = id;
bufferDataTx[8] = position_L;
bufferDataTx[9] = position_H;
bufferDataTx[10] = speed_L;
bufferDataTx[11] = speed_H;

//######## SERVO 01 ##################//
position_H = outServo[0][1]>>8;
position_L = outServo[0][1]&0xff;

id =ID01;
checksum += position_L +position_H +speed_L +speed_H + id;

bufferDataTx[12] = id;
bufferDataTx[13] = position_L;
bufferDataTx[14] = position_H;
bufferDataTx[15] = speed_L;
bufferDataTx[16] = speed_H;

//######## SERVO 02 ##################//
position_H = outServo[0][2]>>8;
position_L = outServo[0][2]&0xff;

id =ID02;
checksum += position_L +position_H +speed_L +speed_H + id;

bufferDataTx[17] = id;
bufferDataTx[18] = position_L;
bufferDataTx[19] = position_H;
bufferDataTx[20] = speed_L;
bufferDataTx[21] = speed_H;

//######## SERVO 10 ##################//
position_H = outServo[1][0]>>8;
position_L = outServo[1][0]&0xff;

id =ID10;
checksum += position_L +position_H +speed_L +speed_H + id;

bufferDataTx[22] = id;
bufferDataTx[23] = position_L;
bufferDataTx[24] = position_H;
bufferDataTx[25] = speed_L;
bufferDataTx[26] = speed_H;

//######## SERVO 11 ##################//
position_H = outServo[1][1]>>8;
position_L = outServo[1][1]&0xff;

id =ID11;
checksum += position_L +position_H +speed_L +speed_H + id;

bufferDataTx[27] = id;
bufferDataTx[28] = position_L;
bufferDataTx[29] = position_H;
```

```
bufferDataTx[30] = speed_L;
bufferDataTx[31] = speed_H;

//######## SERVO 12 ##################//
position_H = outServo[1][2]>>8;
position_L = outServo[1][2]&0xff;

id =ID12;
checksum += position_L +position_H +speed_L +speed_H + id;

bufferDataTx[32] = id;
bufferDataTx[33] = position_L;
bufferDataTx[34] = position_H;
bufferDataTx[35] = speed_L;
bufferDataTx[36] = speed_H;

//######## SERVO 20 ##################//
position_H = outServo[2][0]>>8;
position_L = outServo[2][0]&0xff;

id =ID20;
checksum += position_L +position_H +speed_L +speed_H + id;

bufferDataTx[37] = id;
bufferDataTx[38] = position_L;
bufferDataTx[39] = position_H;
bufferDataTx[40] = speed_L;
bufferDataTx[41] = speed_H;

//######## SERVO 21 ##################//
position_H = outServo[2][1]>>8;
position_L = outServo[2][1]&0xff;

id =ID21;
checksum += position_L +position_H +speed_L +speed_H + id;

bufferDataTx[42] = id;
bufferDataTx[43] = position_L;
bufferDataTx[44] = position_H;
bufferDataTx[45] = speed_L;
bufferDataTx[46] = speed_H;

//######## SERVO 22 ##################//
position_H = outServo[2][2]>>8;
position_L = outServo[2][2]&0xff;

id =ID22;
checksum += position_L +position_H +speed_L +speed_H + id;

bufferDataTx[47] = id;
bufferDataTx[48] = position_L;
bufferDataTx[49] = position_H;
bufferDataTx[50] = speed_L;
bufferDataTx[51] = speed_H;


//######## SERVO 30 ##################//
```

```
        position_H = outServo[3][0]>>8;
        position_L = outServo[3][0]&0xff;

        id =ID30;
        checksum += position_L +position_H +speed_L +speed_H + id;

        bufferDataTx[52] = id;
        bufferDataTx[53] = position_L;
        bufferDataTx[54] = position_H;
        bufferDataTx[55] = speed_L;
        bufferDataTx[56] = speed_H;

        //######## SERVO 31 ##################//
        position_H = outServo[3][1]>>8;
        position_L = outServo[3][1]&0xff;

        id =ID31;
        checksum += position_L +position_H +speed_L +speed_H + id;

        bufferDataTx[57] = id;
        bufferDataTx[58] = position_L;
        bufferDataTx[59] = position_H;
        bufferDataTx[60] = speed_L;
        bufferDataTx[61] = speed_H;

        //######## SERVO 32 ##################//
        position_H = outServo[3][2]>>8;
        position_L = outServo[3][2]&0xff;

        id =ID32;
        checksum += position_L +position_H +speed_L +speed_H + id;

        bufferDataTx[62] = id;
        bufferDataTx[63] = position_L;
        bufferDataTx[64] = position_H;
        bufferDataTx[65] = speed_L;
        bufferDataTx[66] = speed_H;


        //############## checksum ##############//
        checksum =(~checksum) & 0xFF; //only use lower bytes hal.49
        bufferDataTx[67] = checksum;

        kirimData(67);
}

void kirimData(char jmlhData)
{
        if(jumlahData == 0)
        {
                jumlahData = jmlhData;
                USART_ITConfig(USART6, USART_IT_TXE, ENABLE);
        }
}

void USART6_IRQHandler(void)
```

```
{
        static int dataKe;

        if (USART_GetITStatus(USART6, USART_IT_TXE) != RESET)
        {
                USART_SendData(USART6,bufferDataTx[dataKe]);

                dataKe++;

                if(dataKe == jumlahData + 1)
                {
                        jumlahData = 0;
                        dataKe = 0;
                        USART_ITConfig(USART6, USART_IT_TXE,
DISABLE);
                }
        }

}

void inisialisasiPosisiKaki()
{
        int i;

        USART_ITConfig(USART6, USART_IT_TXE, DISABLE);

        //Nilai Default Servo
        nilai_default_servo[0][0] = 650; //700 690
        nilai_default_servo[0][1] = 393; //455
        nilai_default_servo[0][2] = 427; //427

        nilai_default_servo[1][0] = 700; //700
        nilai_default_servo[1][1] = 415; //415
        nilai_default_servo[1][2] = 670; //670

        nilai_default_servo[2][0] = 690; //700
        nilai_default_servo[2][1] = 408; //405
        nilai_default_servo[2][2] = 350; //350

        nilai_default_servo[3][0] = 720; //720
        nilai_default_servo[3][1] = 470; //410 460
        nilai_default_servo[3][2] = 650; //650

        //Koordinat_awal (sebelum bergeser)
        koordinat_awal_x[0] = 7; //6.7
        koordinat_awal_y[0] = 7.3; //6.6
        koordinat_awal_z[0] = -9.3; //-9.3

        koordinat_awal_x[1] = 7; //6.4
        koordinat_awal_y[1] = -6; //-6.45
        koordinat_awal_z[1] = -9.3; //-9.3

        koordinat_awal_x[2] = -7; //-6.98
        koordinat_awal_y[2] = -6.5; //-6.68
        koordinat_awal_z[2] = -9.3; //-9.3
```

```
	koordinat_awal_x[3] = -6; //-6.8
	koordinat_awal_y[3] = 7; //7
	koordinat_awal_z[3] = -9.3; //-9.5

	for(i=0;i<jumlahKaki;i++)
	{
		offset_x_awal[i]=0;
		offset_y_awal[i]=0;
		offset_z_awal[i]=0;

		posisi_sekarang_x[i] = koordinat_awal_x[i];
		posisi_sekarang_y[i] = koordinat_awal_y[i];
		posisi_sekarang_z[i] = koordinat_awal_z[i];

		pergeseranPerkaki(i, 0, 0, 0);
	}
}
```

BIODATA PENULIS

Wahyu Tri Wibowo merupakan anak ketiga dari pasangan M. Bilal dan Sumarni yang berdomisili di Desa Sugihan, Kec. Tengaran, Kabupaten Semarang, Jawa Tengah. Lahir di Kab. Semarang pada 28 Januari 1995. Penulis menempuh pendidikan dasar di SDN 04 Sugihan Kec. Tengaran, pendidikan menengah di SMPN 1 Ampel, dan pendidikan menengah atas di SMAN 1 Salatiga. Saat ini penulis sedang menyelesaikan studi Strata-1 (S1) di Institut Teknologi Sepuluh Nopember (ITS) Surabaya pada Departemen Teknik Elektro, Fakulas Teknologi Elektro dengan bidang studi Elektronika. Selama kuliah, penulis aktif sebagai asisten di laboratorium Elektronika Dasar B202. Penulis juga aktif melakukan penelitian dan karya ilmiah di bidang energi terbarukan dengan mengikuti beberapa perlombaan di bidang *renewable energy*. Menjadi ketua divisi Workshop jurusan teknik elektro pada satu periode kepengurusan 2015/2016. Serta aktif di bidang robotika dengan menjadi perwakilan tim robot ITS di kategori Kontes Robot Pemadam Api Indonesia (KRPAI) kategori Berkaki pada tahun 2015-2017 (2 periode).

Email:
wahyu_tri.wibowo@yahoo.com

Halaman ini sengaja dikosongkan