TUGAS AKHIR – EE184801

RANCANG BANGUN PURWARUPA ROBOT BERKAKI ENAM DENGAN SENSOR JARAK UNTUK MELEWATI TANGGA LURUS

Mochamad Yusuf
NRP 07111640000090

Dosen Pembimbing
Dr.Ir.Djoko Purwanto M.Eng.
Ronny Mardiyanto, ST., MT., Ph.D.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro Dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

TUGAS AKHIR – EE 184801

RANCANG BANGUN PURWARUPA ROBOT BERKAKI ENAM DENGAN SENSOR JARAK UNTUK MELEWATI TANGGA LURUS

Mochamad Yusuf
NRP 07111640000090

Dosen Pembimbing
Dr.Ir.Djoko Purwanto M.Eng.
Ronny Mardiyanto, ST., MT., Ph.D.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro Dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

FINAL PROJECT – **EE 184801**

DESIGN OF SIX-LEGGED ROBOT PROTOTYPE WITH DISTANCE SENSOR TO PASS THE STRAIGHT STAIRS

Mochamad Yusuf
NRP 07111640000090

Supervisor
Dr.Ir.Djoko Purwanto M.Eng.
Ronny Mardiyanto, ST., MT., Ph.D.

DEPARTMENT OF ELECTRICAL ENGINEERING
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2020

PERNYATAAN KEASLIAN TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul :

"Rancang Bangun Purwarupa Robot Berkaki Enam dengan Sensor Jarak untuk Melewati Tangga Lurus."

adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diizinkan 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, 15 Juli 2020

Mochamad Yusuf
NRP. 07111640000090

Halaman ini sengaja dikosongkan

RANCANG BANGUN PURWARUPA ROBOT BERKAKI ENAM DENGAN SENSOR JARAK UNTUK MELEWATI TANGGA LURUS

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan Untuk
Memperoleh Gelar Sarjana Teknik Elektro

Pada

Bidang Studi Teknik Elektronika
Departemen Teknik Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui :

Dosen Pembimbing I

Dr.Ir.Djoko Purwanto M.Eng.
NIP. 196512111990021002

SURABAYA
JULI, 2020

Halaman ini sengaja dikosongkan

RANCANG BANGUN PURWARUPA ROBOT BERKAKI ENAM DENGAN SENSOR JARAK UNTUK MELEWATI TANGGA LURUS

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan Untuk
Memperoleh Gelar Sarjana Teknik Elektro

Pada

Bidang Studi Teknik Elektronika
Departemen Teknik Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui :

Dosen Pembimbing II

Ronny Mardiyanto, ST., MT., Ph.D.
NIP. 198101182003121003

SURABAYA
JULI, 2020

Halaman ini sengaja dikosongkan

RANCANG BANGUN PURWARUPA ROBOT BERKAKI ENAM DENGAN SENSOR JARAK UNTUK MELEWATI TANGGA LURUS

Mochamad Yusuf
07111640000090

Dosen Pembimbing : 1. Dr.Ir.Djoko Purwanto M.Eng.
: 2. Ronny Mardiyanto, ST., MT., Ph.D.

ABSTRAK

Robot berkaki enam merupakan robot yang menggunakan mekanisme gerak berupa kaki yang berjumlah enam. Penggunaan mekanisme gerak kaki membuat robot ini dapat lebih leluasa untuk bergerak di lingkungan yang kasar dengan rintangan-rintangan kecil. Akan tetapi, robot berkaki memiliki kesulitan saat medan yang dilewati bukan merupakan bidang datar seperti tangga. Oleh karena itu, dirancang algoritma untuk robot berkaki enam agar robot tersebut dapat berjalan melewati tangga buatan dengan stabil. Robot berkaki enam ini dilengkapi dengan sebuah sensor jarak mendeteksi tepian tangga. Informasi ini kemudian digunakan sebagai acuan untuk mengendalikan pergerakan pada robot. Algoritma tersebut diharapkan dapat memungkinkan robot berkaki enam yang telah dibuat melakukan perjalanan dari satu lantai ke lantai lainnya dengan memanfaatkan akses tangga yang tersedia. Pada tugas akhir ini, robot berkaki enam dapat melewati tangga dengan waktu 2 menit 49 detik sampai 3 menit 4 detik dengan jarak dari tangga dari 15 cm sampai dengan 40 cm. Pada jarak 45 cm dan 50 cm dari tangga robot tidak mampu menaiki tangga karena posisi robot tidak dapat lurus terhadap tangga. Hal ini membuktikan bahwa algoritma yang dibuat hanya berhasil pada saat posisi dari robot lurus terhadap tangga. Posisi robot yang tidak lurus terhadap tangga diharapkan dapat diatasi dengan memperbaiki cara berjalan dari robot di atas lantai sehingga robot dapat berjalan dengan lurus hingga mencapai tangga.

Kata kunci: Robot Berkaki, Robot Berkaki Enam, Kinematika Gerak Robot, Sensor Jarak

Halaman ini sengaja dikosongkan

DESIGN OF SIX-LEGGED ROBOT PROTOTYPE WITH DISTANCE SENSOR TO PASS THE STRAIGHT STAIRS

Mochamad Yusuf
07111640000090

Dosen Pembimbing : 1. Dr.Ir.Djoko Purwanto M.Eng.
: 2. Ronny Mardiyanto, ST., MT., Ph.D.

ABSTRACT

Six-legged robot is a robot that uses a mechanism of motion in the form of six feet. The use of footwork mechanism makes this robot more free to move in rough environments with small obstacles. However, legged robots have difficulty when the field being passed is not a flat plane like a stairs. Therefore, an algorithm is designed for six-legged robots so that the robot can walk steadily through artificial stairs. This six-legged robot is equipped with a proximity sensor to detect the edges of the stairs. This information is then used as a reference to control the movement of the robot. The algorithm is expected to enable six-legged robots that have been made to travel from one floor to another by utilizing available stair access. In this final project, the six-legged robot can pass the stairs with a time of 2 minutes 49 seconds to 3 minutes 4 seconds with a distance from the stairs from 15 cm to 40 cm. At a distance of 45 cm and 50 cm from the stair the robot is unable to climb the stairs because the robot's position cannot be straight against the stairs. This proves that the algorithm is only successful when the position of the robot is straight toward the stairs. The position of the robot that is not straight toward the stairs is expected to be overcome by improving the way of walking from the robot on the floor so that the robot can walk straight until it reaches the stairs.

Keywords: Legged Robot, Six Legged Robot, Robot Motion Kinematics, Distance Sensor

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji syukur saya panjatkan kepada Allah SWT karena hanya dengan rahmat dan hidayah-Nya Penulis dapat membuat dan menyelesaikan laporan tugas akhir ini dengan tepat waktu dan semestinya.

Tugas akhir merupakan salah satu mata kuliah yang wajib ditempuh di Departemen Teknik Elektro Institut Teknologi Sepuluh Nopember ini. Penelitian yang penulis lakukan mengambil topik robotika. Laporan tugas akhir ini disusun untuk melengkapi dan mendokumentasikan atau membukukan hasil capaian dari penelitian yang telah dilaksanakan.

Pengerjaan laporan tugas akhir ini bisa diselesaikan dengan bantuan pihak ketiga yang meliputi dosen pembimbing serta rekan-rekan sekalian yang telah senantiasa mendukung selama penelitian yang penulis kerjakan, sehingga penulis dengan tulus mengucapkan kasih yang sebesar-besarnya.

Penulis menyadari bahwa terdapat banyak kekurangan dari laporan tugas akhir ini, baik dari segi materi maupun teknis penyajiannya, mengingat masih kurangnya pengetahuan dan pengalaman penulis. Namun pengalaman dan wawasan baru yang penulis banyak dapatkan juga patut untuk penulis syukuri dan diterapkan menjadi ilmu yang manfaat.

Surabaya, 15 Juli 2020

Penulis

Halaman ini sengaja dikosongkan

DAFTAR ISI

PERNYATAAN KEASLIAN TUGAS AKHIR ..... **vii**
LEMBAR PENGESAHAN I ..... **ix**
LEMBAR PENGESAHAN II ..... **xi**
ABSTRAK ..... **xiii**
ABSTRACT ..... **xv**
KATA PENGANTAR ..... **xvii**
DAFTAR ISI ..... **xix**
DAFTAR GAMBAR ..... **xxi**
DAFTAR TABEL ..... **xxiii**
BAB I PENDAHULUAN **1**
1.1. Latar Belakang 1
1.2. Permasalahan 2
1.3. Tujuan 2
1.4. Batasan Masalah 2
1.5. Metodologi 2
1.6. Sistematika Penulisan 3
1.7. Relevansi 4
BAB II TINJAUAN PUSTAKA DAN TEORI PENUNJANG **5**
2.1. Robot berkaki enam 5
2.2. Invers Kinematik 6
2.3. Sensor Jarak VL53L0X 7
2.4. Motor Servo MG995 7
2.5. Motor Servo SG90 9
2.6. PWM Driver PCA9685 9
2.7. Raspberry Pi 3 B 10
2.8. VNC Viewer dan VNC Server 11
2.9. Blender 12
2.10. V-rep Simulator 12
2.11. Animasi Komputer 13
BAB III PERANCANGAN SISTEM **15**
3.1. Pembuatan Tangga 15
3.2. Perancangan Perangkat Keras 16
3.2.1. Perancangan Elektronik 17
3.2.2. Perancangan Mekanik 18
3.3. Perancangan Sistem Gerak Robot 20
3.3.1. Perancangan *Invers* Kinematik 20

3.3.2. Penerapan *Invers* Kinematik pada Robot 22
3.4. Penggunaan Blender untuk Perancangan Gaya Berjalan ...24
3.4.1. Pembuatan Kerangka Robot pada Blender.....24
3.4.2. Penggunaan Animasi Blender untuk Perancangan Gerakan Robot.....25
3.5. Perancangan Algoritma Naik Tangga 27
3.5.1. Perancangan Sistem Deteksi Tepian Tangga.....28
3.5.2. Diagram alir algoritma melewati tangga 29
3.5.3. Perancangan Gaya Berjalan pada Lantai 31
3.5.4. Perancangan Gaya Berjalan untuk Fase 1 32
3.5.5. Perancangan Gaya Berjalan untuk Fase 2 38
3.5.1. Perancangan Gaya Berjalan untuk Fase 3 43
BAB IV PENGUJIAN DAN ANALISIS 51
4.1. Pengujian Deteksi Tepi Tangga 51
4.1.1. Pengujian Deteksi Tepi Bawah Tangga.....51
4.1.2. Pengujian Deteksi Tepi Atas Tangga 52
4.2. Pengujian Naik Tangga di Lingkungan Simulasi.....53
4.3. Pengujian Algoritma Naik Tangga dengan Robot Purwarupa 55
BAB V PENUTUP 57
5.1. Kesimpulan 57
5.2. Saran 57
DAFTAR PUSTAKA.....59
LAMPIRAN 61
BIODATA.....109

DAFTAR GAMBAR

Gambar 2. 1 Robot Berkaki Enam 5
Gambar 2. 2 Ilustrasi Invers Kinematika pada Hexapod..... 6
Gambar 2. 3 Sensor VL53LOX 7
Gambar 2. 4 Motor Servo MG995 8
Gambar 2. 5 Motor Servo SG90 9
Gambar 2. 6 PWM Driver PCA9685 10
Gambar 2. 7 Raspberry Pi 3 B 10
Gambar 2. 8 Pinout Raspberry Pi B 11
Gambar 2. 9 Animasi Komputer dengan Blender 13
Gambar 3. 1 Gambaran Umum Sistem 15
Gambar 3. 2 Ukuran Tangga..... 16
Gambar 3. 3 Tangga buatan. 16
Gambar 3. 4 Diagram sistem elektronik robot 17
Gambar 3. 5 Skematik I2C pada raspberry pi 3 B..... 18
Gambar 3. 6 Dimensi Kerangka Robot 18
Gambar 3. 7 Desain Mekanik Kaki..... 19
Gambar 3. 8 Kerangka Pemutar Sensor 19
Gambar 3. 9 Robot berkaki enam dengan komponen lengkap..... 20
Gambar 3. 10 Invers Kinematik Sudut 1..... 20
Gambar 3. 11 Invers Kinematik Sudut 2 dan Sudut 3..... 21
Gambar 3. 12 Masukan Invers Kinematik pada setiap kaki robot..... 22
Gambar 3. 13 Input Invers Kinematik setelah diseragamkan..... 23
Gambar 3. 14 Kerangka robot pada blender..... 24
Gambar 3. 15 Keyframe animasi pada Blender 2.8..... 25
Gambar 3. 16 Diagram Alir Interpolasi Array Keyframes..... 26
Gambar 3. 17 Illustrasi 3 fase dalam menaiki tangga 27
Gambar 3. 18 Ilustrasi pendeteksian tepian bawah tangga..... 28
Gambar 3. 19 Illustrasi deteksi ujung tangga..... 29
Gambar 3. 20 Diagram Alir Algoritma Melewati Tangga 30
Gambar 3. 21 Penomoran Kaki Robot 31
Gambar 3. 22 Jalur kaki 0 bidang y-z pada fase 1 32
Gambar 3. 23 Jalur kaki 0 bidang y-x pada fase 1 33
Gambar 3. 24 Jalur kaki 1 bidang y-z pada fase 1 34
Gambar 3. 25 Jalur kaki 1 bidang y-x pada fase 1 34
Gambar 3. 26 Jalur kaki 2 bidang y-z pada fase 1 35
Gambar 3. 27 Jalur kaki 2 bidang y-x pada fase 1 35
Gambar 3. 28 Jalur kaki 3 bidang y-z pada fase 1 36

Gambar 3. 29 Jalur kaki 3 bidang y-x pada fase 1 36
Gambar 3. 30 Jalur kaki 5 bidang y-z pada fase 1 36
Gambar 3. 31 Jalur kaki 5 bidang y-x pada fase 1 37
Gambar 3. 32 Jalur kaki 5 bidang y-z pada fase 1 37
Gambar 3. 33 Jalur kaki 5 bidang y-x pada fase 1 38
Gambar 3. 34 Jalur kaki 0 bidang y-z pada fase 2 39
Gambar 3. 35 Jalur kaki 0 bidang y-x pada fase 2 39
Gambar 3. 36 Jalur kaki 1 bidang y-z pada fase 2 40
Gambar 3. 37 Jalur kaki 1 bidang y-x pada fase 2 40
Gambar 3. 38 Jalur kaki 2 bidang y-z pada fase 2 40
Gambar 3. 39 Jalur kaki 2 bidang y-x pada fase 2 41
Gambar 3. 40 Jalur kaki 3 bidang y-z pada fase 2 41
Gambar 3. 41 Jalur kaki 3 bidang y-z pada fase 2 41
Gambar 3. 42 Jalur kaki 4 bidang y-z pada fase 2 42
Gambar 3. 43 Jalur kaki 4 bidang y-z pada fase 2 42
Gambar 3. 44 Jalur kaki 5 bidang y-z pada fase 2 43
Gambar 3. 45 Jalur kaki 5 bidang y-z pada fase 2 43
Gambar 3. 46 Jalur kaki 0 bidang y-z pada fase 3 44
Gambar 3. 47 Jalur kaki 0 bidang y-x pada fase 3 44
Gambar 3. 48 Jalur kaki 1 bidang y-z pada fase 3 45
Gambar 3. 49 Jalur kaki 1 bidang y-x pada fase 3 45
Gambar 3. 50 Jalur kaki 2 bidang y-z pada fase 3 46
Gambar 3. 51 Jalur kaki 2 bidang y-x pada fase 3 46
Gambar 3. 52 Jalur kaki 3 bidang y-z pada fase 3 47
Gambar 3. 53 Jalur kaki 3 bidang y-x pada fase 3 47
Gambar 3. 54 Jalur kaki 4 bidang y-z pada fase 3 47
Gambar 3. 55 Jalur kaki 4 bidang y-x pada fase 3 48
Gambar 3. 56 Jalur kaki 5 bidang y-z pada fase 3 48
Gambar 3. 57 Jalur kaki 5 bidang y-x pada fase 3 49
Gambar 4. 1 Pengujian deteksi tepi bawah tangga.....52
Gambar 4. 2 Pengujian deteksi tepi atas tangga.....52
Gambar 4. 3 Situasi Pengujian Simulasi Naik Tangga 53
Gambar 4. 4 Model robot simulasi 54
Gambar 4. 5 Situasi pengujian pada robot purwarupa 55

DAFTAR TABEL

Tabel 2. 1 Spesifikasi Servo MG995 8
Tabel 2. 2 Spesifikasi Servo SG90..... 9
Tabel 2. 3 Spesifikasi Raspberry Pi 3 B..... 11
Tabel 3. 1 Tabel sambungan servo kaki dan PCA9685..... 17
Tabel 3. 2 Gerakan pada Lantai 31
Tabel 3. 3 Gerakan pada Fase 1 32
Tabel 3. 4 Gerakan pada fase 2 38
Tabel 3. 5 Gaya Berjalan pada Fase 3 43
Tabel 4. 1 Pengujian Deteksi Tepi Bawah Tangga..... ..51
Tabel 4. 2 Hasil deteksi tepi atas tangga 53
Tabel 4. 3 Pengujian di lingkungan simulasi 54
Tabel 4. 4 Hasil pengujian kedua pada robot purwarupa 55

Halaman ini sengaja dikosongkan

BAB I
PENDAHULUAN

1.1. Latar Belakang

Di era teknologi sekarang ini, robot bergerak atau *mobile robots* merupakan teknologi yang sangat bermanfaat bagi manusia karena membantu sebagian pekerjaan manusia seperti membersihkan lantai, membantu membawakan barang, melakukan eksplorasi, pencarian dan penyelamatan. Terdapat beberapa robot bergerak yang telah dikembangkan seperti Atlas, Bigdog, WildCat, dan Spot dari *Boston Dynamics* [1] dan *rover opportunity* dari NASA [2].

Dua macam robot bergerak yang dipelajari saat ini adalah robot gerak beroda dan robot gerak berkaki [3]. Robot beroda merupakan robot yang paling umum digunakan karena dapat bergerak lebih cepat, lebih mudah untuk dirancang dan dibuat [4]. Robot beroda memiliki kestabilan yang baik karena pusat gravitasi dari robot cenderung tidak berubah saat bergerak [4]. Di lain sisi, robot ini memiliki kelemahan dimana robot memiliki kesulitan untuk beradaptasi melewati medan yang sangat kompleks [3]. Sebaliknya, robot dengan mekanisme gerak kaki memiliki keuntungan untuk melewati medan dengan rintangan-rintangan kecil dan tidak tentu seperti bebatuan dibandingkan dengan robot beroda [5].

Pada tugas akhir ini, robot berkaki digunakan untuk melewati sebuah rintangan yang cukup kompleks yaitu tangga. Tangga merupakan suatu konstruksi yang berfungsi sebagai jembatan ke tempat yang letaknya berada di memiliki ketinggian yang berbeda. Pada kawasan pemukiman manusia, tangga merupakan konstruksi yang umum ditemui sehingga tangga juga menjadi suatu tantangan yang harus bisa ditaklukan oleh robot berkaki. Algoritma berjalan biasa yang tidak dapat digunakan pada robot untuk menaiki tangga. Pada skenario perjalanan dalam bangunan bertingkat, robot berkaki memerlukan kemampuan untuk melewati tangga agar dapat melakukan perjalanan dari suatu lantai ke lantai yang lain.

Algoritma yang dimiliki robot berkaki haruslah bisa membawa robot dari lantai bawah ke lantai atas dengan selamat. Tangga memiliki parameter yang berbeda pada setiap bangunan, terutama tingginya. Pendeteksian tepian tangga diperlukan untuk dapat mengetahui posisi robot terhadap tangga. Pendeteksian tangga dapat dilakukan dengan menggunakan sensor jarak untuk menentukan posisi robot terhadap

tangga. Jumlah kaki pada robot juga memengaruhi stabilitas robot. Robot berkaki enam memiliki stabilitas yang lebih baik dari pada robot berkaki dengan jumlah kaki yang kurang dari enam [5]. Oleh karena itu, digunakan robot berkaki enam untuk stabilitas lebih baik di atas tangga. Dengan algoritma untuk melewati tangga ini, diharapkan robot dapat berhasil untuk menaiki tangga yang telah ditentukan.

1.2. Permasalahan

Perumusan masalah pada penelitian ini adalah:

1. Penggunaan sensor jarak dalam menentukan tepian tangga.
2. Perencanaan gerakan robot untuk menaiki tangga
3. Perancangan algoritma naik tangga dengan informasi dari sensor jarak dan gerakan yang direncanakan

1.3. Tujuan

Tujuan dari dilakukannya penelitian ini adalah:

1. Mampu menggunakan sensor jarak untuk mendeteksi tepi bawah dan tepi atas tangga.
2. Mampu merancang gerakan menaiki tangga
3. Mampu merancang algoritma untuk menaiki tangga

1.4. Batasan Masalah

Berikut ini merupakan batasan masalah yang ada pada penelitian ini:

1. Sensor jarak yang digunakan adalah VL53L0X
2. Dimensi kaki robot adalah 5 cm pada coxa, 6 cm pada femur, dan 13 cm pada tibia.
3. Dimensi yang anak tangga yang digunakan memiliki tinggi 10 cm, lebar 50 cm, dan panjang 15 cm.
4. Tangga memiliki 6 buah anak tangga.

1.5. Metodologi

Berikut ini adalah langkah-langkah yang dikerjakan pada penelitian ini:

1. Studi Literatur

   Pada studi literatur dilakukan pengumpulan berbagai literatur atau sumber sebagai rujukan yang berupa teori, data, ataupun penelitian yang dapat menunjang dalam penulisan tugas akhir ini.

Literatur ini dapat diambil dari buku-buku, jurnal-jurnal, artikel, dan forum diskusi.

2. Tahap Perancangan dan Pembuatan Sistem Gerak Robot

   Pada tahap ini dilakukan perancangan sistem gerak robot berupa kaki-kaki robot yang digerakkan berdasarkan kinematika invers. Dengan menggunakan kinematika invers, kaki robot dapat diatur berdasarkan titik koordinat *end-effector* dari kaki robot tersebut.

3. Tahap Perancangan Sistem Deteksi Tepian Tangga

   Pada tahap ini dilakukan perancangan sistem deteksi tepian tangga. Pendeteksian tepian tangga ini berperan untuk mengubah kondisi pada algoritma menaiki tangga sehingga robot melakukan perjalanan menaiki tangga

4. Tahap Perancangan Algoritma Menaiki Tangga

   Pada tahap ini dilakukan perancangan algoritma pada robot untuk menaiki tangga dengan ukuran yang telah ditentukan. Perancangan algoritma ini meliputi penggunaan sistem deteksi tepian tangga dan perancangan gerakan robot saat di lantai dan di tangga.

5. Tahap Pengujian dan Evaluasi

   Pada tahap pengujian, alat diuji di kondisi yang telah ditentukan untuk mengetahui karakteristik dan kekurangan dari algoritma yang dirancang sehingga dapat dilakukan perbaikan untuk menyesuaikan keluaran yang dihasilkan sesuai dengan yang dikehendaki.

6. Penulisan Laporan Tugas Akhir

   Pada tahap ini, dilakukan penyusunan dan penulisan laporan tugas akhir sesuai dengan struktur yang telah ditentukan. Terdapat tiga bagian besar untuk dimasukkan ke dalam laporan tugas akhir, yaitu bagian awal yang memuat bahan-bahan preliminer, bagian inti/pokok yang memuat naskah utama dari tugas akhir, dan bagian akhir yang memuat bahan-bahan referensi. Penyusunan dan penulisan laporan tugas akhir dilakukan bersamaan dengan perancangan dan pengujian alat.

1.6. Sistematika Penulisan

Laporan pada penelitian ini dibagi menjadi lima bab. Berikut ini adalah penjelasan mengenai masing-masing bab:

BAB 1 PENDAHULUAN

Pada bab ini dijelaskan tentang latar belakang, permasalahan, tujuan, batasan masalah, metodologi, sistematika penulisan, serta relevansi dari penelitian yang dilakukan.

BAB 2 TINJAUAN PUSTAKA

Pada bab ini dibahas mengenai tinjauan pustaka yang membantu penelitian. Beberapa teori yang dapat membantu dalam penelitian ini berlangsung adalah robot berkaki enam, motor servo, raspberry pi, sensor jarak, dan invers kinematik.

BAB 3 PERANCANGAN SISTEM

Pada bab ini membahas tentang perancangan sistem yang akan digunakan pada penelitian ini. Perancangan sistem ini meliputi perancangan perangkat keras dan perancangan perangkat lunak.

BAB 4 PENGUJIAN DAN ANALISIS

Pada bab ini dibahas mengenai pengujian dari perangkat keras dan perangkat lunak. Sistem pengambilan data juga diuji dengan melakukan evaluasi pada data hasil dari pembelajaran obyek. Selanjutnya dilakukan analisa berdasarkan data hasil pengujian.

BAB 5 PENUTUP

Bab ini berisi kesimpulan dan saran dari hasil pembahasan yang telah diperoleh.

1.7. Relevansi

Hasil dari penelitian ini diharapkan dapat membantu agar robot berkaki enam atau *hexapod* dapat dilengkapi dengan algoritma melewati tangga. Perangkat pendeteksi tepian tangga diharapkan bisa ditingkatkan sehingga tidak hanya mendeteksi tepian tangga tapi juga bisa digunakan untuk mendapatkan parameter tangga yang lainnya seperti tinggi, lebar, panjang, dan jumlah anak tangga.

BAB II
TINJAUAN PUSTAKA DAN TEORI PENUNJANG

Pada bab ini akan dijelaskan mengenai tinjauan pustaka dan teori penunjang yang dapat membantu penelitian. Tinjauan pustaka akan berisi mengenai penelitian-penelitian yang telah dilakukan sebelumnya. Teori ini digunakan sebagai dasar untuk perancangan sistem pada bab III.

2.1. Robot berkaki enam

Robot berkaki terinspirasi pada hewan yang diamati di alam, pendekatan yang sering dilakukan pada desain dan konstruksinya adalah mengembangkan tiruan mekatronik hewan yang dimaksudkan untuk direplikasi, baik dalam hal dimensi fisiknya, atau dalam hal karakteristik seperti gaya berjalan dan aktuasi anggota badan [6]. Robot berkaki banyak menunjukkan keuntungan yang signifikan dibandingkan dengan robot beroda untuk berjalan di medan yang berat karena mereka tidak memerlukan kontak terus-menerus dengan tanah [7].

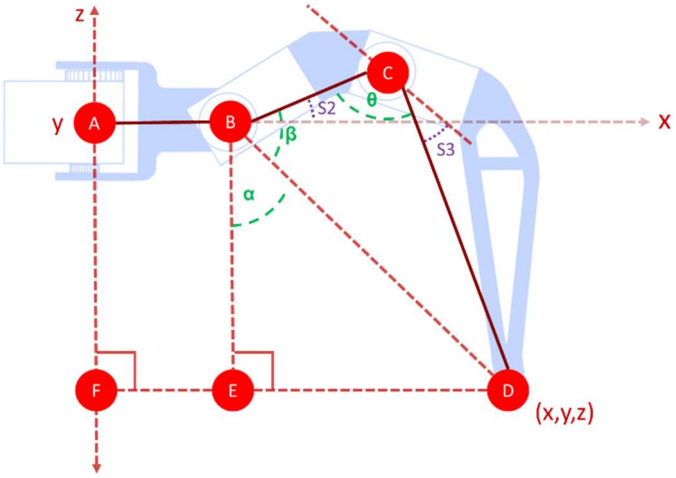
Robot berkaki enam merupakan robot yang bergerak dengan memanfaatkan enam kaki yang dipasang di badannya. Robot berkaki enam atau robot *hexapod* pada umumnya dapat dikategorikan menjadi persegi panjang dan heksagonal [7]. Pada tugas akhir ini digunakan robot *hexapod* dengan bentuk persegi panjang. *Hexapod* dengan bentuk persegi panjang terinspirasi dari serangga yang memiliki enam kaki yang simetris sepanjang dua sisi, dimana setiap sisinya memiliki tiga kaki [7]. Robot berkaki enam memiliki kelebihan karena terkena sedikit dampak dari medan sehingga memiliki stabilitas yang baik di lingkungan alami [8].

Gambar 2. 1 Robot Berkaki Enam

Pada tugas akhir ini digunakan robot berkaki enam karena robot dengan kaki enam memiliki stabilitas yang lebih baik ketika bergerak dan berdiri daripada robot dengan kaki kurang dari enam [5]. Kaki-kaki pada robot yang digunakan pada tugas akhir ini memiliki 18 DoF (*Degree of Freedom*) secara keseluruhan. Untuk mengontrol kaki-kaki pada robot berkaki enam ini penulis menggunakan metode *inverse kinematics* dengan perhitungan berdasarkan pendekatan geometri.

2.2. Invers Kinematik

Kinematika mempelajari gerakan dari tubuh tanpa mempertimbangkan gaya ataupun momen yang menyebabkan gerakan tersebut [9]. Invers kinematik merupakan suatu metode analisa untuk melakukan transformasi dari ruang Cartesian ke ruang sendi [10].



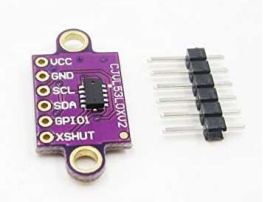
Gambar 2. 2 Ilustrasi Invers Kinematika pada Hexapod

Terdapat dua solusi pendekatan *invers* kinematik, yaitu secara geometri dan secara aljabar yang digunakan untuk memperoleh solusi *invers* kinematik secara analitis [9]. Pada tugas akhir ini, digunakan *invers* kinematik dengan pendekatan secara geometri berdasarkan penelitian yang telah dilakukan oleh Johan Wijaya Kusuma [10]. Metode *invers* kinematik tersebut dibuat untuk untuk menghitung *invers* kinematik pada kaki robot berkaki enam yang memiliki 18 DoF. Pendekatan *invers* kinematik ini menggunakan analisis persamaan kinematik dapat

diselesaikan dengan cara yang paling dasar yaitu menggunakan trigonometri dengan bantuan grafik [10].

2.3. Sensor Jarak VL53L0X

Sensor jarak merupakan sensor yang mampu mengukur jarak dari sensor terhadap suatu titik. Sensor jarak pada umumnya bekerja dengan merambatkan sinyal tertentu dan kemudian membaca perubahan yang terjadi pada sinyal yang kembali atau dipantulkan berupa perubahan intensitas atau waktu dari sinyal tersebut dikembalikan [11].



Gambar 2. 3 Sensor VL53LOX [12]

Sensor jarak memiliki beberapa faktor yang dapat memengaruhi pengukuran, yaitu resolusi, kecepatan pembaruan (*update rate*), dan jangkauan (*range*). Resolusi merupakan tingkat ketelitian dari sensor jarak terhadap jarak yang diukur [11]. Kecepatan pembaruan merupakan kemampuan sensor dalam mengambil infomasi jarak dalam waktu tertentu [11]. Kemudian, jangkauan adalah jarak minimum dan maksimum yang mampu diukur oleh sensor [11].

Pada tugas akhir ini kami menggunakan sensor jarak VL53L0X yang merupakan sensor jarak berbasis VC. Sensor ini menggunakan teknologi VCSEL (Vertical Cavity Surface-Emitting Laser) yang memancarkan sinar laser berdaya rendah dengan panjang gelombang 940 nm. VL53L0X memiliki jangkauan ukur sejauh 2 m. VL53L0X dilengkapi dengan I2C yang dapat diubah alamatnya agar memungkinkan untuk dipasang lebih dari pada satu sensor.

2.4. Motor Servo MG995

Pada robot berkaki, motor servo merupakan salah satu komponen penting karena berperan sebagai sendi atau *joint* pada kaki robot. Motor servo merupakan motor yang dapat memberikan posisi sudut yang diinginkan sesuai dengan sinyal input yang diberikan. Sirkuit servo dibangun tepat di dalam unit motor dan memiliki poros yang dapat

diposisikan, yang biasanya dilengkapi dengan roda gigi. Motor dikendalikan dengan sinyal listrik yang menentukan jumlah gerakan poros. Servo dikontrol dengan mengirimkan sebuah pulsa listrik dengan lebar bervariabel atau *pulse width modulation* (PWM) melalui kabel kontrol [14]. Spesifikasi servo MG995 dapat dilihat pada Tabel 2. 1.

Tabel 2. 1 Spesifikasi Servo MG995

Spesifikasi	Nilai
Tegangan Operasi	4.8~ 6.6v
Torsi	9.4kgf/cm (4.8v); 11kgf/cm (6v)
Kecepatan	0.2sec/60degree (4.8v) 0.16sec/60degree (6.0v)
Arus ditarik	10 mA (Idle) 170 mA (No. load) 1200 mA (Stall)
Dead band	1 us
Berat	55 g
Dimensi	40.7×19.7×42.9mm

Gambar 2. 4 Motor Servo MG995 [13, hlm. 995]

2.5. Motor Servo SG90

Gambar 2. 5 Motor Servo SG90 [15, hlm. 90]

Motor Servo SG90 merupakan servo mikro atau servo berukuran kecil yang berdaya rendah. Torsi sebesar 1.8 kgf·cm pada servo ini tidak cocok untuk digunakan sebagai kaki robot. Pada tugas akhir ini, servo SG90 digunakan sebagai pemutar sensor jarak VL53L0X yang dipasang secara vertikal. Servo SG90 dapat dilihat pada Gambar 2. 5. Adapun spesifikasi dari servo SG90 dapat dilihat pada Tabel 2. 2.

Tabel 2. 2 Spesifikasi Servo SG90

Spesifikasi	Nilai
Berat	9 gram
Dimensi	22.2 x 11.8 x 31 mm
Torsi	1.8 kgf·cm
Tegangan operasi	4.8
Dead bandwidth	10 us
Kecepatan	0.1 s/60 derajat

2.6. PWM Driver PCA9685

PCA9685 merupakan PWM (*Pulse Width Modulation*) *driver* dengan 16-kanal output yang dapat dikontrol melalui I2C. PCA9685 dapat digunakan untuk mengontrol motor servo. Setiap motor servo memiliki pengontrol PWM tersendiri dengan frekuensi tetap yang beroperasi pada frekuensi yang dapat diprogram dari tipikal 24 Hz hingga 1526 Hz dengan *duty cycle* yang dapat disesuaikan dari 0 hingga 100% untuk memungkinkan motor servo diatur pada nilai tertentu [16].

Gambar 2. 6 PWM Driver PCA9685 [17]

2.7. Raspberry Pi 3 B

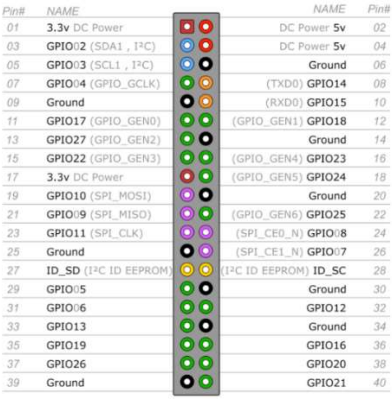
Raspberry pi merupakan perangkat komputer portabel yang memiliki ukuran yang kecil dengan dimensi 85 mm x 56 mm x 17 mm. Banyak peneliti melakukan penelitian mereka dengan kecenderungan untuk memanfaatkan lebih banyak keunggulan komputer Raspberry Pi [19]. Raspberry pi dioperasikan seperti komputer pada umumnya dengan masukan berupa keyboard, mouse dan layar penampil yang dihubungkan pada port HDMI. Raspberry Pi juga dapat dioperasikan tanpa menggunakan perangkat diatas dengan menghubungkannya dengan komputer lain yang telah dilengkapi oleh VNC Viewer. Raspberry pi dengan sistem operasi NOOBS telah dilengkapi dengan VNC Server yang memungkinkan Raspberry Pi untuk dikontrol melalui VNC Viewer dari komputer lain dan menggunakan masukan dan layar dari komputer tersebut. Raspberry Pi 3 B dilengkapi dengan spesifikasi seperti pada Tabel 2. 3 Spesifikasi Raspberry Pi 3 B:


Gambar 2. 7 Raspberry Pi 3 B [18]

Tabel 2. 3 Spesifikasi Raspberry Pi 3 B

Komponen	Keterangan
Dimensi	85mm x 56mm x 17mm
Prosesor	Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
RAM	1 GB
Konektivitas	CM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
GPIO	40-pin extended GPIO
USB Port	4 USB 2.0
Display Port	Full size HDMI

Pinout dari Raspberry Pi 3 B dapat dilihat pada gambar berikut:



Gambar 2. 8 Pinout Raspberry Pi B [20]

2.8. VNC Viewer dan VNC Server

VNC *Viewer* dan VNC *Server* merupakan program yang dibuat oleh Real VNC untuk mengakses perangkat komputer yang dimiliki dari mana saja dan kapan saja [21]. VNC server memungkinkan perangkat Raspberry Pi untuk dikontrol secara jauh atau *remote* dari perangkat lain dengan menggunakan VNC Viewer. VNC Server merupakan perangkat lunak yang telah tersedia pada sistem operasi NOOBS yang dijalankan pada Raspberry Pi. VNC Server dan Viewer dapat terhubung melalui jaringan lokal maupun jaringan internet dengan melakukan verifikasi sistem keamanan dari pengakses (VNC *viewer*) ke perangkat penyedia (VNC *server*) berupa akun yang telah dibuat.

VNC *server* dan *viewer* dapat melakukan komunikasi data berupa tangkapan layar secara langsung, masukan *keyboard* dan *mouse*, dan transfer file. Penggunaan VNC *server* dan *viewer* ini dapat memudahkan penelitian ini karena penulis mendapatkan keleluasaan dalam mengatur perangkat Raspberry Pi yang terpasang pada robot secara terpisah.

2.9. Blender

Blender merupakan perangkat lunak 3 dimensi (3D) terbuka (*open source*) dan gratis yang memiliki banyak fitur yang berkaitan dengan kreasi pada ruang 3D seperti pemodelan, *rigging*, animasi, simulasi, *rendering*, *compositing*, dan *motion tracking* [22]. Pada tugas akhir ini, Blender digunakan untuk menyederhanakan perhitungan pada perencanaan gerakan robot (*gait*) dengan menggunakan fitur *rigging* dan animasi pada Blender.

Pembuatan animasi karakter 3D biasanya menggunakan kerangka digital untuk mempermudah manipulasi bentuk objek atau bentuk karakter 3D. Pembuatan kerangka digital pada karakter 3D dapat dilakukan dengan teknik *Rigging*. *Rigging* merupakan teknik yang digunakan pada animasi berkerangka untuk merepresentasikan sebuah model karakter 3D menggunakan serangkaian tulang digital yang saling berhubungan [23].

Meskipun penggunaan kerangka digital dapat mempermudah manipulasi bentuk karakter 3D, tetapi proses manipulasi dan transformasi pada tulang sendiri masih cukup rumit. Oleh karena itu, diterapkan kinematika *invers* pada transformasi tulang digital untuk mempermudah proses manipulasi antar tulang.

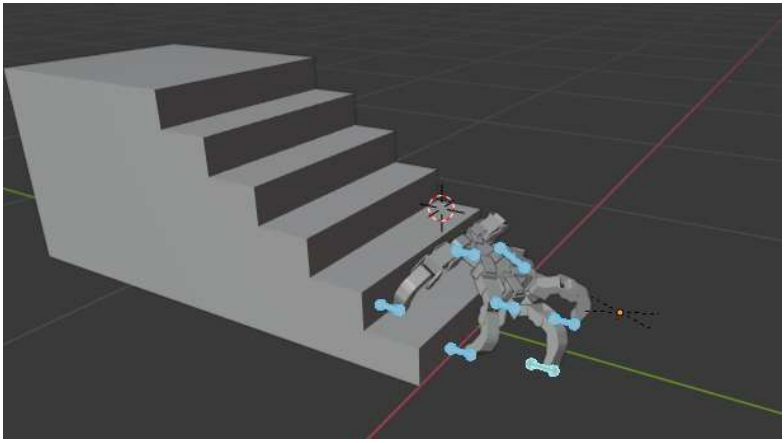
2.10. V-rep Simulator

V-rep merupakan simulator robotika yang dibuat oleh *CoppeliaRobotics* untuk mensimulasikan berbagai perilaku robotika baik pada *mobile robot* maupun *non-mobile robot*. V-rep yang digunakan pada tugas akhir ini adalah V-rep versi pro edu. V-rep versi ini memungkinkan pengguna untuk dapat mengakses perangkat lunak ini secara gratis.

Pada tugas akhir ini, V-rep digunakan untuk mensimulasikan robot berkaki enam untuk melewati tangga yang telah ditentukan. Robot berkaki enam dan tangga yang telah dibuat model 3 dimensinya perlu dikonfigurasi terlebih dahulu mekanismenya dalam V-rep agar dapat beroperasi seperti yang dikehendaki.

2.11. Animasi Komputer

Animasi komputer, yang selanjutnya disebut animasi, adalah setiap perhitungan berbasis komputer yang digunakan untuk menghasilkan keluaran visual yang menciptakan persepsi pergerakan nyata [24]. Terdapat 3 jenis pendekatan umum untuk kontrol gerak yaitu animasi artistik yang merupakan animasi yang dikembangkan oleh *animator*, animasi *data-driven* yang merupakan gerakan yang didigitalisasi dan dipetakan ke objek grafis, dan animasi prosedural yang gerakannya dikontrol via model komputasi [24]. Pada tugas akhir ini, digunakan animasi sebagai metode perancangan gerak untuk menaiki tangga. Metode animasi dipilih pada tugas akhir ini untuk merancang gerakan karena cara ini dianggap lebih mudah oleh penulis daripada merancang dengan membuat model dari pergerakan robot.

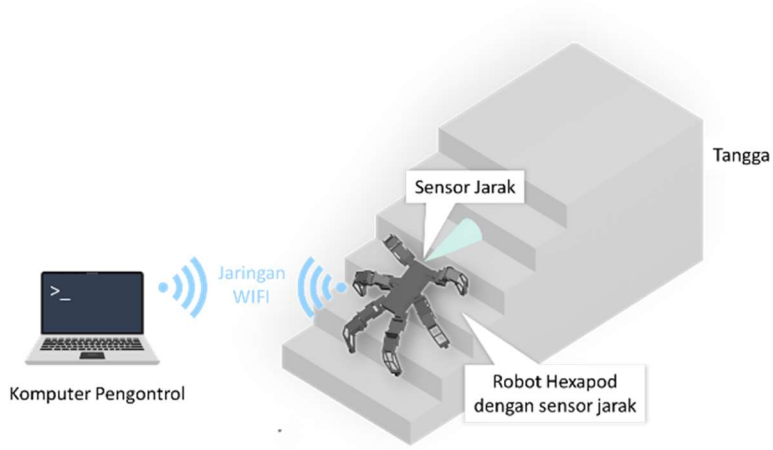


Gambar 2. 9 Animasi Komputer dengan Blender

Halaman ini sengaja dikosongkan

BAB III
PERANCANGAN SISTEM

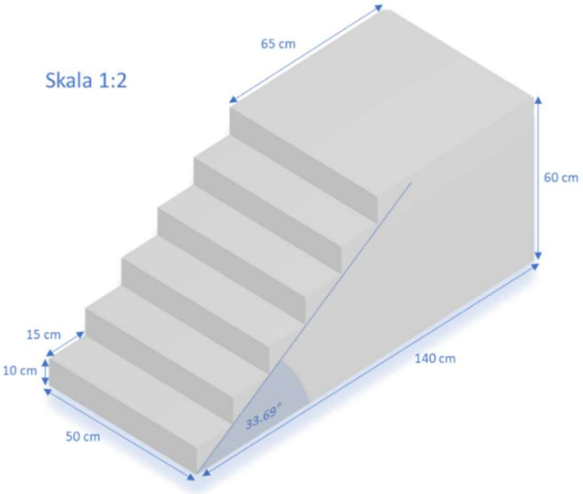
Pada bab ini akan dijelaskan mengenai perancangan sistem yang meliputi perancangan perangkat keras, perancangan kinematika robot, perancangan gerakan dengan blender, dan algoritma robot untuk melewati tangga. Robot berkaki enam akan berpindah dari lantai bawah ke lantai atas tangga dengan dimensi tangga yang telah ditentukan.



Gambar 3. 1 Gambaran Umum Sistem

3.1. Pembuatan Tangga

Pada tugas akhir ini, tangga dibuat dengan skala setengah atau 0.5 dari ukuran asli berdasarkan standar tangga. Tangga yang dibuat untuk penelitian ini memiliki ukuran tinggi 10 cm, panjang 15 cm dan lebar 50 cm pada setiap anak tangganya. Tangga yang dibuat memiliki 6 buah anak tangga dan bordes atau bidang luas pada ujung atas tangga sebesar 50 cm x 50 cm. Sehingga, tangga yang dibuat memiliki ukuran tinggi total 60 cm, panjang 140 cm, dan lebar 50 cm. Ilustrasi ukuran tangga dapat dilihat pada Gambar 3. 2 dan tangga yang telah dibuat ada pada Gambar 3. 3.



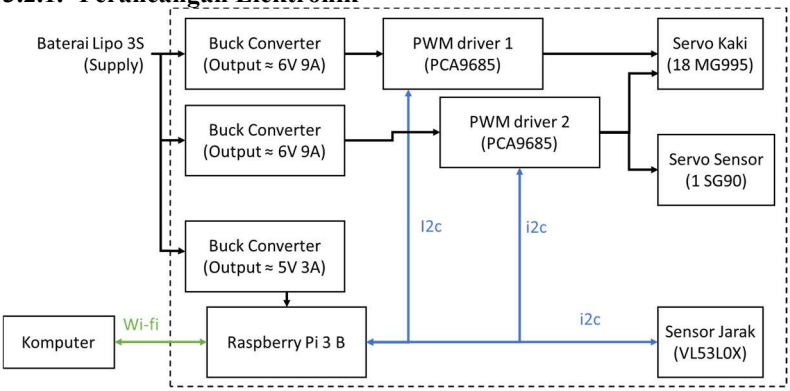
Gambar 3. 2 Ukuran Tangga

Gambar 3. 3 Tangga buatan.

3.2. Perancangan Perangkat Keras

Perancangan perangkat keras pada robot berkaki enam yang digunakan pada penelitian ini meliputi perancangan elektronik dan perancangan mekanik robot.

3.2.1. Perancangan Elektronik



Gambar 3. 4 Diagram sistem elektronik robot

Diagram sistem elektronik robot pada penelitian ini dapat dilihat pada Gambar 3. 4. Robot ini menggunakan baterai Lipo 3 sel sebagai sumber daya. Tegangan dari baterai perlu diubah menjadi lebih kecil untuk menyesuaikan dengan kebutuhan dari komponen elektronik dalam robot. *Buck Converter* digunakan untuk mengubah tegangan dari baterai menjadi 5 V untuk raspberry pi dan 6 V untuk suplai motor servo. Penggunaan *buck converter* dengan kemampuan untuk menarik arus sebesar 18 Ampere dibutuhkan dengan asumsi setiap motor servo pada kaki membutuhkan arus maksimal 1 Ampere saat beroperasi.s

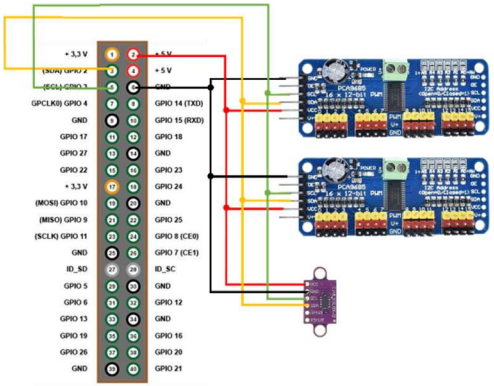
Servo yang memutar sensor jarak secara vertikal dihubungkan dengan pin nomor 6 pada PCA 1, sedangkan pemasangan motor servo kaki pada kedua PCA9685 ditunjukkan pada Tabel 3. 1.

Tabel 3. 1 Tabel sambungan servo kaki dan PCA9685

Kaki	Coxa	Femur	Tibia
0	13 (PCA 2)	15 (PCA 2)	12 (PCA 2)
1	0 (PCA 1)	2 (PCA 1)	3 (PCA 1)
2	5 (PCA 2)	7 (PCA 2)	4 (PCA 2)
3	9 (PCA 1)	8 (PCA 1)	11 (PCA 1)
4	2 (PCA 2)	0 (PCA 2)	3 (PCA 2)
5	13 (PCA 1)	15 (PCA 1)	12 (PCA 1)

Sensor jarak VL53L0X dan dua PCA9685 dihubungkan dengan Raspberry Pi 3 B melalui jalur I2C dengan pin GPIO 2 dihubungkan

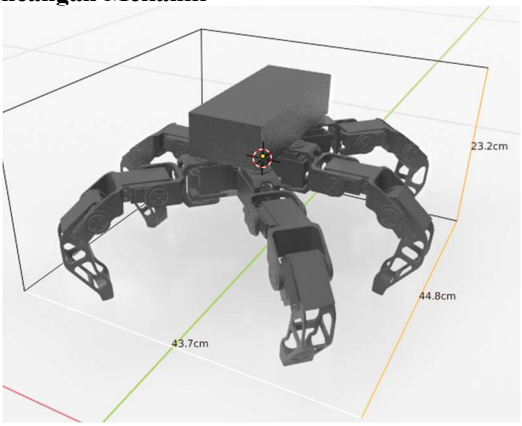
dengan SDA dan GPIO 3 sebagai SCL seperti yang ditunjukkan pada Gambar 3. 5.



Gambar 3. 5 Skematik I2C pada raspberry pi 3 B

Raspberry Pi 3 B dikendalikan oleh komputer melalui jaringan Wi-Fi menggunakan VNC Viewer dan VNC Server yang telah dikonfigurasi sebelumnya. Komputer berfungsi untuk melakukan pemrograman dalam raspberry pi dan melakukan eksekusi program yang telah dibuat untuk diopeasikan pada robot berkaki enam.

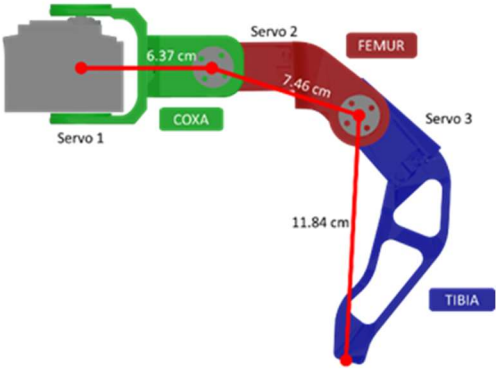
3.2.2. Perancangan Mekanik



Gambar 3. 6 Dimensi Kerangka Robot

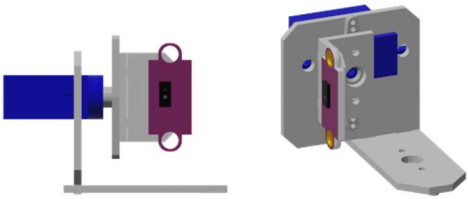
Pada tugas akhir ini, kerangka robot yang digunakan berbasis kerangka *PhantomX* yang telah dimodifikasi. Dimensi kerangka robot dengan posisi kaki awal adalah 43,7 x 44,8 x 23,2 cm seperti yang ditunjukkan pada Gambar 3. 6. Kerangka robot ini dibuat dengan menggunakan mesin 3D printer dengan bahan PLA.

Kerangka robot ini memiliki 6 buah kaki yang dipasang pada tubuh utama. Kaki robot memiliki 3 bagian yaitu *coxa, femur*, dan *tibia*. Panjang dari *coxa, femur*, dan *tibia* masing-masing adalah 6,37 cm, 7,46 cm, dan 11,84 cm sepeti yang dapat dilihat pada Gambar 3. 7.



Gambar 3. 7 Desain Mekanik Kaki

Pada tugas akhir ini, pemutar sensor digunakan untuk memutar sensor secara vertikal sehingga dibuat kerangka yang seperti pada Gambar 3. 8. Kerangka dari pemutar sensor menggunakan bahan akrilik yang dipotong dengan laser untuk mendapatkan hasil yang sesuai dengan desain yang dibuat. Pemutar sensor ini dipasang pada ujung papan komponen dan mengarah kedepan robot.



Gambar 3. 8 Kerangka Pemutar Sensor

Penempatan komponen pada robot dilakukan dengan menggunakan tempat komponen yang terbuat dari 2 buah papan dengan dimensi 21 x 9 cm. Papan yang dibuat dipasang bertumpuk dihubungkan menggunakan spacer 6 cm. Pada Gambar 3. 9, komponen robot telah dipasang lengkap dengan semua kerangka dan komponen yang digunakan. Massa total dari robot yang telah dipasang lengkap mencapai 2.5 kg.

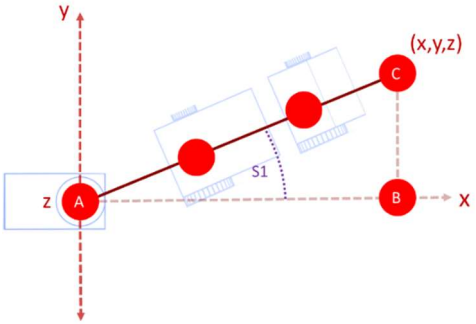
Gambar 3. 9 Robot berkaki enam dengan komponen lengkap

3.3. Perancangan Sistem Gerak Robot

Pada Perancangan sistem gerak robot, dibahas mengenai *invers* kinematik dan penerapannya pada robot berkaki enam yang digunakan.

3.3.1. Perancangan *Invers* Kinematik

Invers kinematik yang digunakan pada penelitian ini menggunakan *invers* kinematik berdasarkan penelitian dari jurnal yang dilakukan oleh Johan Wijaya Kusuma [10].

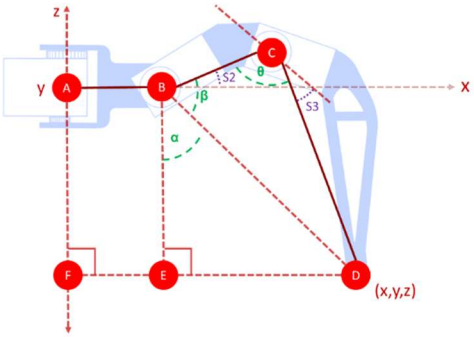


Gambar 3. 10 Invers Kinematik Sudut 1

Pada Gambar 3. 10, ditunjukkan kaki robot dari sisi atas. Titik A adalah servo yang menggerakkan kaki secara horizontal, B adalah C pada sumbu x, dan C adalah ujung kaki. S1 merupakan sudut dari servo 1 atau servo pada titik A. Persamaan untuk mencari sudut dari servo 1 ada pada persamaan 3.1.

$$S1 = tan^{-1} \frac{BC}{AB} = tan^{-1} \frac{y}{x} \quad (1)\ [10]$$

Pada Gambar 3. 11, ditunjukkan A adalah servo 1, B adalah Servo 2, C adalah servo 3 dan D adalah ujung kaki. E dan F merupakan titik potong antara A dan B dengan D. S2 dan S3 adalah sudut servo 2 dan sudut servo 3.



Gambar 3. 11 Invers Kinematik Sudut 2 dan Sudut 3

Untuk mencari sudut 2 dapat digunakan persamaan 3.2 dibawah ini:

$$S2 = (\beta + \alpha) - 90 \quad (2)[10]$$

Untuk mendapatkan β dan α dapat digunakan persamaan dibawah ini:

$$\alpha = tan^{-1} \frac{DE}{BE} \quad (3)\ [6]$$

$$DE = DF - EF \quad (4)\ [6]$$

$$BD = \sqrt{DE^2 + BE^2} \quad (5)\ [6]$$

$$\beta = arccos\left(\frac{BC^2 + BD^2 - CD^2}{2.BC.BD}\right) \quad (6)\ [6]$$

Untuk mencari sudut 3 dapat digunakan persamaan 3.3 dibawah ini:

$$S3 = (90 - \theta) \quad (7)\ [6]$$

Untuk mendapatkan θ dapat digunakan persamaan dibawah ini:

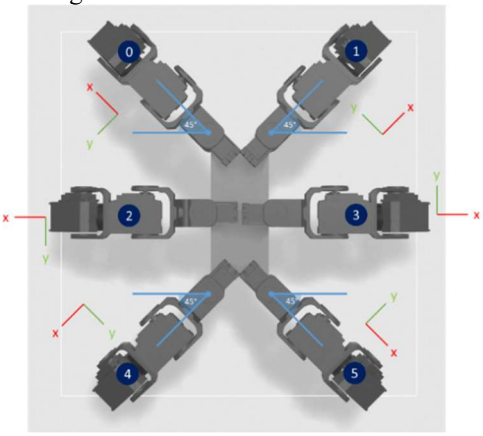
$$\theta = arccos(\frac{BC^2+CD^2-BD^2}{2.BC.CD}) \quad (8)\ [6]$$

, dimana:

AB = Panjang coxa
BC = Panjang femur
CD = Panjang tibia
AF = BE = z
DF = x

3.3.2. Penerapan *Invers* Kinematik pada Robot

Invers Kinematik yang telah dibahas diatas tidak dapat langsung digunakan karena tidak semua kaki robot dipasang sama. Perbedaan letak kaki dan arah kaki yang berbeda perlu untuk diperhitungkan untuk mendapatkan keseragaman masukan *invers* kinematik untuk semua kaki.



Gambar 3. 12 Masukan Invers Kinematik pada setiap kaki robot

Pada Gambar 3. 12 merupakan input *invers* kinematik jika diterapkan secara langsung. Input dari *invers* kinematik dapat diseragamkan dengan memutar dan membalikkan input *invers* kinematik pada setiap kaki robot.

Pemutaran *invers* kinematik pada kaki-kaki robot tersebut adalah sebagai beikut:

1. Pada kaki 0: $x'_0 = -(\cos(-45) * x_0 - \sin(-45) * y_0)$
   $y'_0 = -(\sin(-45) * x_0 + \cos(-45) * y_0)$
   $z'_0 = z_0$

2. Pada kaki 1:

$$x'_1 = \cos(45) * x_1 - \sin(45) * y_1$$
$$y'_1 = \sin(45) * x_1 + \cos(45) * y_1$$
$$z'_1 = z_1$$

3. Pada kaki 2:

$$x'_2 = -x_2$$
$$y'_2 = -y_2$$
$$z'_2 = z_2$$

4. Pada kaki 3:

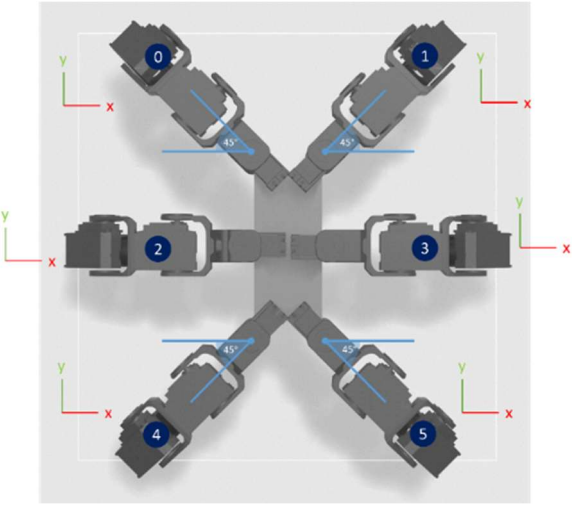
$$x'_3 = x_3$$
$$y'_3 = y_3$$
$$z'_3 = z_3$$

5. Pada kaki 4:

$$x'_4 = -(\cos(45) * x_4 - \sin(45) * y_4)$$
$$y'_4 = -(\sin(45) * x_4 + \cos(45) * y_4)$$
$$z'_4 = z_4$$

6. Pada kaki 5:

$$x'_5 = \cos(-45) * x_5 - \sin(-45) * y_5$$
$$y'_5 = \sin(-45) * x_5 + \cos(-45) * y_5$$
$$z'_5 = z_5$$



Gambar 3. 13 Input Invers Kinematik setelah diseragamkan

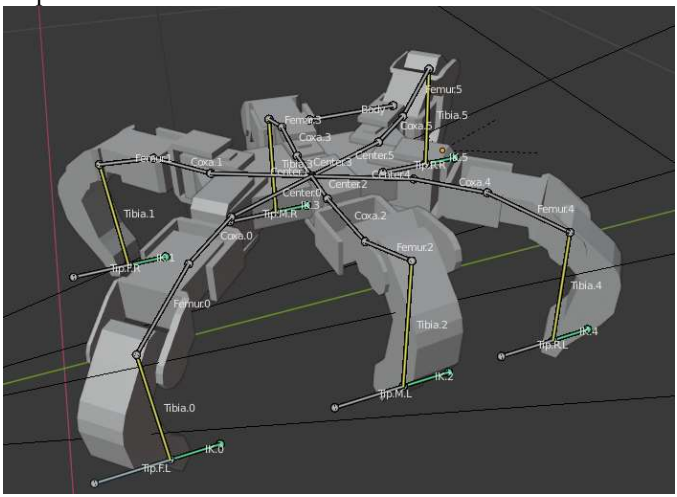
Input *invers* kinematik pada robot yang telah diseragamkan arahnya dapat dilihat pada Gambar 3. 13. Penyeragaman arah ini dapat mempermudah perencanaan gerakan menggunakan *invers* kinematik karena tidak perlu memperhitungkan arah dari setiap kaki pada perhitungan input.

3.4. Penggunaan Blender untuk Perancangan Gaya Berjalan

Perencanaan gerakan dengan menggunakan perhitungan manual adalah hal yang sulit dan memakan waktu cukup lama. Kurangnya visualisasi data dapat menyebabkan robot untuk bertindak tidak sesuai dengan ekspektasi. Penggunaaan Blender dalam penelitian ini digunakan untuk mempermudah visualisasi gerakan dan mempersingkat waktu perancangan gerakan.

3.4.1. Pembuatan Kerangka Robot pada Blender

Perancangan gerakan pada blender dengan robot berkaki enam dilakukan dengan mengimpor model dan membuat kerangka atau *armature* pada blender seperti pada Gambar 3. 14. Kerangka yang dibuat ini terdiri dari tulang-tulang atau *bones*. Tulang-tulang ini berfungsi untuk memanipulasi bentuk dari karakter robot.



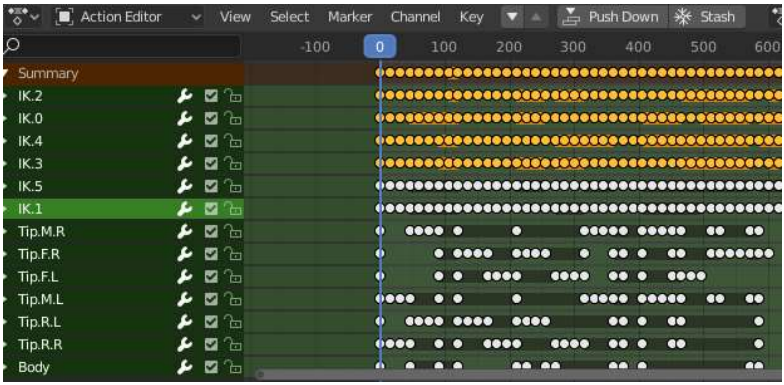
Gambar 3. 14 Kerangka robot pada blender

Kerangka robot yang dibuat terdiri dari tulang *coxa*, *femur, tibia*, IK (*invers* kinematik), *tip*, dan *body*. Pada *tibia* ditambahkan *modifier* berupa *invers* kinematik dengan 3 rantai meliputi tulang *coxa, femur,* dan *tibia* tersebut dan target *invers* kinematiknya adalah tulang IK. *Parent* dari tulang IK tersebut adalah *center* yang merupakan bagian dari *body* atau tubuh. *Center* digunakan sebagai *parent* dari tulang IK karena posisi *tail* atau ekor dari tulang *center* merupakan titik origin dari *invers* kinematik dari robot.

Selain di-*parent*-kan dengan tulang *center*, tulang IK juga ditambahkan *modifier copy location* untuk menyalin posisi dari tulang *tip*. Tulang *tip* tidak di-*parent*-kan pada apapun agar tidak berubah nilai posisinya jika tulang lain dipindahkan. Tulang *tip* adalah tulang yang digunakan penulis untuk merencanakan gerakan dari robot. Selain tulang *tip*, tulang *body* juga digunakan penulis untuk memanipulasi pergerakan dari robot. Tulang *body* digunakan untuk memindahkan dan memutarkan tubuh robot selain kaki.

3.4.2. Penggunaan Animasi Blender untuk Perancangan Gerakan Robot

Animasi yang dibuat dengan mengubah posisi tulang *tip* dan tulang *body* pada robot direkam dan disimpan dalam suatu *keyframe* yang berisi informasi berupa posisi 3 dimensi, rotasi 3 dimensi, dan nomor *frame*. Nomor *frame* tersebut berperan sebagai ukuran waktu dalam animasi dimana lama waktu dari *frame* satu ke *frame* selanjutnya ditentukan oleh *frame per second* (FPS). *Keyframe* pada blender ditampilkan seperti pada Gambar 3. 15.

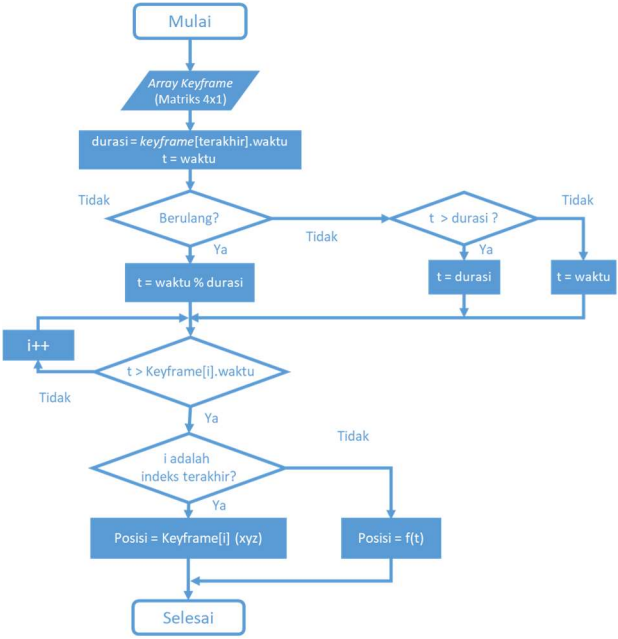


Gambar 3. 15 Keyframe animasi pada Blender 2.8

Untuk dapat digunakan pada robot, perlu dilakukan transfer data dari Blender ke program. Data yang ditransfer adalah setiap *frame* dari tulang IK. Tulang IK direkam setiap 15 *frame* dengan 30 FPS dari *frame* ke 0 sampai *frame* akhir dari animasi. Rekaman tulang IK diekspor dengan matrix 4x1 yang terdiri dari posisi x, posisi y, posisi z, dan waktu. Waktu didapatkan dengan membagi nomor *frame* dengan FPS. Sehingga, data yang diekspor adalah seperti berikut:

$$Keyframe = \begin{bmatrix} x \\ y \\ z \\ waktu \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \dfrac{nomor\ frame}{fps} \end{bmatrix}$$

Data tersebut diekspor ke dalam file dalam format .dat dalam array dan diurutkan berdasarkan waktu. Setelah diekspor, file tersebut dibaca oleh program untuk diubah lagi kedalam bentuk data yang sama. Pembacaan file ini memerlukan fungsi yang dapat membaca posisi x, y, dan z dari waktu ke waktu. Fungsi yang digunakan dapat berupa interpolasi antar *keyframe* sederhana seperti yang ditunjukkan pada diagram alir dibawah ini:



Gambar 3. 16 Diagram Alir Interpolasi Array Keyframes

Fungsi *f(t)* pada diagram alir di atas dijabarkan seperti pada persamaan (9).

$$f(t) = \frac{(p_{i+1} - p_i)}{(t_{i+1} - t_i)} \cdot (t - t_i) \qquad (9)$$

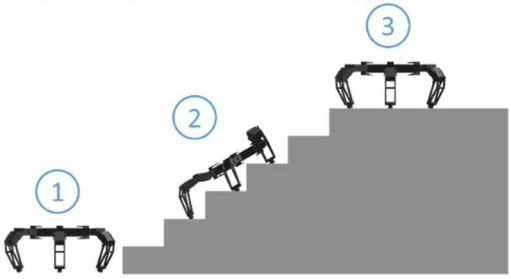
Keterangan :

t = waktu input
t_i = waktu pada *keyframe* i
t_{i+1} = waktu pada *keyframe* i+1
p_i = data posisi (x, y, z) pada *keyframe* i
p_{i+1} = data posisi (x, y, z) pada *keyframe* i + 1

Diagram alir pada Gambar 3. 16 memungkinkan program robot untuk menghasilkan titik linier antara *keyframe* satu dan *keyframe* selanjutnya dengan menggunakan input waktu. Penggunaan sistem ini dapat diimplementasikan ke dalam *loop* program utama robot tanpa memblokir *loop* tersebut.

Program tersebut juga dilengkapi dengan pengulangan siklus gerakan dengan membagi waktu dengan durasi animasi gerakan. Pengulangan ini berguna untuk melakukan siklus pengulangan seperti siklus berjalan dengan catatan pose awal dan pose akhir siklus sama.

3.5. Perancangan Algoritma Naik Tangga



Gambar 3. 17 Illustrasi 3 fase dalam menaiki tangga

Menaiki tangga merupakan proses untuk berpindah dari lantai yang rendah ke lantai yang lebih tinggi dengan menggunakan tangga. Saat menaiki tangga, terdapat 3 fase perjalanan yaitu dari fase peralihan dari lantai bawah ke tangga, fase berjalan diatas tangga, dan fase peralihan dari tangga ke lantai atas. Perubahan fase dari fase 1 sampai fase 3 memerlukan pemicu berupa deteksi tepian bawah tangga dan tepian atas

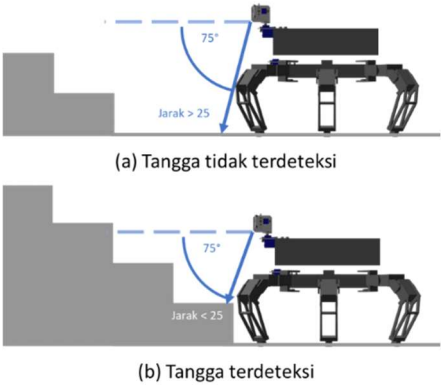
tangga. Ilustrasi ketiga fase dalam menaiki tangga dapat dilihat pada Gambar 3. 17.

3.5.1. Perancangan Sistem Deteksi Tepian Tangga

Saat fase peralihan diujung tangga, robot memerlukan penggunaan perangkat deteksi agar robot dapat mengetahui posisi robot terhadap tangga yang ada didepannya. Pada tugas akhir ini, perangkat deteksi yang digunakan adalah sensor jarak VL53L0X yang dilengkapi dengan servo SG90 untuk memutar sensor secara vertikal. Mode yang digunakan pada sensor jarak adalah mode kecepatan tinggi karena pada *library* program yang digunakan membuat program utama terblokir saat melakukan pengukuran.

Sistem deteksi yang digunakan mengetahui posisi robot terhadap tangga dengan mendeteksi tepian tangga bawah dan tepian tangga atas dengan membedakan jarak antara lantai dengan anak tangga. Tepian tangga bawah merupakan anak tangga yang posisinya paling di bawah dan tepian tangga atas merupakan anak tangga paling di atas. Pendeteksian tepian tangga bawah dan atas adalah sebagai berikut:

1. Pendeteksian tepian bawah tangga



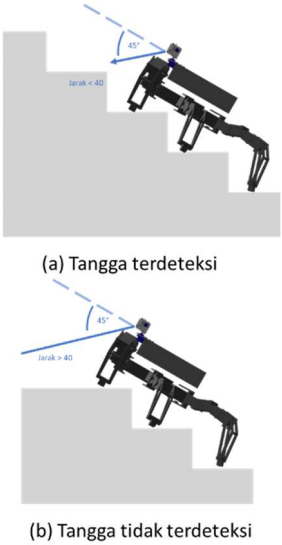
Gambar 3. 18 Ilustrasi pendeteksian tepian bawah tangga

Sebelum menaiki tangga robot berjalan menuju kearah tangga dengan tubuh robot yang sejajar dengan lantai. Deteksi tepian bawah tangga dilakukan dengan memutar servo dengan sudut 75° kebawah. Saat robot mendeteksi lantai maka jarak antara sensor dengan lantai lebih dari

25 cm dan saat mendeteksi anak tangga paling bawah nilai jarak sensor kurang dari 25 cm. Setelah dideteksi anak tangga pertama maka robot akan masuk ke fase pertama atau fase peralihan dari lantai bawah ke tangga. Illustrasi deteksi tepian bawah tangga dapat dilihat pada Gambar 3. 18.

2. Pendeteksian tepian atas tangga

Pada saat robot menaki tangga pada fase kedua, robot membentuk sudut 30° dengan bidang horizontal. Deteksi tepian pada tepian tangga paling atas dilakukan dengan memutar sensor sebesar 45° kebawah. Saat sensor menyorot ke anak tangga, sensor memiliki nilai kurang dari 40 cm dan saat sensor menyorot ke lantai atas maka sensor akan memiliki nilai lebih dari 40 cm.



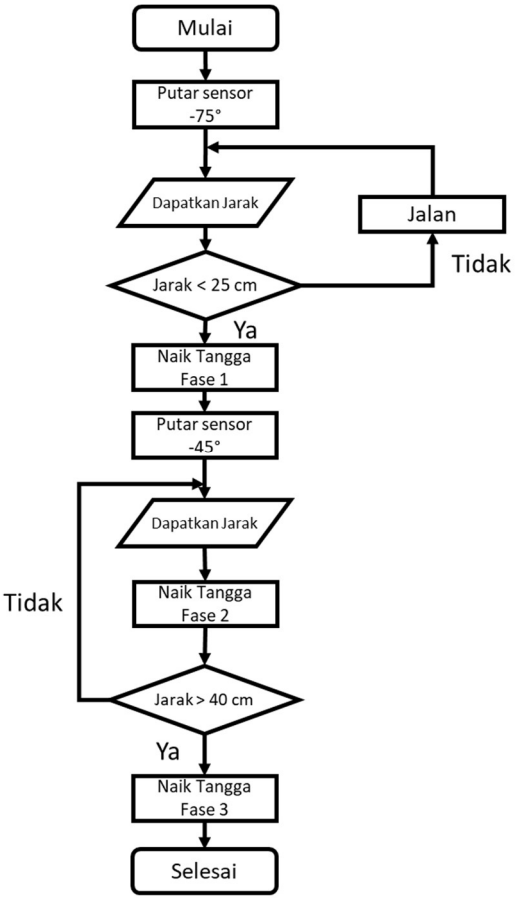
Gambar 3. 19 Illustrasi deteksi ujung tangga

3.5.2. Diagram alir algoritma melewati tangga

Pada Gambar 3. 20, digambarkan algoritma robot untuk menaiki tangga. Algoritma melewati tangga dimulai dengan memutar sensor sebesar -75° dan mendapatkan jarak dari sensor dan jika jarak masih lebih dari 25 cm maka robot akan terus berjalan hingga jarak robot kurang dari

25 cm. Saat sensor mendapatkan jarak kurang dari 25 cm maka dilakukan fase 1 menaiki tangga.

Setelah fase 1 selesai, robot akan memutar sensor sebesar -45° dan memulai fase 2 yang akan berulang sampai sensor bisa mendapatkan nilai yang lebih dari 40 cm. Saat didapatkan nilai lebih dari 40 cm dari robot, robot akan memulai fase 3 sebagai fase terakhir. Ketika fase 3 berakhir dan robot sudah berada lantai atas maka robot dianggap berhasil melewati tangga.



Gambar 3. 20 Diagram Alir Algoritma Melewati Tangga

3.5.3. Perancangan Gaya Berjalan pada Lantai

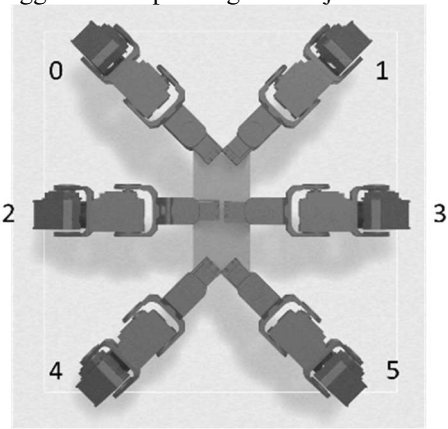
Sebelum robot mendeteksi tangga, robot akan menempuh jarak yang ditentukan hingga deteksi tangga terjadi dan beralih ke fase 1. Perancangan gerakan menggunakan cara berjalan 3+3 dimana 3 kaki menyangga dan 3 kaki mengayun seperti yang ditunjukkan pada Tabel 3. 2.

Tabel 3. 2 Gerakan pada Lantai

Kaki \ n	0	1
0		
1		
2		
3		
4		
5		

Mengayun Menyangga

Pada tabel tersebut, kaki merupakan indeks dari kaki robot yang merujuk kepada Gambar 3. 21 dan n adalah langkah-langkah dari awal hingga akhir siklus gerakan. Pada saat mengayun kaki robot akan naik dan bergerak ke depan dan pada saat menyangga kaki robot akan menarik kebelakang sehingga robot dapat bergerak maju.



Gambar 3. 21 Penomoran Kaki Robot

3.5.4. Perancangan Gaya Berjalan untuk Fase 1

Keberhasilan menaiki tangga didapatkan dari 3 fase gerakan pada robot. Setiap fase memiliki gerakan yang berbeda-beda sebab permukaan yang dilalui setiap fase juga berbeda. Permukaan pada fase pertama adalah lantai kemudian anak tangga pertama.

Pada Tabel 3. 3 ditunjukkan gerakan yang direncanakan dengan *n* sebagai penomoran langkah dari awal sampai akhir dan kaki pada tabel adalah indeks dari kaki robot yang dijelaskan pada Gambar 3. 21. Gerakan tersebut merepresentasikan kaki yang sedang menyangga dan kaki yang sedang mengayun.

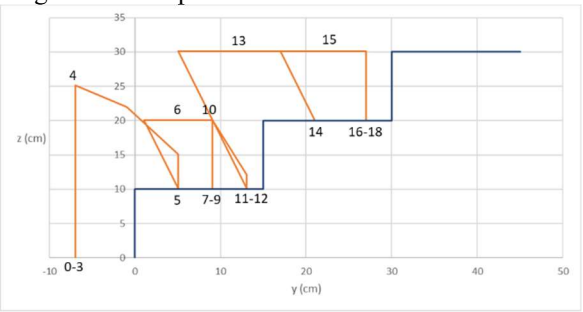
Tabel 3. 3 Gerakan pada Fase 1

Kaki \ n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0					■		■				■			■		■			
1				■		■						■			■		■		
2	■							■		■			■					■	
3		■						■		■			■					■	
4	■				■						■			■		■			
5		■				■						■			■		■		

■ Mengayun □ Menyangga

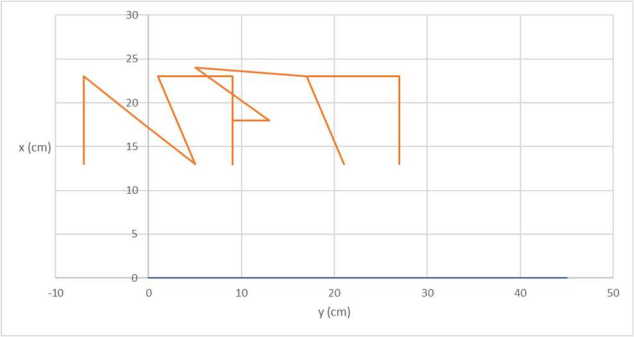
Berdasarkan tabel diatas, jalur yang ditempuh oleh kaki robot dapat ditampilkan seperti pada Gambar 3. 22 sampai Gambar 3. 33. Kedua belas gambar tersebut menunjukkan jalur yang ditempuh oleh setiap kaki robot pada bidang y-z dan bidang y-x.

1. Grafik gerakan kaki pada kaki 0



Gambar 3. 22 Jalur kaki 0 bidang y-z pada fase 1

Jalur kaki robot dirancang pada ruang 3 dimensi. Untuk mempermudah pemahaman dari jalur yang dirancang maka jalur digambarkan pada bidang y-z dan y-x. Pada Gambar 3. 22, ditunjukkan grafik dengan garis biru tua dan garis jingga. Garis biru tua pada grafik tersebut merupakan garis yang mewakili tangga dan garis jingga merupakan garis dari jalur dari kaki robot ke-0 yang direncanakan pada bidang y-z atau dari samping kanan.



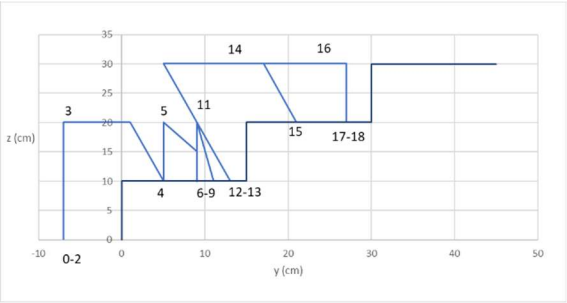
Gambar 3. 23 Jalur kaki 0 bidang y-x pada fase 1

Gambar 3. 22 dilengkapi oleh label yang merupakan nilai *n* dari Tabel 3. 3. Posisi kaki menyangga pada Tabel 3. 3 ditunjukkan pada Gambar 3. 22 berupa garis jingga yang bersinggungan dengan garis tangga dan sumbu x sedangkan posisi kaki yang mengayun ditunjukkan dengan garis jingga yang tidak bersinggungan dengan garis tangga maupun sumbu x. Kaki ke-0 memiliki jalur dengan titik awal dari lantai sampai ke anak tangga ke-2.

Pada Gambar 3. 23, ditunjukkan dua buah garis pada bidang y-x dengan warna yang sama. Garis biru tua merupakan garis yang mewakili garis tengah tangga yang tampak dari atas. Garis ini tidak mewakili bentuk tangga dari atas tetapi hanya untuk menyamakan posisi garis jalur kaki robot pada bidang y-z dan bidang y-x yang diletakkan secara bertumpuk atas dan bawah. Garis jingga pada Gambar 3. 23 juga mewakili jalur dari kaki robot 0 tetapi pada bidang y-x. Grafik jalur kaki robot pada bidang y-x tidak dilengkapi dengan label *n* dari Tabel 3. 3 karena posisi menyangga dan mengayun dari kaki hanya dapat telihat jelas pada grafik dengan sumbu z.

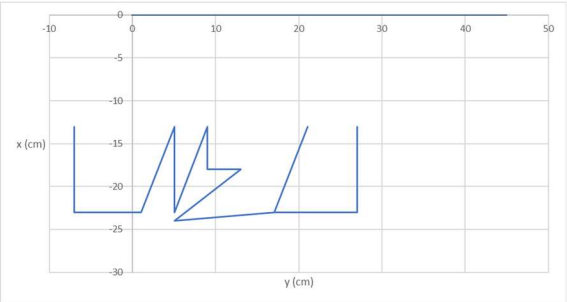
2. Grafik gerakan kaki pada kaki 1

Pada Gambar 3. 24, ditunjukkan grafik jalur kaki robot ke-1 pada bidang y-z. Jalur kaki ke-0 dan jalur kaki ke-1 memiliki kesamaan titik awal dan titik akhir pada fase 1 dan jumlah ayunan, yaitu 5 kali ayunan. Tetapi, jalur yang digunakan memiliki perbedaan karena respon dari servo yang didapatkan dari simulasi V-REP membuat jalur yang digunakan pada kaki ke-0 harus disesuaikan. Jalur kaki ke-0 memiliki posisi yang lebih tinggi saat mengayun pada $n = 4$ dari pada jalur kaki ke-1 pada saat $n = 3$ dan memiliki beberapa perbedaan pada ayunan $n = 6$ dan $n = 10$ pada kaki ke-0 atau ayunan $n = 5$ dan $n = 11$ pada kaki ke-1.



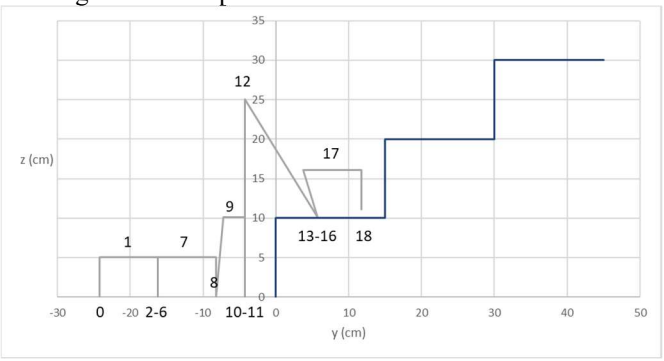
Gambar 3. 24 Jalur kaki 1 bidang y-z pada fase 1

Gambar 3. 25 merupakan grafik jalur dari kaki ke-0 pada fase 1. Grafik pada jalur kaki ke-1 ini dirancang dengan menyesuaikan dengan posisi dari ujung kaki dari bidang y-z agar tidak telalu pendek atau terlalu panjang untuk dijangkau kaki robot.



Gambar 3. 25 Jalur kaki 1 bidang y-x pada fase 1

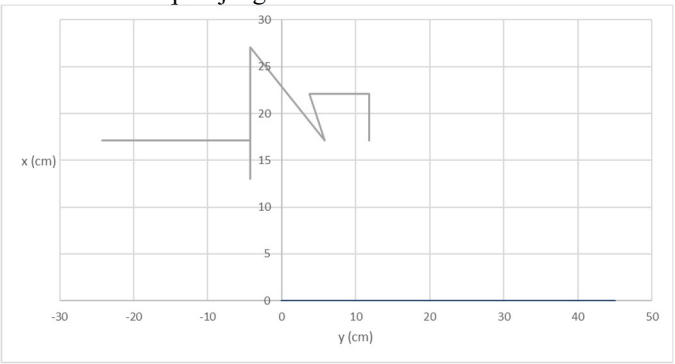
3. Grafik gerakan kaki pada kaki 2



Gambar 3. 26 Jalur kaki 2 bidang y-z pada fase 1

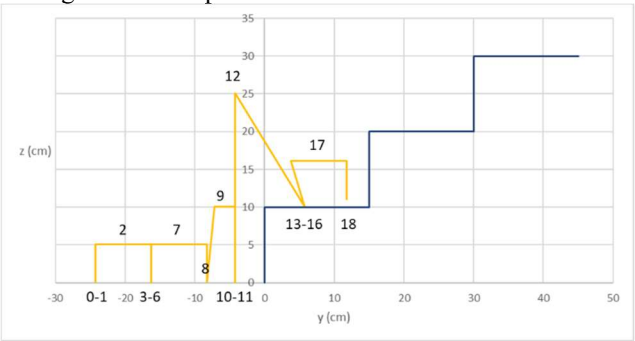
Jalur dari kaki ke-2 dan kaki ke-3 bergerak dengan mengayun dan menyangga secara bersamaan mulai dari n = 4 sampai dengan n = 18 seperti yang ditunjukkan pada Gambar 3. 28 dan Gambar 3. 30. Pada n = 0 sampai n = 3 robot berjalan dengan gaya 4+2 dimana kaki ke-2 bergerak bersamaan dengan kaki ke-4 dan kaki ke-3 bergerak bersamaan dengan kaki ke-5 untuk menyeimbangkan robot dengan membawa tubuh robot lebih ke depan. Titik awal dari kaki ke-2 dan kaki ke-3 adalah dari lantai dan titik akhirnya adalah pada anak tangga pertama.

Nilai x pada jalur kaki ke-2 dan ke-3 pada bidang y-x yang ditunjukkan pada Gambar 3. 27 dan Gambar 3. 29 memiliki fungsi yang sama seperti kaki ke-0 dan kaki ke-1 yaitu untuk membuat posisi dari *invers* kinematik tetap terjangkau oleh kaki.

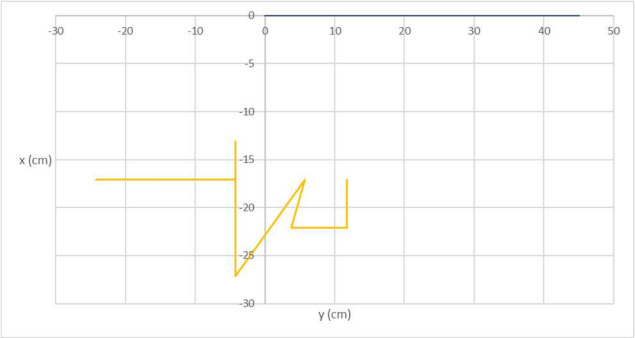


Gambar 3. 27 Jalur kaki 2 bidang y-x pada fase 1

4. Grafik gerakan kaki pada kaki 3

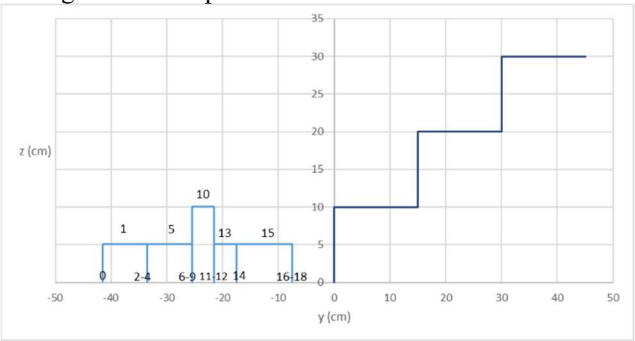


Gambar 3. 28 Jalur kaki 3 bidang y-z pada fase 1

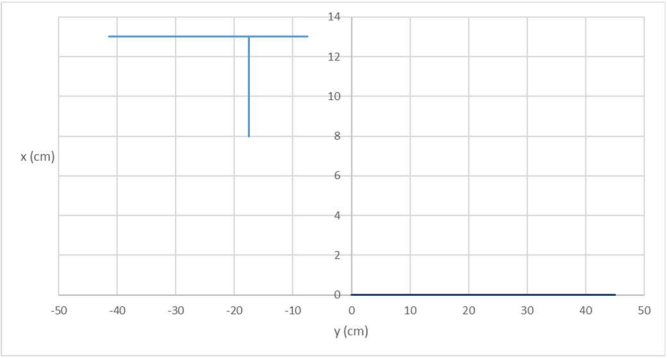


Gambar 3. 29 Jalur kaki 3 bidang y-x pada fase 1

5. Grafik gerakan kaki pada kaki 4



Gambar 3. 30 Jalur kaki 5 bidang y-z pada fase 1

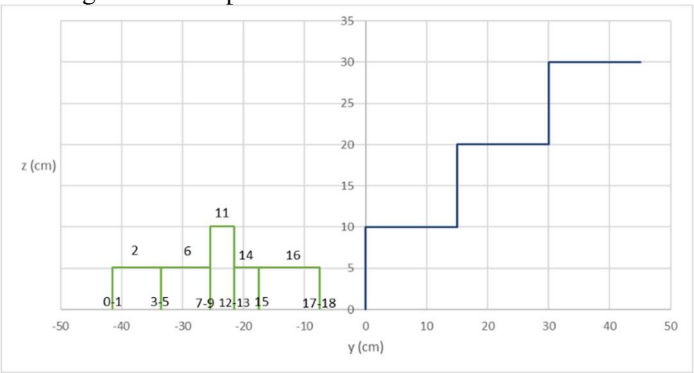


Gambar 3. 31 Jalur kaki 5 bidang y-x pada fase 1

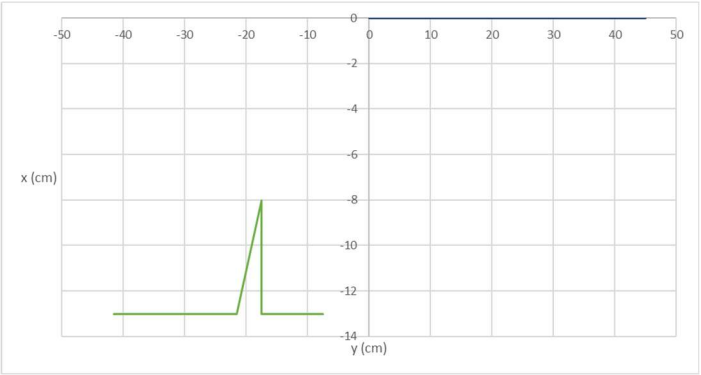
Kaki ke-4 dan kaki ke-5 merupakan kaki yang teletak paling belakang dan paling bawah pada saat akhir fase 1. Kedua kaki ini memiliki pengaruh yang cukup besar pada kestabilan robot pada fase 1. Pergerakan dari kaki ke-4 dan ke-5 perlu diatur sedemikian rupa agar robot tidak jatuh ke belakang seperti yang ditunjukkan pada Gambar 3. 30 dan Gambar 3. 32.

Seperti kaki yang lain, nilai x dari jalur kaki ke-4 dan kaki ke-5 memiliki fungsi untuk membuat titik dari *invers* kinematik dalam jangkauan kaki robot.

6. Grafik gerakan kaki pada kaki 5



Gambar 3. 32 Jalur kaki 5 bidang y-z pada fase 1



Gambar 3. 33 Jalur kaki 5 bidang y-x pada fase 1

3.5.5. Perancangan Gaya Berjalan untuk Fase 2

Pada perencanaan naik tangga fase 2 digunakan 2 macam cara jalan, yaitu 4+2 dan 3+3. Gaya jalan 4+2 adalah 2 kaki mengayun dan 4 kaki menyangga robot dan gaya jalan 3+3 adalah 3 kaki mengayun dan 3 kaki menyangga. Penggunaan cara jalan 4+2 digunakan untuk mendapatkan kestabilan yang lebih baik daripada cara jalan 3+3 karena lebih banyak kaki yang menyangga. Penggunaan cara jalan 3+3 memiliki durasi siklus yang lebih singkat, yaitu 2 langkah sedangkan cara jalan 4+2 memiliki durasi 3 langkah.

Pada fase ke 2, robot berjalan diatas tangga secara berulang-ulang hingga sistem deteksi memicu robot untuk menghentikan fase 2 dan melanjutkan ke fase 3.

Tabel 3. 4 Gerakan pada fase 2

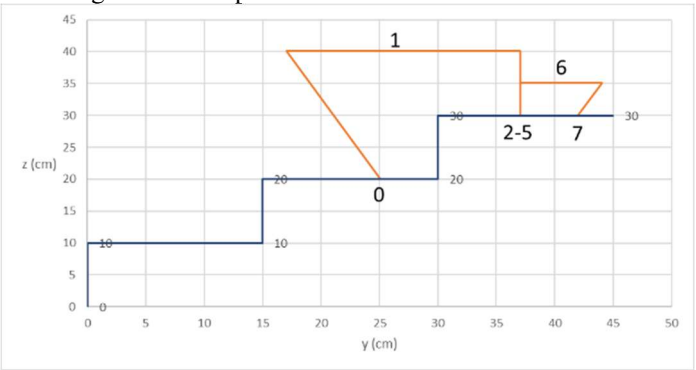
Kaki \ n	0	1	2	3	4	5	6	7
0		■					■	
1			■			■		
2				■		■		
3				■			■	
4		■					■	
5			■			■		

■ Mengayun □ Menyangga

Berdasarkan tabel diatas, jalur yang ditempuh oleh kaki robot dapat ditampilkan seperti pada Gambar 3. 34 sampai dengan Gambar 3.

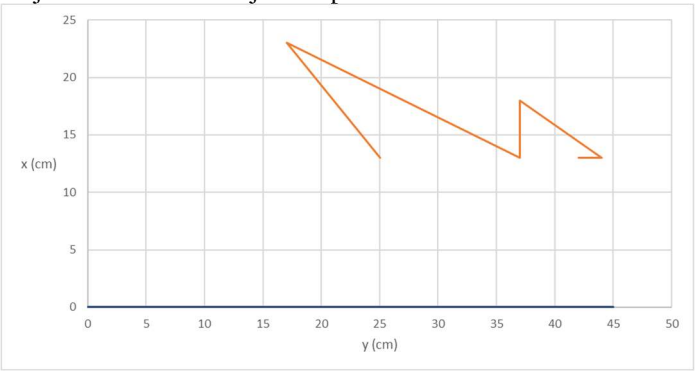
45. Kedua belas gambar tersebut menunjukkan jalur yang ditempuh oleh setiap kaki robot pada bidang y-z dan bidang y-x.

1. Grafik gerakan kaki pada kaki 0



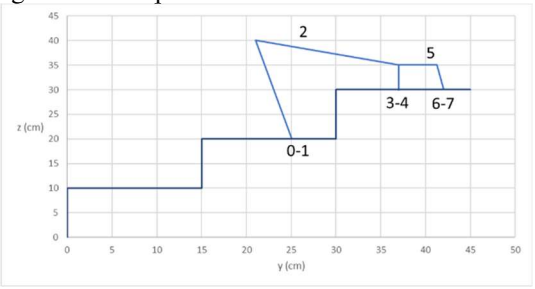
Gambar 3. 34 Jalur kaki 0 bidang y-z pada fase 2

Grafik jalur gerakan dari kaki ke-0 dan kaki ke-1 pada bidang y-z yang ditunjukkan pada Gambar 3. 34 dan Gambar 3. 36. Jalur kaki ke-0 memiliki ayunan yang sedikit lebih panjang daripada jalur kaki ke-1 karena kaki ke-0 yang bergerak terlebih dahulu memerlukan ruang yang lebih luas agar kaki tidak terbentur tangga. Setelah kaki telah menyangga pada anak tangga berikutnya, robot menjadi cukup stabil untuk memberikan ruang pada kaki ke-1 untuk bergerak. Nilai x terhadap nilai y pada jalur kaki ke-0 ditunjukkan pada Gambar 3. 35 dan Gambar 3. 37.

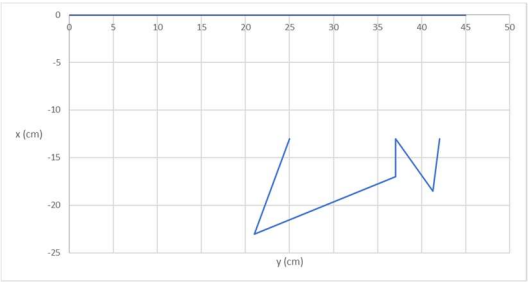


Gambar 3. 35 Jalur kaki 0 bidang y-x pada fase 2

2. Grafik gerakan kaki pada kaki 1

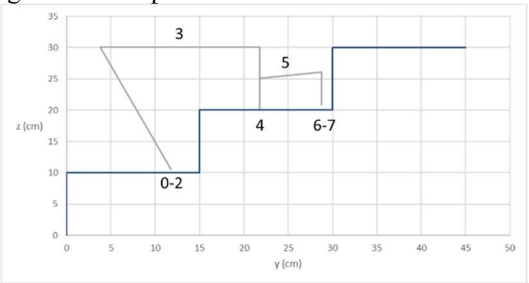


Gambar 3. 36 Jalur kaki 1 bidang y-z pada fase 2



Gambar 3. 37 Jalur kaki 1 bidang y-x pada fase 2

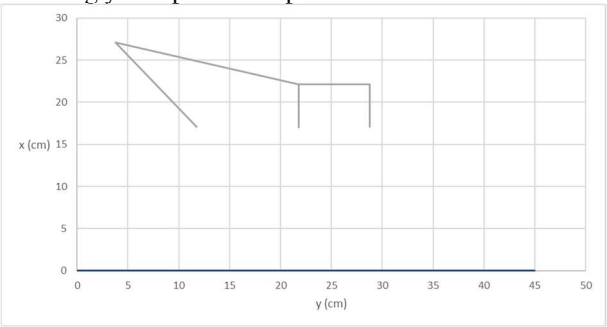
3. Grafik gerakan kaki pada kaki 2



Gambar 3. 38 Jalur kaki 2 bidang y-z pada fase 2

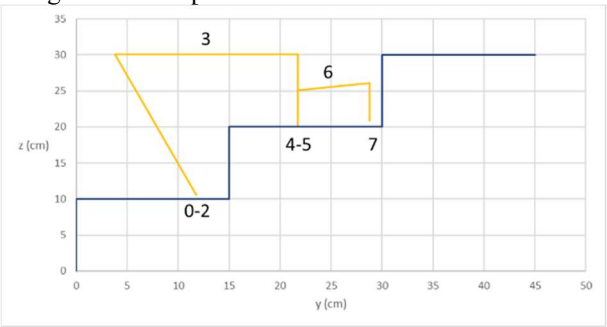
Jalur kaki ke-2 dan ke-3 memiliki jalur yang sama dengan nilai x yang dibalik tetapi dengan ayunan dan penyanggan diwaktu yang berbeda. Titik akhir dari jalur kaki ke-2 dan ke-3 adalah tangga berikutnya. Grafik jalur kaki ke-2 dan ke-3 pada bidang y-z dapat dilihat

pada Gambar 3. 38 dan Gambar 3. 40. Untuk grafik jalur kaki ke-2 dan ke-3 pada bidang y-x dapat dilihat pada Gambar 3. 39 dan Gambar 3. 41.

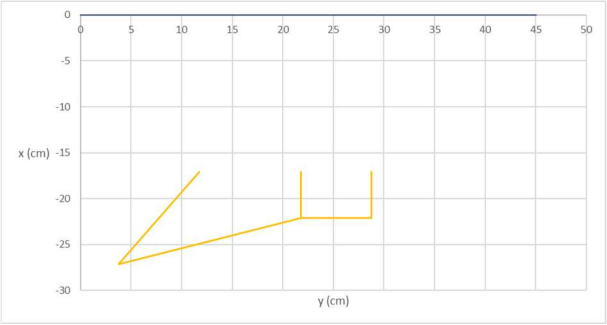


Gambar 3. 39 Jalur kaki 2 bidang y-x pada fase 2

4. Grafik gerakan kaki pada kaki 3

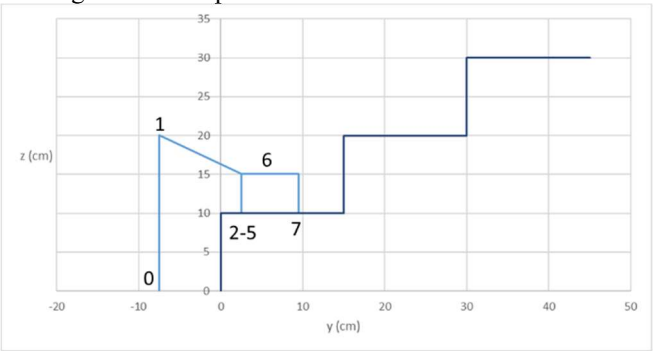


Gambar 3. 40 Jalur kaki 3 bidang y-z pada fase 2

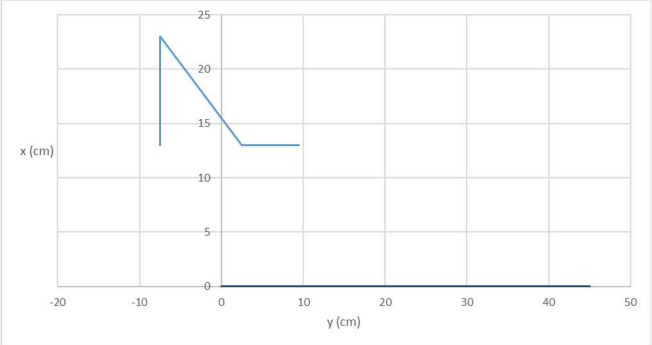


Gambar 3. 41 Jalur kaki 3 bidang y-z pada fase 2

5. Grafik gerakan kaki pada kaki 4



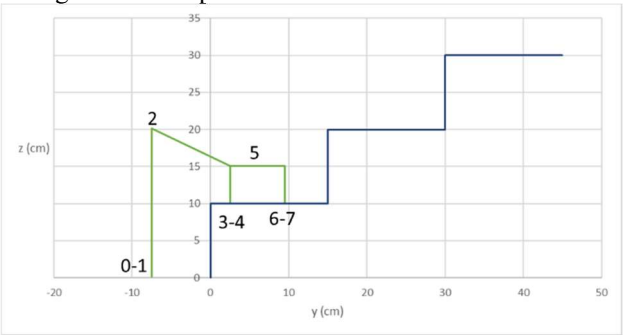
Gambar 3. 42 Jalur kaki 4 bidang y-z pada fase 2



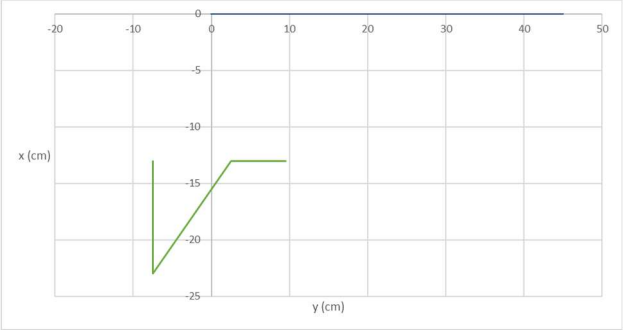
Gambar 3. 43 Jalur kaki 4 bidang y-z pada fase 2

Jalur kaki ke-4 dan ke-5 memiliki jalur kaki yang sama juga dengan nilai x yang balik untuk menyesuaikan posisinya. Tujuan dari jalur kaki ke-4 dan ke-5 adalah tangga berikutnya. Grafik jalur kaki ke-2 dan ke-3 pada bidang y-z dapat dilihat pada Gambar 3. 42 danGambar 3. 44. Untuk grafik jalur kaki ke-2 dan ke-3 pada bidang y-x dapat dilihat pada Gambar 3. 43 dan Gambar 3. 45.

6. Grafik gerakan kaki pada kaki 5



Gambar 3. 44 Jalur kaki 5 bidang y-z pada fase 2



Gambar 3. 45 Jalur kaki 5 bidang y-z pada fase 2

3.5.1. Perancangan Gaya Berjalan untuk Fase 3

Tabel 3. 5 Gaya Berjalan pada Fase 3

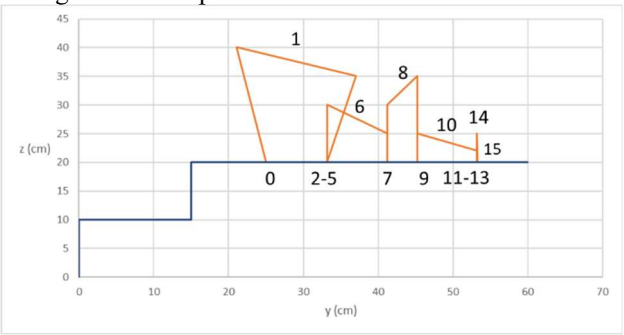
Kaki \ n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0		■					■		■		■				■	
1			■			■		■				■		■		
2				■		■		■						■		
3				■			■		■						■	
4		■					■		■		■					
5			■			■		■				■				

■ Mengayun □ Menyangga

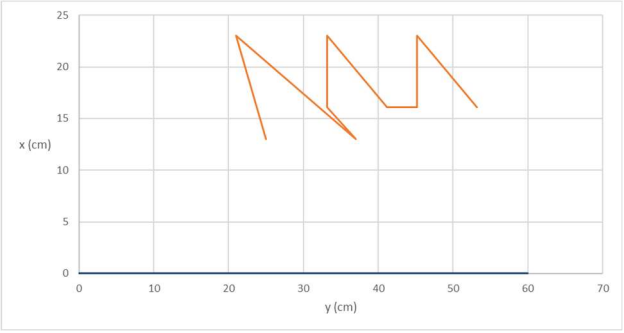
Fase ketiga merupakan fase terakhir dimana robot telah mendeteksi ujung dari tangga. Robot akan beralih dari siklus naik tangga atau fase kedua dan naik ke lantai di atas tangga. Gerakan robot pada setiap langkahnya ditampilkan pada Tabel 3. 5.

Berdasarkan tabel tersebut, jalur yang ditempuh oleh kaki robot dapat ditampilkan seperti pada Gambar 3. 46 sampai Gambar 3. 57. Kedua belas gambar tersebut menunjukkan jalur yang ditempuh oleh setiap kaki robot pada bidang y-z dan bidang y-x.

1. Grafik gerakan kaki pada kaki 0



Gambar 3. 46 Jalur kaki 0 bidang y-z pada fase 3



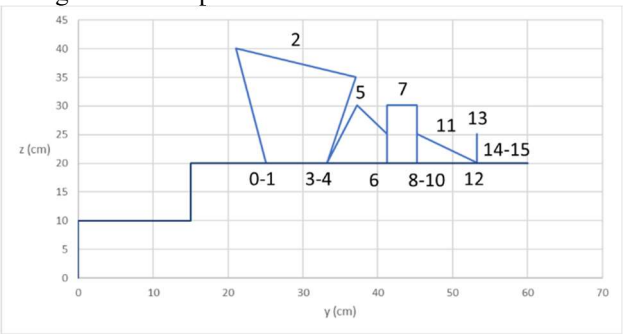
Gambar 3. 47 Jalur kaki 0 bidang y-x pada fase 3

Pada fase 3, kaki ke-0 dan kaki ke-1 berada pada lantai atas pada titik awal dan titik akhir dari jalur yang direncanakan. Kedua kaki tersebut melangkah kedepan untuk memberikan ruang pada keempat kaki lainnya di lantai atas tangga. Gerakan mengayun pada $n = 14$ pada kaki ke-0 dan

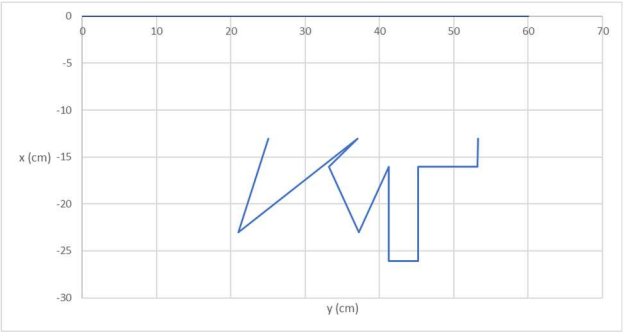
n = 13 pada kaki ke-1 merupakan gerakan terakhir dimana pada gerakan ini direncanakan dengan gaya berjalan 4+2 untuk kaki ke-0, 1, 2, dan 3. Perpindahan yang terjadi pada *n* = 13 dan 14 yang sangat kecil sebesar 0.04 cm sehingga terlihat seperti garis. Ayunan terakhir tersebut dimaksudkan agar pose robot dapat menjadi pose awal dimana semua servo sudutnya adalah 0°. Grafik jalur kaki pada kaki ke-0 dan kaki ke-1 ditunjukkan pada Gambar 3. 48 dan Gambar 3. 50.

Nilai x pada jalur kaki ke-0 dan ke-1 pada bidang y-x yang ditunjukkan pada Gambar 3. 49 dan Gambar 3. 51 memiliki fungsi yang sama seperti kaki ke-0 dan kaki ke-1 yaitu untuk membuat posisi dari masukan *invers* kinematik tetap terjangkau oleh kaki robot.

2. Grafik gerakan kaki pada kaki 1

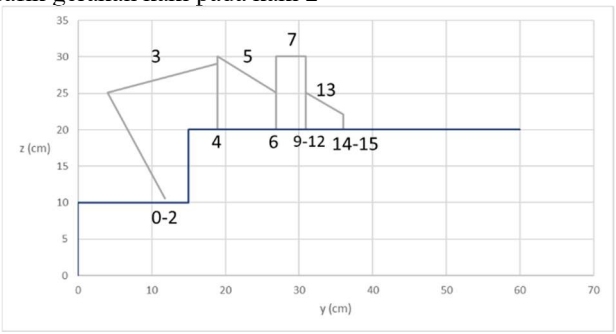


Gambar 3. 48 Jalur kaki 1 bidang y-z pada fase 3



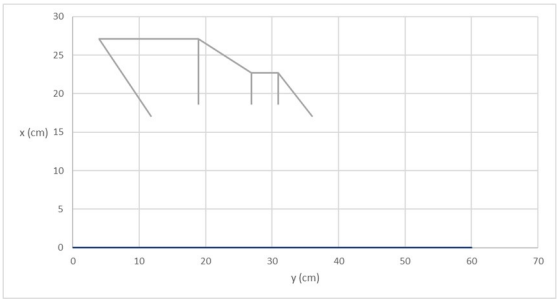
Gambar 3. 49 Jalur kaki 1 bidang y-x pada fase 3

3. Grafik gerakan kaki pada kaki 2



Gambar 3. 50 Jalur kaki 2 bidang y-z pada fase 3

Pada kaki ke-2 dan kaki ke-3, jalur kaki direncanakan untuk bergerak dari anak tangga kedua dari atas kemudian naik ke lantai atas dan maju hingga terdapat ruang bagi dua kaki belakang robot di atas lantai. Jalur kaki ke-2 dan ke-3 memiliki 4 kali ayunan dengan jalur yang hampir sama dengan waktu yang berbeda. Perbedaan pada kedua jalur kaki tersebut ada pada $n = 7$ di jalur kaki ke-2 dan $n = 8$ di jalur kaki ke-3. Perbedaan tersebut ada disebabkan oleh perubahan perancangan jalur yang belum diterapkan oleh penulis dan tidak memiliki tujuan tertentu serta kecil pengaruhnya terhadap keseimbangan robot sebab robot telah disangga oleh tiga kaki lainnya. Grafik dari jalur kaki ke-2 dan ke-3 dapat dilihat pada Gambar 3. 50 dan Gambar 3. 52.

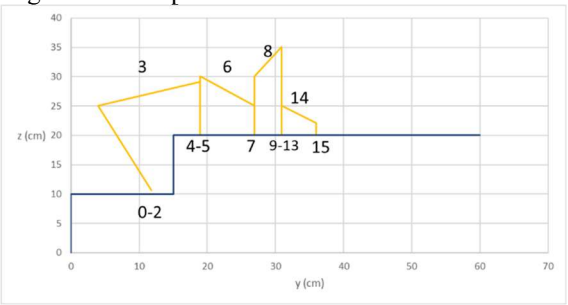


Gambar 3. 51 Jalur kaki 2 bidang y-x pada fase 3

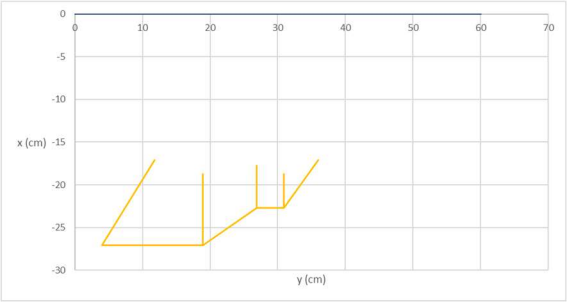
Grafik jalur kaki ke-2 dan kaki ke-3 pada bidang y-x dapat dilihat pada Gambar 3. 51 dan Gambar 3. 53. Nilai x dari kaki ke-2 dan kaki ke-

3 merupakan kebalikan dan perancangannya didapatkan dari penyesuaian agar titik *invers* kinematik dalam jangkauan robot.

4. Grafik gerakan kaki pada kaki 3

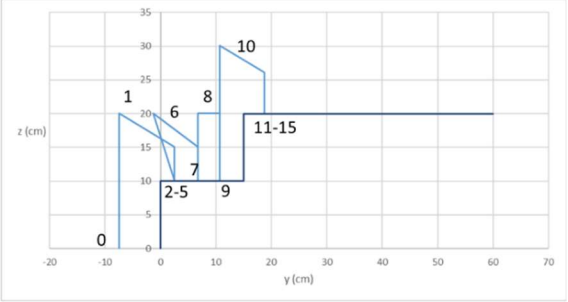


Gambar 3. 52 Jalur kaki 3 bidang y-z pada fase 3



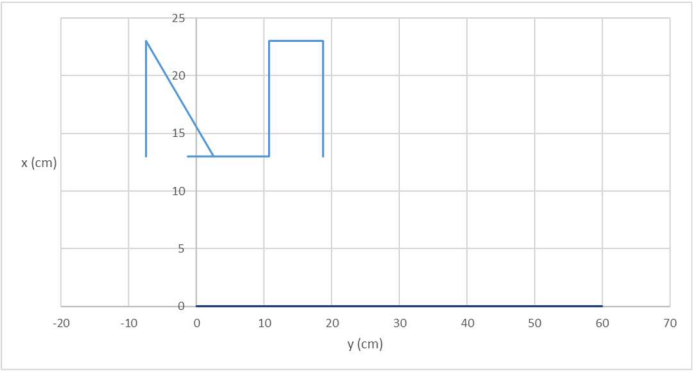
Gambar 3. 53 Jalur kaki 3 bidang y-x pada fase 3

5. Grafik gerakan kaki pada kaki 4



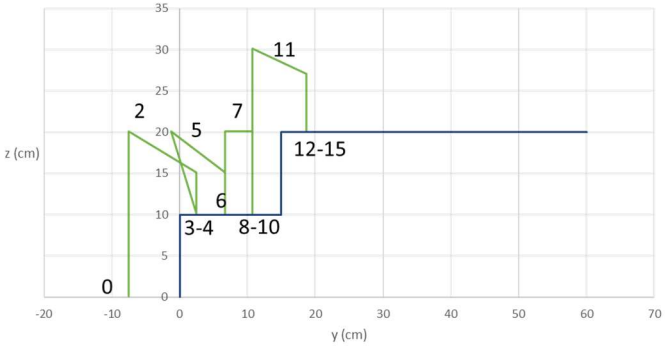
Gambar 3. 54 Jalur kaki 4 bidang y-z pada fase 3

Grafik dari jalur kaki ke-4 dan ke-5 dapat dilihat pada Gambar 3. 54 dan Gambar 3. 56. Kaki ke-4 dan ke-5 memiliki jalur yang sama pada bidang y-z sedangkan pada bidang y-x nilai yang dimiliki adalah berkebalikan. Grafik jalur kaki ke-4 dan kaki ke-5 pada bidang y-x dapat dilihat pada Gambar 3. 55 dan Gambar 3. 57.

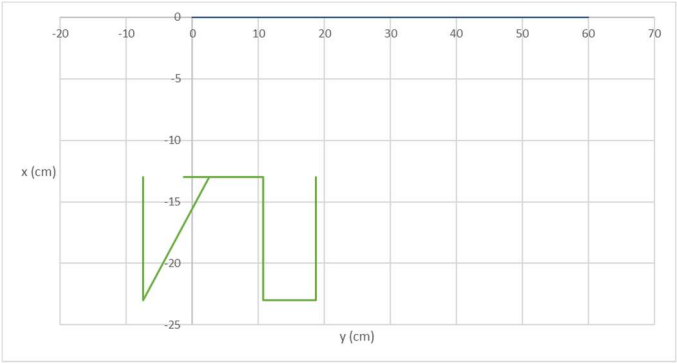


Gambar 3. 55 Jalur kaki 4 bidang y-x pada fase 3

6. Grafik gerakan kaki pada kaki 5



Gambar 3. 56 Jalur kaki 5 bidang y-z pada fase 3



Gambar 3. 57 Jalur kaki 5 bidang y-x pada fase 3

Halaman ini sengaja dikosongkan

BAB IV
PENGUJIAN DAN ANALISIS

Pada bab ini dibahas mengenai pengujian dan analisis dari sistem yang telah dirancang dan dijelaskan pada bab III. Pengujian yang akan dilakukan meliputi pengujian deteksi tepian tangga, pengujian algoritma pada lingkungan simulasi dan pengujian pada robot purwarupa.

4.1. Pengujian Deteksi Tepi Tangga

Pada pengujian ini, dilakukan dua macam pengujian yaitu pengujian deteksi tepi bawah tangga dan tepi atas tangga. Berikut adalah pengujian deteksi tepi bawah tangga dan tepi atas tangga:

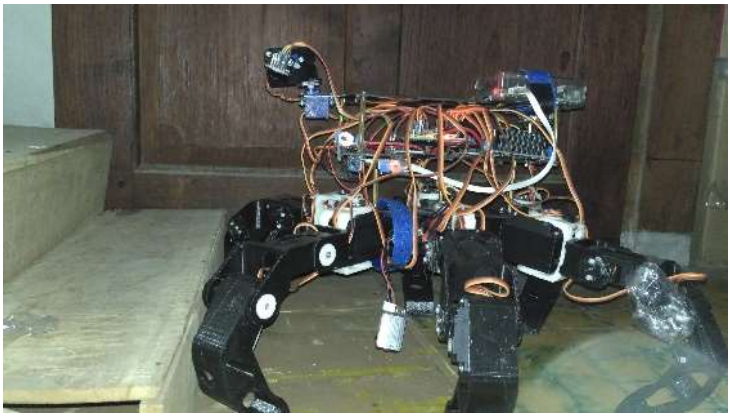
4.1.1. Pengujian Deteksi Tepi Bawah Tangga

Pada pendeteksian tepi bawah tangga, sensor diputar oleh servo dengan sudut sebesar 75° kebawah dan robot berada dalam pose berjalan diatas lantai. Tepi bawah tangga atau anak tangga pertama akan terdeteksi jika sensor jarak berada diatas anak tangga pertama. Berikut adalah tabel pengujian deteksi tepi bawah tangga:

Tabel 4. 1 Pengujian Deteksi Tepi Bawah Tangga

No.	Jarak dengan tangga (cm)	Output Sensor (cm)	Output (< 25 cm)
1	0	22.4	Ya
2	0	22.5	Ya
3	0	22.3	Ya
4	3	23.6	Ya
5	3	24.1	Ya
6	3	25	Tidak
7	5	29.8	Tidak
8	5	29.1	Tidak
9	5	30	Tidak

Berdasarkan tabel pengujian diatas, sistem akan mendeteksi tepi bawah tangga jika jarak antara robot dengan tangga kurang dari sama dengan 3 cm.



Gambar 4. 1 Pengujian deteksi tepi bawah tangga

4.1.2. Pengujian Deteksi Tepi Atas Tangga

Gambar 4. 2 Pengujian deteksi tepi atas tangga

Pada pengujian ini, pose robot diatur seperti pada saat fase 2 dan memutar sensor sebesar 45° kebawah. Pengujian deteksi tepi atas dilakukan dengan meletakkan robot pada tangga dengan kaki depan berada di anak tangga ke 2 sampai anak tangga ke 6 atau paling atas. Hasil dari pengujian deteksi tepi atas tangga dapat dilihat pada Tabel 4. 2 dibawah ini.

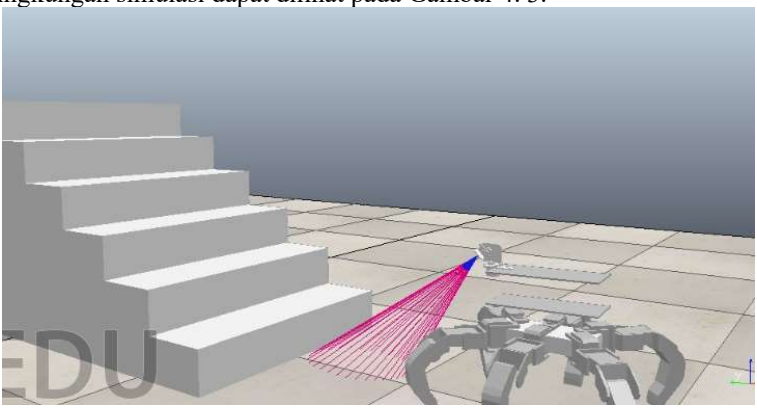
Tabel 4. 2 Hasil deteksi tepi atas tangga

Anak Tangga	Output Sensor (cm)	Output (>40 cm)
2	30.6 – 31.2	Tidak
3	28.3 – 29.5	Tidak
4	24.1 – 23.8	Tidak
5	27.3 – 34.7	Tidak
6	41.5 – 43.8	Ya

Hasil dari pengujian menunjukkan bahwa pendeteksian berhasil karena deteksi hanya terjadi pada saat robot berada di atas tangga atau anak tangga teratas.

4.2. Pengujian Naik Tangga di Lingkungan Simulasi

Pengujian kedua adalah pengujian algoritma naik tangga yang telah dibuat di lingkungan simulasi V-rep. Tata letak robot dan tangga pada lingkungan simulasi dapat dilihat pada Gambar 4. 3.

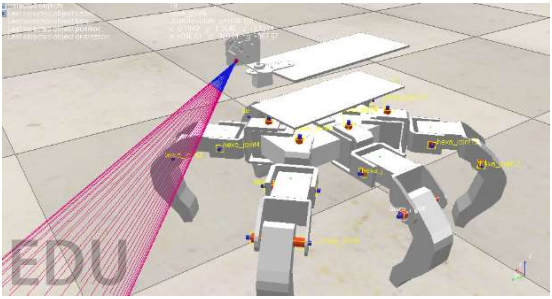


Gambar 4. 3 Situasi Pengujian Simulasi Naik Tangga

Model robot yang digunakan pada simulasi pada V-rep merupakan versi *low poly* atau resolusi rendah dari model yang digunakan untuk kerangka robot seperti yang dapat dilihat pada Gambar 4. 4 Model robot simulasi. Servo atau *joint* pada model robot simulasi diletakkan pada tempat yang sama seperti pada model asli robot. Sensor jarak atau *proximity sensor* pada V-rep digunakan untuk mensimulasikan sensor VL53L0X. Model tangga pada lingkungan simulasi juga telah

disesuaikan dengan ukuran tangga yang digunakan pada pengujian robot purwarupa.

Pada pengujian simulasi ini disesuaikan parameter massa robot dan spesifikasi motor servo. Massa dari kerangka robot diatur menjadi 2.5 kg, torsi servo diatur 8 kgfcm atau 0.78 Nm, dan kecepatan servo diatur 60°/0.2 detik atau 300°/detik. Torsi dari servo telah dikurangi dari rating yang diberikan sebesar 9.4 untuk meningkatkan keyakinan dari simulasi yang dilakukan.



Gambar 4. 4 Model robot simulasi

Hal yang diuji pada pengujian ini adalah keberhasilan robot dalam naik tangga dengan jarak yang diberikan dari tangga. Robot dianggap berhasil naik tangga jika robot berhasil melakukan perjalanan dari lantai bawah ke lantai yang berada di atas tangga. Hasil dari pengujian tersebut dicatat pada Tabel 4. 3 di bawah ini.

Tabel 4. 3 Pengujian di lingkungan simulasi

No.	Jarak dengan Tangga (cm)	Keberhasilan	Waktu (menit:detik)
1	15	Berhasil	2:40
2	20	Berhasil	2:42
3	25	Berhasil	2:41
4	30	Berhasil	2:43
5	35	Berhasil	2:46
6	40	Berhasil	2:47
7	45	Berhasil	2:46
8	50	Berhasil	2:53

Berdasarkan tabel pengujian diatas, pengujian algoritma naik tangga yang dilakukan pada lingkungan simulasi berhasil dilakukan dengan waktu tempuh antara 2 menit 40 detik sampai 2 menti 53 detik.

4.3. Pengujian Algoritma Naik Tangga dengan Robot Purwarupa

Pada pengujian ini, robot purwarupa yang telah diprogram dengan algoritma yang telah dirancang akan diuji untuk menaiki tangga. Pengujian ini dilakukan untuk membuktikan algoritma dan gerakan yang telah dirancang. Pengujian dilakukan dengan meletakkan robot dengan jarak 15 sampai 50 cm dan mengarahkannya ketangga.

Gambar 4. 5 Situasi pengujian pada robot purwarupa

Berikut adalah tabel pengujian algoritma naik tangga dengan robot purwarupa.

Tabel 4. 4 Hasil pengujian kedua pada robot purwarupa

No.	Jarak dengan Tangga (cm)	Keberhasilan	Waktu (menit:detik)
1	15	Berhasil	2:49
2	20	Berhasil	2:50
3	25	Berhasil	2:53
4	30	Berhasil	2:55
5	35	Berhasil	3:02
6	40	Berhasil	3:04
7	45	Gagal	1:26
8	50	Gagal	1:29

Pada pengujian ketiga ini, robot berhasil untuk menaiki tangga dari jarak 15 cm sampai dengan 40 cm dari tangga. Pada jarak 45 cm dan 50 cm, robot mengalami jatuh dan gagal pada saat memulai fase 2 dan di anak tangga ke empat. Kaki robot kelima yang seharusnya sudah menginjak anak tangga kedua pada n = 2 berdasarkan Tabel 3. 4 di menit 1:26 dan 1:29 masih tergantung di udara yang menyebabkan robot jatuh ke belakang. Kegagalan kaki kelima yang tidak dapat menginjak anak tangga kedua ini disebabkan oleh kemiringan robot dari awal menaiki tangga.

Pada jarak robot terhadap tangga 45 cm dan 50 cm, robot memiliki kemiringan saat robot sampai dengan anak tangga pertama. Kemiringan ini menyebabkan robot untuk bergerak naik tangga dengan kemiringan yang sama dan menyebabkan robot jatuh dan gagal dalam menaiki tangga dengan gerakan yang telah dirancang. Dari percobaan yang dilakukan algoritma dan gerakan yang dirancang hanya bisa berhasil menaiki tangga saat robot lurus terhadap tangga dari awal gerakan.

Kemiringan robot pada awal menaiki tangga disebabkan oleh gerakan berjalan di atas lantai yang tidak lurus. Gerakan berjalan di atas lantai pada robot yang telah dirancang untuk bergerak lurus ke depan pada kenyataannya masih berbelok dan menjadi sangat berpengaruh terhadap hasil dari robot dalam menaiki tangga terutama pada jarak 45 cm dan 50 cm. Gerakan berjalan robot ini berbelok disebabkan oleh tidak adanya koreksi kesalahan yang dilakukan untuk mengatasi kemiringan yang ada pada gerakan berjalan tersebut.

BAB V
PENUTUP

5.1. Kesimpulan

Setelah dilakukan pengujian dan analisis pada bab iv, didapatkan kesimpulan sebagai berikut:

1. Gerakan dan algoritma robot untuk menaiki tangga yang dirancang berhasil dengan jarak robot dari tangga 15 cm, 20 cm, 25 cm, 30 cm, 35 cm, dan 40 cm.
2. Gerakan dan algoritma robot hanya dapat berhasil jika posisi robot lurus dalam menaiki tangga. Hal ini ditunjukkan pada percobaan dengan jarak robot dari tangga 45 cm dan 50 cm yang gagal disebabkan oleh kemiringan robot pada awal gerakan menaiki tangga.
3. Gerakan dan algoritma tidak bersifat adaptif karena perancangan dan pengujian dilakukan hanya untuk ukuran anak tangga 10 cm x 15 cm x 50 cm.

5.2. Saran

Gerakan robot dalam berjalan di atas lantai yang tidak lurus menyebabkan kegagalan pada percobaan dengan jarak antara robot dengan tangga 45 cm dan 50 cm. Gerakan berjalan di atas robot perlu dibenahi agar robot dapat berjalan lurus dari titik awal sampai di depan tangga. Perbaikan ini meliputi perancangan robot dengan melibatkan sensor arah seperti kompas untuk mengoreksi kesalahan arah dari robot.

Gerakan yang telah dirancang juga masih belum adaptif karena hanya dirancang dan diuji untuk ukuran anak tangga 10 cm x 15 cm x 50 cm dimana ukuran ini adalah setengah dari ukuran anak tangga aslinya. Diharapkan gerakan dan algoritma pada tugas akhir ini dapat dikembangkan lebih lanjut agar dapat bersifat adaptif terhadap ukuran anak tangga yang berbeda.

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

[1] "Legacy Robots | Boston Dynamics." https://www.bostondynamics.com/legacy (diakses Jul 15, 2020).

[2] "Mars Exploration Rovers - NASA Mars." https://mars.nasa.gov/mer/ (diakses Jul 15, 2020).

[3] J. Niu *dkk.*, "Study on structural modeling and kinematics analysis of a novel wheel-legged rescue robot," *International Journal of Advanced Robotic Systems*, vol. 15, no. 1, hlm. 1729881417752758, Jan 2018, doi: 10.1177/1729881417752758.

[4] "Robot Platform | Knowledge | Types of Robot Wheels." http://www.robotplatform.com/knowledge/Classification_of_Robots/Types_of_robot_wheels.html#standard_wheel (diakses Jul 10, 2020).

[5] "Robot Platform | Knowledge | Wheeled Robots." http://www.robotplatform.com/knowledge/Classification_of_Robots/legged_robots.html (diakses Nov 03, 2019).

[6] M. F. Silva dan J. T. Machado, "A literature review on the optimization of legged robots," *Journal of Vibration and Control*, vol. 18, no. 12, hlm. 1753–1767, Okt 2012, doi: 10.1177/1077546311403180.

[7] X. Ding, Z. Wang, A. Rovetta, dan J. M. Zhu, "Locomotion Analysis of Hexapod Robot," 2010, doi: 10.5772/8822.

[8] F. Tedeschi dan G. Carbone, "Design Issues for Hexapod Walking Robots," *Robotics*, vol. 3, no. 2, hlm. 181–206, Jun 2014, doi: 10.3390/robotics3020181.

[9] S. Kucuk dan Z. Bingul, "Robot Kinematics: Forward and Inverse Kinematics," 2006.

[10] J. W. Kusuma, "PENERAPAN INVERS KINEMATIK TERHADAP PERGERAKAN KAKI PADA ROBOT HEXAPOD," hlm. 10.

[11] "Distance Sensing - SparkFun Electronics." https://www.sparkfun.com/distance_sensing (diakses Mei 26, 2020).

[12] "Amazon.com : VL53L0X Time-of-Flight Distance Sensor Breakout VL53L0XV2 Module for Arduino : Camera & Photo." https://www.amazon.com/VL53L0X-Flight-Distance-Breakout-VL53L0XV2/dp/B07168QP71 (diakses Jul 15, 2020).

[13] "TowerPro MG995 Servo Motor High quality , High torque 360degree," *SchemoBotics*. http://schemobotics.com/product/towerpro-mg995-servo-motor-high-quality-high-torque-360degree/ (diakses Jul 15, 2020).
[14] "How Servo Motors Work | Servo Motor Controllers." https://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html (diakses Jul 15, 2020).
[15] "Servo Motor SG-90 Basics, Pinout, Wire Description, Datasheet." https://components101.com/servo-motor-basics-pinout-datasheet (diakses Jul 15, 2020).
[16] A. P. Shende dan R. S. Anande, "Motion Imitation Robotic Arm, (MIRA)," vol. 06, no. 01, hlm. 7, 2019.
[17] "HX Studio - 16-Channel 12-bit PWM/Servo Driver - I2C interface - PCA9685 for Arduino Raspberry Pi DIY Servo Shield Module - gratustoy." https://gratustoy.com/hx-studio-16-channel-12-bit-pwm-servo-driver-i2c-interface-pca9685-for-arduino-raspberry-pi-diy-servo-shield-module/ (diakses Jul 15, 2020).
[18] R. B. M. Indonesia, "RASPBERRY Pi3 Model B - PT. RAKSASA BISNIS INDONESIA." https://www.ralali.com/v/raksasa-bisnis/product/raspberry-pi3-model-b-164691001 (diakses Jul 15, 2020).
[19] E. küçükkülahlı dan R. GULER, "Open Source Mobile Robot with Raspberry Pi," *Balkan Journal of Electrical and Computer Engineering*, vol. 3, Des 2015, doi: 10.17694/bajece.29976.
[20] "Raspberry Pi Pinout," *IoT Bytes*, Apr 28, 2016. https://iotbytes.wordpress.com/raspberry-pi-pinout/ (diakses Jul 15, 2020).
[21] "Capabilities | VNC® Connect." https://www.realvnc.com/en/connect/capabilities/ (diakses Jul 15, 2020).
[22] "About — blender.org." https://www.blender.org/about/ (diakses Jul 15, 2020).
[23] "What is 3D Rigging For Animation & Character Design?," *Concept Art Empire*, Jul 26, 2018. https://conceptartempire.com/what-is-rigging/ (diakses Jun 14, 2020).
[24] O. Kainz, F. Jakab, dan S. Kardos, "The computer animation in education," Okt 2013, hlm. 201–206, doi: 10.1109/ICETA.2013.6674428.

LAMPIRAN

1. MainProgram.cpp

```
#define SIMULATION
#define WINDOWS
#include "MotionData.h"
#include "HexapodIK.h"
#include "ServoDriver.h"
#include "SensorControl.h"
#include "TerminalDisplay.h"
#ifdef WINDOWS
#include "WindowsInputController.h"
#else
#include "RaspiInputController.h"
#endif



#ifndef MAIN_PROG
#define MAIN_PROG
Controller controller;

#ifdef SIMULATION
#include "Vrep_connector.h"
Vrep_connector sim;
#endif
HexapodIK g_IK;
ServoDriver g_driver;
SensorControl g_sensor;
int g_stairState = 0;
int g_stairMode = 0;
bool stairClimbing = false;
int climbState = 0;
bool climbing = false;
//TerminalDisplay g_display;
float updateRate = 1;
float floorDistance = 25, outboundDistance = 40, stairFloor
= 35;
int checkCount = 0;


bool swReset = false, swPause = false;
bool swReset_last = false, swPause_last = false;
float timeScale = 1;
bool quit = false;

#ifdef SIMULATION
```

```
const string climb1DataPath = "F:\\Tugas
Akhir\\Simulation\\Blender_sim\\Output\\Climbing_1.dat";
const string climb2DataPath = "F:\\Tugas
Akhir\\Simulation\\Blender_sim\\Output\\Climbing_2.dat";
const string climb3DataPath = "F:\\Tugas
Akhir\\Simulation\\Blender_sim\\Output\\Climbing_3.dat";
#else

const string climb1DataPath =
"/home/pi/Desktop/MotionFiles/Climbing_1.dat";
const string climb2DataPath =
"/home/pi/Desktop/MotionFiles/Climbing_2.dat";
const string climb3DataPath =
"/home/pi/Desktop/MotionFiles/Climbing_3.dat";
#endif

bool Init() {
#ifdef SIMULATION
        cout << "Running in simulator Mode\n\n";
        if (!sim.Init())
                return false;

        sim.GetLegDistance();
        g_driver.Init(sim);
        g_sensor.Init(g_driver, sim);
#else
        cout << "Running in non-simulator Mode\n\n";
        g_driver.Init();
        controller.Init();
        g_sensor.Init(g_driver);
#endif
        g_IK.Init(g_driver);
        stairClimbing = false;


        Term::Init();
        return true;
}

IKKeyFrame keyframes[6];



void Generate_climbing_1() {
        climbing = true;
        MotionData m = MotionData(climb1DataPath);

        keyframes[0].AssignKeyFrame(m.frame[0]);
        keyframes[1].AssignKeyFrame(m.frame[1]);
        keyframes[2].AssignKeyFrame(m.frame[2]);
```

```
        keyframes[3].AssignKeyFrame(m.frame[3]);
        keyframes[4].AssignKeyFrame(m.frame[4]);
        keyframes[5].AssignKeyFrame(m.frame[5]);

        for (int i = 0; i < 6; i++)
        {
                keyframes[i].loopMode =
IKKeyFrame::loop_mode_none;
        }
}

void Generate_climbing_2() {
        Term::Print(0, Term::Row() - 4, "Procceding to
Climbing 2");
        climbing = true;
        MotionData m = MotionData(climb2DataPath);

        keyframes[0].AssignKeyFrame(m.frame[0]);
        keyframes[1].AssignKeyFrame(m.frame[1]);
        keyframes[2].AssignKeyFrame(m.frame[2]);

        keyframes[3].AssignKeyFrame(m.frame[3]);
        keyframes[4].AssignKeyFrame(m.frame[4]);
        keyframes[5].AssignKeyFrame(m.frame[5]);

        for (int i = 0; i < 6; i++)
        {
                keyframes[i].loopMode =
IKKeyFrame::loop_mode_repeat;
        }
}

void Generate_climbing_3() {
        climbing = true;
        MotionData m = MotionData(climb3DataPath);

        keyframes[0].AssignKeyFrame(m.frame[0]);
        keyframes[1].AssignKeyFrame(m.frame[1]);
        keyframes[2].AssignKeyFrame(m.frame[2]);

        keyframes[3].AssignKeyFrame(m.frame[3]);
        keyframes[4].AssignKeyFrame(m.frame[4]);
        keyframes[5].AssignKeyFrame(m.frame[5]);

        for (int i = 0; i < 6; i++)
        {
                keyframes[i].loopMode =
IKKeyFrame::loop_mode_none;
        }
```

```
}

void initiateStairClimbing() {
	Generate_climbing_1();
	g_IK.UseKeyFrames(keyframes, true);
	climbState = 0;
}

void initiateNormalMode() {
	if (!stairClimbing || swReset)
		g_IK.InitializeKeyFrames();
}




stringstream s;
int Update() {
	float castPoint = g_sensor.GetDistance();
	//*/Term::clrDisp();
	s = stringstream();
	s << "Jarak: " << castPoint << "\n";
	Term::Print(20, 2, s.str());
	s = stringstream();
	s << "CheckCount: " << checkCount << "\n";
	Term::Print(20, 3, s.str());
	if (!stairClimbing) {
		g_sensor.SetAngle(0, -75);
		if (castPoint >= floorDistance) {
			updateRate = 1;
			g_IK.Move(Vector4(0, 8, 5));
			g_IK.Update(updateRate * timeScale);
			checkCount = 0;
		}
		else if (castPoint < floorDistance)
		{
			checkCount++;
			if (checkCount > 15) {
				g_IK.Move(Vector4(0, 0, 10));
				g_IK.Update(updateRate *
timeScale);
				climbing = true;
				initiateStairClimbing();
				stairClimbing = true;
				g_sensor.SetAngle(0, -45);
				updateRate = 1;
				checkCount = 0;
			}
		}
```

```
        }
        else {
                if (climbing) {
                        if (climbState == 0 &&
g_IK.getTime_s() > keyframes[0].getDuration())
                        {
                                Generate_climbing_2();
                                g_IK.UseKeyFrames(keyframes,
true);
                                climbState = 1;
                                updateRate = 0.5;

                        }
                        if (climbState == 1 ) {
                                if (castPoint >
outboundDistance)
                                        checkCount++;
                                else
                                        checkCount = 0;
                                if (checkCount > 10000) {
                                        //stairClimbing =
false;
                                        Generate_climbing_3();

        g_IK.SuspendKeyFrames(keyframes, true, 1);
                                        climbState = 2;
                                        updateRate = 0.5;
                                        g_sensor.SetAngle(-5,
0);
                                }
                        }
                }
        }
        if (stairClimbing) {
                if (climbing)
                        s << "Mode: stair climbing";
        }
        else
                s << "Mode: Normal";
        s << "\nState: " << climbState;
        Term::Print(0, 0, s.str());
        g_IK.Update(updateRate * timeScale);
        return 0;
}

void Try() {
        Vector4 frame0[] = { Vector4(0,0,0,0),
Vector4(0,0,5,1), Vector4(0,5,5,2), Vector4(0,5,0,3),
Vector4(0,0,0,4), Vector4(0,0,0,5) };
```

```
        Vector4 frame1[] = { Vector4(0,0,0,0),
Vector4(0,0,0,5) };//, Vector4(0, 0, 0, 1), Vector4(0, 0, 0,
2), Vector4(0, 0, 5, 3), Vector4(0, 0, 0, 4), Vector4(0, 0,
0, 5), Vector4(0, 0, 0, 6), Vector4(0, 0, 0, 7), Vector4(0,
0, 0, 8)};
        Vector4 frame2[] = { Vector4(0,0,0,0),
Vector4(0,0,0,5) };// , Vector4(0, 0, 0, 1), Vector4(0, 0,
0, 2), Vector4(0, 0, 0, 3), Vector4(0, 0, 5, 4), Vector4(0,
0, 0, 5), Vector4(0, 0, 0, 6), Vector4(0, 0, 0, 7),
Vector4(0, 0, 0, 8) };

        Vector4 frame3[] = { Vector4(0,0,0,0),
Vector4(0,0,0,5) };// , Vector4(0, 0, 0, 1), Vector4(0, 0,
0, 2), Vector4(0, 0, 0, 3), Vector4(0, 0, 0, 4), Vector4(0,
0, 5, 5), Vector4(0, 0, 0, 6), Vector4(0, 0, 0, 7),
Vector4(0, 0, 0, 8)};
        Vector4 frame4[] = { Vector4(0,0,0,0),
Vector4(0,0,0,5) };//, Vector4(0,0,0,1), Vector4(0,0,0,2),
Vector4(0,0,0,3), Vector4(0,0,0,4), Vector4(0,0,0,5),
Vector4(0,0,5,6), Vector4(0,0,0,7), Vector4(0,0,0,8) };
        Vector4 frame5[] = { Vector4(0,0,0,0),
Vector4(0,0,0,5) };// , Vector4(0, 0, 0, 1), Vector4(0, 0,
0, 2), Vector4(0, 0, 0, 3), Vector4(0, 0, 0, 4), Vector4(0,
0, 0, 5), Vector4(0, 0, 0, 6), Vector4(0, 0, 5, 7),
Vector4(0, 0, 0, 8)};

        keyframes[0].AssignKeyFrame(frame0, sizeof(frame0) /
sizeof(Vector4));
        keyframes[1].AssignKeyFrame(frame1, sizeof(frame1) /
sizeof(Vector4));
        keyframes[2].AssignKeyFrame(frame2, sizeof(frame2) /
sizeof(Vector4));

        keyframes[3].AssignKeyFrame(frame3, sizeof(frame3) /
sizeof(Vector4));
        keyframes[4].AssignKeyFrame(frame4, sizeof(frame4) /
sizeof(Vector4));
        keyframes[5].AssignKeyFrame(frame5, sizeof(frame5) /
sizeof(Vector4));

        for (int i = 0; i < 6; i++)
        {
                keyframes[i].loopMode =
IKKeyFrame::loop_mode_repeat;
        }
}



int main() {
```

```
        Init();
        updateRate = 1;

        g_sensor.SetAngle(0, -90);
        g_sensor.GetDistance();
        g_sensor.GetDistance();
        g_sensor.GetDistance();



        g_IK.Update(0);
        Sleep(500);

        stairClimbing = false;
        initiateNormalMode();
        int count;
        while (!controller.GetKey(Controller::esc) && !quit)
{
                Update();


        }
        //myfile.close();
        Term::Print(50, Term::Row() - 4, "Turning off
program...");
        Timer::Delay_ms(1000);
        return 0;
}


#endif
```

2. HexapodIK.h

```
#undef SIMULATION
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <string.h>
#include <sstream>
#include "mathing/quaternion.h"
#include "Vector4.h"
#include <array>
#include <bitset>
#include <vector>
#include "Mathf.h"
```

```
#include "Timer.h"
#include "ServoDriver.h"
#include <fstream>
#include "TerminalDisplay.h"
#include "MotionData.h"
#include <algorithm>


#ifdef SIMULATION
#include "Vrep_connector.h"
#endif

#ifndef HEXAPOD_IK_H
#define HEXAPOD_IK_H

using namespace std;
using namespace std::chrono;
using namespace mathing;
using namespace Mathf;



// Array of keys (vector4) for time function points
class IKKeyFrame {

public:enum LoopMode
{
        loop_mode_repeat, loop_mode_pingPong, loop_mode_none
};
public:LoopMode loopMode = loop_mode_repeat;
public:std::vector<Vector4> keyFrame;
            float duration;
public:float getDuration() {
        return duration;
}
public:float timeShift = 0;
            int old_TimePassed;
            int timePassed;
            // Return time / duration passed after 'GetTime()'
or 'GetValue()' called
public:int getTimePass() {
        return timePassed;
}
            /* Return true if there is new (time / duration)
pass
            (Only can call once)*/
public:bool isNewPass() {
        return timePassed > old_TimePassed;
}
```

```
public:void donePassChecking() {
        old_TimePassed = timePassed;
}

            static bool sortKeys(const Vector4& i, const
Vector4& j) {
                    return (i.w < j.w);
            }

            //-----------------------------------------------
----------- Constructor & Initializer

public:IKKeyFrame() {
        loopMode = loop_mode_repeat;
        duration = 0;
        old_TimePassed = 0;
        timePassed = 0;
}

            // Change frames without change time data
public:void Change(IKKeyFrame newFrames) {
        loopMode = newFrames.loopMode;
        duration = newFrames.duration;
        keyFrame = newFrames.keyFrame;
}

public:IKKeyFrame(LoopMode KeyLoopMode) {
        duration = 0;
        old_TimePassed = 0;
        timePassed = 0;
        loopMode = KeyLoopMode;
}

public:IKKeyFrame(std::vector<Vector4> KeyFrames, LoopMode
KeyLoopMode) {
        duration = 0;
        old_TimePassed = 0;
        timePassed = 0;

        keyFrame = KeyFrames;
        std::sort(keyFrame.begin(), keyFrame.end(),
sortKeys);
        duration = keyFrame.back().w;
        loopMode = KeyLoopMode;
}

public:void UpdateDuration() {
        duration = keyFrame.back().w;
}
```

```
            // Push key to array. Use 'FinishPushing' to end
pushing

public:void Push(Vector4* key, int arraySize) {
        for (int i = 0; i < arraySize; i++)
        {
                keyFrame.push_back(key[i]);
        }
}

public:void Push(Vector4 key) {
        keyFrame.push_back(key);
}

public:void Push(Vector4 key, float time) {
        key.w = time;
        keyFrame.push_back(key);
}

public:void FinishPushing() {
        std::sort(keyFrame.begin(), keyFrame.end(),
sortKeys);
        duration = keyFrame.back().w;

}

            //-----------------------------------------------
----------- Modifier

public:void ModifyAt(Vector4 newValue, int index) {
        keyFrame.at(index) = newValue;
        std::sort(keyFrame.begin(), keyFrame.end(),
sortKeys);
        duration = keyFrame.back().w;
}

public:void InsertKeyFrame(Vector4 key) {
        keyFrame.push_back(key);
        std::sort(keyFrame.begin(), keyFrame.end(),
sortKeys);
        duration = keyFrame.back().w;
}

public:void AssignKeyFrame(std::vector<Vector4> keys) {
        keyFrame = std::vector<Vector4>(keys);
        std::sort(keyFrame.begin(), keyFrame.end(),
sortKeys);
        duration = keyFrame.back().w;
}
```

```
public:void AssignKeyFrame(Vector4* keys, int length) {
        keyFrame = vector<Vector4>();
        for (int i = 0; i < length; i++)
        {
                keyFrame.push_back(keys[i]);
        }
        std::sort(keyFrame.begin(), keyFrame.end(),
sortKeys);
        duration = keyFrame.back().w;
}

public:void AssignKeyFrame(IKKeyFrame newframe) {
        keyFrame = std::vector<Vector4>(newframe.keyFrame);
        std::sort(keyFrame.begin(), keyFrame.end(),
sortKeys);
        duration = keyFrame.back().w;

        loopMode = newframe.loopMode;
}

           //-------------------------------------------------
---------- Output Functions

           float GetTime(float time) {
                  timePassed = time / duration;
                  stringstream s;
                  s << timePassed;
                  Term::Print(20, 6, s.str());
                  if (time < 0)
                         time = 0;
                  switch (loopMode)
                  {
                  case IKKeyFrame::loop_mode_none:
                         time = min(time, duration);
                         break;
                  case IKKeyFrame::loop_mode_repeat:
                         time = fmodf(time, duration);
                         break;
                  case IKKeyFrame::loop_mode_pingPong:
                         if (timePassed % 2 == 0)
                                time = fmodf(time,
duration);
                         else
                                time = duration -
fmodf(time, duration);
                         break;
                  default:
                         break;
                  }
                  return time;
```

```
        }

public:float GetEvaluatedTime(float time) {
        time = GetTime(time + timeShift * duration);
        return time;
}

public:Vector4 GetValue(float time) {
        time = GetTime(time + timeShift * duration);


        int sizeOfArray = keyFrame.size();
        //cout << "size : " << sizeOfArray << "\n";
        float tmax = keyFrame[sizeOfArray - 1].w;
        int i = 0;
        float ts = 0;
        //cout << "\t\t\t\tt = " << t << "\n";
        for (i = 0; i < sizeOfArray; i++)
        {
                //cout << "\t\t\t\t" << i << " w : " <<
data[i].w << "\n";
                if (i == sizeOfArray - 1)
                        break;
                if (time >= keyFrame[i].w && time <=
keyFrame[i + 1].w)
                {
                        float j = keyFrame[i + 1].w -
keyFrame[i].w;
                        ts = time - keyFrame[i].w;
                        //cout << ts << "\n" << j << "\n";
                        ts /= j;
                        break;
                }
        }
        ///cout << "\t\t\t\t" << i << ", t : " << sizeOfArray
<< "\n";
        Vector4 x;
        if (i == sizeOfArray - 1)
                x = keyFrame[i];
        else
                x = keyFrame[i] + (keyFrame[i + 1] -
keyFrame[i]) * ts;
        return x;
};

           Vector4 GetValue(float time, float percentShift) {
                   Vector4 v = GetValue(time + percentShift *
duration);

                   return v;
```

```
        }

public:void Print(string name) {
        std::cout << "\nKeyFrame '" << name << "' :\n";
        for (size_t i = 0; i < keyFrame.size(); i++)
        {
                std::cout << " t: " << keyFrame.at(i).w << "
= " << keyFrame.at(i).x << ", " << keyFrame.at(i).y << ", "
<< keyFrame.at(i).z << "\n";
        }
}

};


// Parameter struck for walking algoritm
class IKStepParam {
public:Vector4 offset, delta, liftOffset;
public:float id, stepLift, floorHeight, timeShift,
liftPeakPosition;
public:IKKeyFrame::LoopMode loopMode =
IKKeyFrame::loop_mode_repeat;

        IKStepParam() {
                offset = Vector4();
                delta = Vector4();
                floorHeight = 0;
                stepLift = 0;
                loopMode = IKKeyFrame::loop_mode_repeat;
                timeShift = 0;
                liftPeakPosition = 0;
        };

        IKStepParam(Vector4 _offset, Vector4 _delta, float
_floorHeight, float _height, IKKeyFrame::LoopMode _loopMode,
float _timeShift) {
                offset = _offset;
                liftOffset = Vector4();
                delta = _delta;
                floorHeight = _floorHeight;
                stepLift = _height;
                loopMode = _loopMode;
                timeShift = _timeShift;
                liftPeakPosition = 0;
        }
```

```
            IKStepParam(Vector4 _offset, Vector4 _delta, float
_floorHeight, float _height, IKKeyFrame::LoopMode _loopMode,
float _timeShift, float _liftPosition) {
                    offset = _offset;
                    liftOffset = Vector4();
                    delta = _delta;
                    floorHeight = _floorHeight;
                    stepLift = _height;
                    loopMode = _loopMode;
                    timeShift = _timeShift;
                    liftPeakPosition = _liftPosition;
            }

            static IKStepParam CreateWalkParam(Vector4
stepVector, float floorHeight, float liftHeight, float
liftPosition) {
                    IKStepParam param = IKStepParam(Vector4(),
stepVector, floorHeight, liftHeight,
IKKeyFrame::loop_mode_repeat, 0, liftPosition);
                    return param;
            }


};




class HexapodIK {
        ServoDriver servo;
        MotionData motion = MotionData();


        const string walkDataPath = "F:\\Tugas
Akhir\\Simulation\\Blender_sim\\Output\\Walk.dat";

        float lengan1 = 6.37163;
        float lengan2 = 7.45592;
        float lengan3 = 11.8404;

        float joint2_a = 0;
        float joint3_a = 0;

        float minMagnitude = 13, maxMagnitude = 20;

        float angle1_Offset[6] = { -45.0, 45.0, 0.0, 0.0,
45.0, -45.0 };
        float angle2_Offset = 18;
```

```
        float angle3_Offset = 17;

        Vector4 legAxis[6] = { Vector4(1, 1, 1, 1), Vector4(-
1, -1, 1, 1), Vector4(1, 1, 1, 1), Vector4(-1, -1, 1, 1),
Vector4(1, 1, 1, 1), Vector4(-1, -1, 1, 1) };




        // ------------------------------------------------
----------------------------------------------------------
Param: IK States

public:Vector4 originIKPosition[6] = { Vector4(0,0,0),
Vector4(0,0,0), Vector4(0,0,0),
Vector4(0,0,0),Vector4(0,0,0) ,Vector4(0,0,0) };
           Vector4 currentIKPosition[6] = { Vector4(0,0,0),
Vector4(0,0,0), Vector4(0,0,0),
Vector4(0,0,0),Vector4(0,0,0) ,Vector4(0,0,0) };
           float servoAngles[18] = { 0 };

           IKKeyFrame currentIKKeys[6];
           IKStepParam currentIKParam[6];


public:void Init(ServoDriver _driver) {
        ResetTime();
        servo = _driver;
        Vector4 arm[4];

        cout << "\n\n";
        for (int i = 0; i < 6; i++)
        {
                char hexaJointPrefix[] = "hexa_joint";
                char legTipPrefix[] = "hexa_footTip";

                stringstream ss2;
                ss2 << legTipPrefix << (i + 1);
                string s2 = ss2.str();
                originIKPosition[i] = CalculateFK(i,
Vector4(angle1_Offset[i], 0, 0));


                currentIKPosition[i] = originIKPosition[i];

                originIKPosition[i].z *= -1;
                cout << "origin[" << i << "] = " <<
originIKPosition[i].print() << "\n";

        }
```

```
        InitializeKeyFrames();

}


public:HexapodIK() {}
           // Forward Kinematics

#pragma region Forward Kinematics


           // Perhitungan FK dasar
public:Vector4 CalculateFK(Vector4 angle) {
        Vector4 forward = Vector4();
        angle.y -= angle2_Offset;
        angle.z -= angle3_Offset;
        angle *= Deg2Rad;

        float theta = (90 * Deg2Rad) - angle.z;


        float bd = sqrt(lengan2 * lengan2 + lengan3 * lengan3
- 2 * lengan2 * lengan3 * (float)cos(theta));
        float bc = lengan2;
        float bc2 = lengan2 * lengan2;
        float bd2 = bd * bd;
        float cd2 = lengan3 * lengan3;
        float beta = acos((bc2 + bd2 - cd2) / (2 * bc * bd));
        float alpha = (beta - angle.y);
        float panjangH = lengan1 + bd * cos(alpha);
        float panjangV = bd * sin(alpha);
        //cout << (bc2 + bd2 - cd2) / (2 * bc * bd) << "\n";
        forward.x = cos(angle.x) * panjangH;
        forward.y = sin(angle.x) * -panjangH;
        forward.z = panjangV;
        return forward;
}


           // Menghitung FK berdasarkan indeks kaki
(legIndex)
public:Vector4 CalculateFK(int legIndex, Vector4 joint) {
        Vector4 forward = CalculateFK(joint);
        forward *= legAxis[legIndex];
        cout << "\n\nFwd[" << legIndex << "] = " <<
forward.print();
        return forward;
}
```

```
#pragma endregion

#pragma region Inverse Kinematics

           int ik_rec = 0;
           // Perhitungan IK dasar
public:Vector4 CalculateIK(Vector4 point) {
        point.z = -point.z;

        float sudut1 = atan2(-point.y, point.x);
        float xLengan = sqrt(point.x * point.x + point.y *
point.y);
        sudut1 *= Rad2Deg;
        float de = xLengan - lengan1;
        float alpha = atan2(de, point.z) * (Rad2Deg);
        float bd = sqrt(de * de + point.z * point.z);
        float bc = lengan2;
        float bc2 = bc * bc;
        float bd2 = bd * bd;
        float cd = lengan3;
        float cd2 = lengan3 * lengan3;
        float beta = acos((bc2 + bd2 - cd2) / (2 * bc * bd))
* (Rad2Deg);
        float sudut2 = (beta + alpha) - 90;
        float tetha = acos((bc2 + cd2 - bd2) / (2 * bc * cd))
* (Rad2Deg);
        float sudut3 = 90 - tetha;

        Vector4 output = Vector4(sudut1, sudut2 +
angle2_Offset, -(sudut3 + angle3_Offset));

        if (ik_rec == 0 && (isnan(output.x) ||
isnan(output.y) || isnan(output.z))) {
               point.z = -point.z;

        CalculateIK(point.clampMagnitude(minMagnitude,
maxMagnitude));

        }
        ik_rec = 0;
        return output;
}

           // Menghitung IK yang telah diselaraskan axisnya
dengan 'angle1_offset'
public:Vector4 CalculateIKAligned(Vector4 point, int
legIndex) {
        point *= legAxis[legIndex];
        float offset = angle1_Offset[legIndex] * (Deg2Rad);
```

```
        float _x = cos(offset) * point.x - sin(offset) *
point.y;
        float _y = sin(offset) * point.x + cos(offset) *
point.y;
        point = Vector4(_x, _y, point.z);

        Vector4 Output = CalculateIK(point);

        /*if (isnan(Output.x) || isnan(Output.y) ||
isnan(Output.z))
                cerr << "[error IK[" << legIndex << "]](t = "
<<timeIK_s << "): " << point.print() << "\twith mag " <<
point.getMagnitude() << "\n";*/
        return Output;
}

           // Menghitung IK yang telah ditambahkan dengan
offset aslinya
public:Vector4 CalculateIKRelative(Vector4 point, int
legIndex) {
        Vector4 originPoint = currentIKPosition[legIndex];
        float offset = angle1_Offset[legIndex] * (Deg2Rad);
        float _x = cos(offset) * originPoint.x - sin(offset)
* originPoint.y;
        float _y = sin(offset) * originPoint.x + cos(offset)
* originPoint.y;

        originPoint = Vector4(_x, _y, originPoint.z);
        originPoint = originPoint + point;
        return CalculateIK(originPoint);
}

           // Menghitung IK yang telah ditambahkan dengan
offset aslinya dan diselaraskan axisnya dengan
'angle1_offset'
public:Vector4 CalculateIKRelativeAndAligned(Vector4 point,
int legIndex) {
        point = point + currentIKPosition[legIndex];
        return CalculateIKAligned(point, legIndex);
}

#pragma endregion

#pragma region LegControl

           // Mengatur sudut servo robot
public:void SetServoAngle(int servoIndex, float angle)
{
        if (isnan(angle))
                return;
```

```
        servo.SetServoAngle(servoIndex, angle);
        servoAngles[servoIndex] = angle;
}

           // Mengatur sudut servo dalam satu kaki robot
public:void SetLegAngle(int legIndex, Vector4 angles) {
        uint8_t index = (legIndex) * 3;

        if (legIndex % 2 == 0) {
                angles.y = -angles.y;
                angles.z = -angles.z;
        }

        if (isnan(angles.x) || isnan(angles.y) ||
isnan(angles.z)) {
                std::stringstream s;
                s << "<!> Leg[" << legIndex << "] is nan ("
<< timeIK_s << " s)";
                Term::Print(0, 6 + legIndex, s.str());
        }
        //cout << "\n\n     Angle : " << angles.print() <<
"\n";
        SetServoAngle(index, angles.x);
        SetServoAngle(index + 1, angles.y);
        SetServoAngle(index + 2, angles.z);
}



public:void SetPosition(Vector4* tipPosition) {
        for (size_t i = 0; i < 6; i++)
        {
                SetLegAngle(i,
CalculateIKAligned(tipPosition[i], i));//
.clampMagnitude(minMagnitude, maxMagnitude), i));
        }
        //Timer::Delay_ms(10);
}
#pragma endregion

#pragma region  KeyFrames

           void GenerateWalkKeyFrames(IKKeyFrame* output) {

                   motion = MotionData(walkDataPath);
                   for (int i = 0; i < 6; i++)
                   {

output[i].AssignKeyFrame(motion.frame[i]);
```

```
                        output[i].loopMode =
IKKeyFrame::loop_mode_repeat;
                }
        }


        // Apply walking keyframes from step parameters
public:void InitializeKeyFrames() {
        Vector4 delta = Vector4(0, 0, 0);

        motion = MotionData(walkDataPath);
        for (int i = 0; i < 6; i++)
        {

        currentIKKeys[i].AssignKeyFrame(motion.frame[i]);
                currentIKKeys[i].loopMode =
IKKeyFrame::loop_mode_repeat;
        }

        timePass = currentIKKeys[0].getTimePass();
        lastDelta = delta;

        for (int i = 0; i < 6; i++)
        {
                currentIKKeys[i] =
IKKeyFrame(motion.frame[i], IKKeyFrame::loop_mode_repeat);

                for (int j = 0; j <
currentIKKeys[i].keyFrame.size(); j++)
                {
                        currentIKKeys[i].keyFrame.at(j) =
originIKPosition[i] + currentIKKeys[i].keyFrame.at(j) *
Vector4(delta.x, delta.y, delta.z, 1);
                }
                currentIKKeys[i].UpdateDuration();
        }



}

        Vector4 lastDelta;
        int timePass = -1;
public:void ApplyWalkKeyFrames(Vector4 delta) {
        delta.x /= 2;
        delta.y /= 2;
        if (delta.x == 0 && delta.y == 0) {
                delta.z = 0;
        }
```

```
		timePass = currentIKKeys[0].getTimePass();
		lastDelta = delta;

		for (int i = 0; i < 6; i++)
		{
			currentIKKeys[i] =
IKKeyFrame(motion.frame[i], IKKeyFrame::loop_mode_repeat);

			for (int j = 0; j <
currentIKKeys[i].keyFrame.size(); j++)
			{
				currentIKKeys[i].keyFrame.at(j) =
originIKPosition[i] + currentIKKeys[i].keyFrame.at(j) *
Vector4(delta.x, delta.y, delta.z, 1);
			}
			currentIKKeys[i].UpdateDuration();
		}
}



#pragma region Simple walker

public:void Move(Vector4 moveVector) {
		ApplyWalkKeyFrames(moveVector);
}


#pragma endregion

public:void UseKeyFrames(IKKeyFrame* frames, bool
setFirstAsOffset) {
		ResetTime();
		for (int i = 0; i < 6; i++)
		{
			currentIKKeys[i].AssignKeyFrame(frames[i]);
			if (setFirstAsOffset) {
				for (int j = 0; j <
currentIKKeys[i].keyFrame.size(); j++)
				{

		currentIKKeys[i].keyFrame.at(j) =
currentIKKeys[i].keyFrame.at(j) + originIKPosition[i];

		//currentIKKeys[i].keyFrame.at(j).w -= 0.5;
					//cout << "Keyframe gamla " <<
i << " : " << currentIKKeys[i].keyFrame.at(0).print() <<
"\n";
				}
```

```
        //currentIKKeys[i].keyFrame.erase(currentIKKeys[i].ke
yFrame.begin());
                }
                currentIKKeys[i].UpdateDuration();
                //currentIKKeys[i].Print(to_string(i));
        }
        /*
        cout << "\n\n\nKeyframe 0: \n";
        for (float t = 0; t <=
currentIKKeys[0].getDuration(); t += 0.05) {
                cout << currentIKKeys[0].GetValue(t).print()
<< "\n";
        }
        cout << "end";
        */
}

           int suspendedPhase = 0;
           IKKeyFrame suspendedKeyFrames[6];
           bool suspendedKeyFrames_firstIsOffset;

public:void SuspendKeyFrames(IKKeyFrame* _frames, bool
setFirstAsOffset, int suspendTime) {
        suspendedKeyFrames[0] = _frames[0];
        suspendedKeyFrames[1] = _frames[1];
        suspendedKeyFrames[2] = _frames[2];

        suspendedKeyFrames[3] = _frames[3];
        suspendedKeyFrames[4] = _frames[4];
        suspendedKeyFrames[5] = _frames[5];
        suspendedKeyFrames_firstIsOffset = setFirstAsOffset;
        suspendedPhase = suspendTime;
}

           void ExecuteSuspended() {
                   suspendedPhase = 0;
                   UseKeyFrames(suspendedKeyFrames,
suspendedKeyFrames_firstIsOffset);
           }

#pragma region Updater

private:float timeIK_s = 0, timeDelta_s = 0,
servo_update_period = 0.001;
private:uint64_t timeIK_us = 0, time_delta_us = 0,
lastTimeIK_us = 0;
                float timeIK_last = 0;

                // return timeIK in microseconds
```

```
public:uint64_t getTime_us() { return timeIK_us; }
           // return timeDelta in microseconds
public:uint64_t getTimeDelta_us() { return time_delta_us; }
           // return timeIK in seconds
public:float getTime_s() { return timeIK_s; }
           // return timeIK in seconds
public:float getTimeDelta_s() { return timeDelta_s; }


public:void ResetTime() {
        timeIK_us = 0;
        lastTimeIK_us = Timer::micros() +(uint64_t)2e+6;
        timeIK_s = 0;
        timeIK_last = 0;
}


           // Call on begin every update loop
           void UpdateTime(float timeScale) {
                   uint64_t t = Timer::micros();
                   time_delta_us = t - lastTimeIK_us;
                   if (lastTimeIK_us > t)
                   {
                           if (timeScale == 0) {
                                   ResetTime();
                           }
                           time_delta_us = 0;
                   }
                   else {
                           lastTimeIK_us = t;

                           timeIK_us += time_delta_us;
                           timeDelta_s = (float)(time_delta_us
/ 1e+6);
                           timeIK_s += timeDelta_s *
timeScale;
                   }
           }


           std::stringstream s;
           float lastKeyTime = 0;
public:void UpdateIK(float time) {

        if (time - timeIK_last < servo_update_period && time
> 0)
                return;
        timeIK_last = time;
        //cout << "t: " << time << " s\n";
```

```
        float keyTime =
currentIKKeys[0].GetEvaluatedTime(time);
        if (suspendedPhase > 0 && keyTime - lastKeyTime < 0)
        {
                ExecuteSuspended();
                suspendedPhase = 0;
        }
        lastKeyTime = keyTime;
        for (int i = 0; i < 6; i++)
        {
                currentIKPosition[i] =
currentIKKeys[i].GetValue(time);
                //cout << "timeShift[" << i << "] : " <<
currentIKKeys[i].GetEvaluatedTime(time) << "\n";
        }
        SetPosition(currentIKPosition);
}

           // ------------------------------------------------
------------------------------------------------------------
--      ↳ Function: Main Updater

           // Update ik realtime
public:void Update(float timeScale) {
        UpdateTime(timeScale);
        //#ifdef SIMULATION
        s = stringstream();
        s << "Time: " << to_string(timeIK_s);
        Term::Print(20, 4, s.str());
        s = stringstream();
        s << "Pass: " << currentIKKeys[5].getTimePass();
        Term::Print(20, 5, s.str());
        //#endif
        UpdateIK(timeIK_s);

        //Term::Print(0, 0, currentIKPosition[0].print());
}


#pragma endregion
};


#endif
```

2. ServoDriver.h

```
#pragma once
```

```
#ifdef SIMULATION
#include "Vrep_connector.h"
#endif
#include "PCA9685.h"
#include "TerminalDisplay.h"
#include <string.h>
#include <sstream>

#define MIN_PULSE_WIDTH       650
#define MAX_PULSE_WIDTH       2650
#define DEFAULT_PULSE_WIDTH   1650
#define FREQUENCY             50

class ServoDriver
{
        PCA9685 pca[2];
public:char servoIndex[18] = {
//      coxa    femur   tibia
        16+13,  16+15,  16+12,  //      0
        0,              2,              3,              //
        1
        16+5,   16+7,   16+4,   //      2
        9,              8,              11,             //
        3
        16+2,   16+0,   16+3,   //      4
        13,             15,             12              //
        5
};
public:unsigned char sensorServoIndex[2] = { 4,      6 };
public:float servoOffset[18] = {
        //      coxa    femur   tibia
        -5,             0,              0,      //      0
        -4,             10,             0,              //
        1
        3,              -10,            0,      //      2
        5,              0,              0,              //
        3
        3,              -15,            -10,    //      4
        -2,             18,             0               //
        5
};
public:float tofServoOffset[2] = {0};
#ifdef SIMULATION
        Vrep_connector sim;

public:ServoDriver(){};

public:void Init(Vrep_connector _sim) {
                sim = _sim;
        }
```

```
        // Angle is -90 to 90
public:void SetServoAngle(int index, float angle) {
                sim.SetServo(index, angle +
servoOffset[index]);
        }

public:void SetSensorAngle(float horizontal, float vertical)
{
        sim.SetSensorAngle(horizontal, vertical);
}




#else
float map(float value, float minIn, float maxIn, float
minOut, float maxOut)
{
    float r = (maxIn - minIn);
    //if (value)
    float x = (value - minIn) / r;

    r = maxOut - minOut;
    x = x * r + minOut;
    return x;
}

int pulseWidth(float angle)
{
  int pulse_wide, analog_value;
  pulse_wide   = map(angle, -90, 90, MIN_PULSE_WIDTH,
MAX_PULSE_WIDTH);
  analog_value = int(float(pulse_wide) / 1000000 * FREQUENCY
* 4096);
  //Serial.println(analog_value);
  return analog_value;
}

public:void SetSensorAngle(float horizontal, float vertical)
{
        vertical = -vertical;
        int index = sensorServoIndex[0];
        int pcaIndex = index / 16;
        if (pcaIndex == 1)
                index -= 15;
        pca[pcaIndex].set_pwm(index, 0,
pulseWidth(horizontal));

        index = sensorServoIndex[1];
```

```
        pcaIndex = index / 16;
        if (pcaIndex == 1)
                index -= 15;
        pca[pcaIndex].set_pwm(index, 0,
pulseWidth(vertical));

}

        // Angle is -90 to 90
public:void SetServoAngle(int index, float angle) {
                //std::stringstream s;
                angle = angle + servoOffset[index];
                //s << "servo[" << index << " - ";
                int i = index;
                index = servoIndex[index];
                //s << index << "] = " << angle;

                int pcaIndex = 0;
                if (index > 15){
                        pcaIndex = 1;
                        index -= 16;
                }
                //s << " (" << pcaIndex << ") ";
                //Term::Print(20, 8 + i, s.str());
                pca[pcaIndex].set_pwm(index, 0,
pulseWidth(angle));
        }

#endif


public:void Init() {
                pca[0] = PCA9685(defaultDev, 0x40);
                pca[0].set_pwm_freq(50.0);
                pca[1] = PCA9685(defaultDev, 0x41);
                pca[1].set_pwm_freq(50.0);
        }

public:void Init(int address1, int address2) {
                pca[0] = PCA9685(defaultDev, address1);
                pca[1] = PCA9685(defaultDev, address2);
        }


};
```

3. SensorControl.h

```
//#pragma once


#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sstream>
#include "mathing/quaternion.h"
#include "Vector4.h"
#include <math.h>
#include "Mathf.h"
#include "ServoDriver.h"

//#define SIMULATION


#ifdef SIMULATION
#include "Vrep_connector.h"
extern "C" {
#include "extApi.h"
}


#else

//#include "RaspiController.h"
#include "VL53L0X.hpp"
#include <chrono>
#include <csignal>
#include <exception>
#include <iomanip>
#include <iostream>
#include <unistd.h>
#include <cmath>


#endif

using namespace std;
using namespace mathing;
using namespace Mathf;


#ifndef SIMULATION
class VL53L0X_driver {

        VL53L0X sensor;
```

```
public:enum MeasurementMode {
        mode_high_speed,
        mode_high_accuracy,
        mode_long_range
};
public:MeasurementMode measurementMode;

        int Init() {
                // Create the sensor with default values
                try {
                        // Initialize the sensor
                        sensor.initialize();
                        // Set measurement timeout value
                        sensor.setTimeout(200);
                }
                catch (const std::exception& error) {
                        std::cerr << "Error initializing
sensor with reason:" << std::endl << error.what() <<
std::endl;
                        return 1;
                }

                SetMeasurementMode(measurementMode);
        }

public:int SetMeasurementMode(MeasurementMode mode) {
        switch (mode)
        {
        case mode_long_range:
                try {
                        // Lower the return signal rate limit
(default is 0.25 MCPS)
                        sensor.setSignalRateLimit(0.1);
                        // Increase laser pulse periods
(defaults are 14 and 10 PCLKs)

        sensor.setVcselPulsePeriod(VcselPeriodPreRange, 18);

        sensor.setVcselPulsePeriod(VcselPeriodFinalRange,
14);
                }
                catch (const std::exception& error) {
                        std::cerr << "Error enabling long
range mode with reason:" << std::endl << error.what() <<
std::endl;
                        return 2;
                }
                break;
        case mode_high_speed:
                try {
```

```
                        // Reduce timing budget to 20 ms
(default is about 33 ms)

        sensor.setMeasurementTimingBudget(20000);
                }
                catch (const std::exception& error) {
                        std::cerr << "Error enabling high
speed mode with reason:" << std::endl << error.what() <<
std::endl;
                        return 3;
                }
                break;
        case mode_high_accuracy:
                try {
                        // Increase timing budget to 200 ms

        sensor.setMeasurementTimingBudget(200000);
                }
                catch (const std::exception& error) {
                        std::cerr << "Error enabling high
accuracy mode with reason:" << std::endl << error.what() <<
std::endl;
                        return 3;
                }
                break;
        default:
                break;
        }
}

           // Reading in milimeters
        uint16_t Read() {
                uint16_t distance;
                try {
                        // Read the range. Note that it's a
blocking call
                        distance =
sensor.readRangeSingleMillimeters();
                }
                catch (const std::exception& error) {
                        std::cerr << "Error geating
measurement with reason:" << std::endl << error.what() <<
std::endl;
                        // You may want to bail out here,
depending on your application - error means issues on I2C
bus read/write.
                        // return 3;
                        distance = 8096;
                }
```

```
                // Check IO timeout and print range
information
                if (sensor.timeoutOccurred()) {
                        //std::cout << "\rReading" << i << " |
timeout!" << std::endl;
                        return 8096;
                }
                else {
                        //std::cout << "\rReading" << i << " |
" << distance << std::endl;
                        return distance;
                }
        }

        float Read_cm() {
                float distance;
                try {
                        // Read the range. Note that it's a
blocking call
                        distance =
(float)sensor.readRangeSingleMillimeters() / 10;
                }
                catch (const std::exception& error) {
                        std::cerr << "Error geating
measurement with reason:" << std::endl << error.what() <<
std::endl;
                        // You may want to bail out here,
depending on your application - error means issues on I2C
bus read/write.
                        // return 3;
                        distance = 3.40282e+038;
                }

                // Check IO timeout and print range
information
                if (sensor.timeoutOccurred()) {
                        //std::cout << "\rReading" << i << " |
timeout!" << std::endl;
                        return 3.40282e+038;
                }
                else {
                        //std::cout << "\rReading" << i << " |
" << distance << std::endl;
                        return distance;
                }
        }
};

#endif
```

```
class SensorControl
{
        ServoDriver servo;

public:SensorControl(){}

#ifdef SIMULATION
        Vrep_connector sim;
#else
        VL53L0X_driver tofDriver;
#endif
        //simxUChar tofState = 0;
        float tofPoint[3] = { 0 };
        int tofTargetHandle = 0;
        float tofNormal[3] = { 0 };
        float tofAngle[2] = { 0 };
        float tofLengan1_x = (-1.2000e-02 + 1.2000e-02),
tofLengan1_y = -1.4000e-02, tofLengan1_z = +2.6000e-02,
tofLengan2 = -1.3311e-02;
        Vector4 tofAxis, tofHeading, tofOffset;

#ifdef SIMULATION
public:void Init(ServoDriver _driver, Vrep_connector _sim) {
        servo = _driver;
        sim = _sim;
        tofDriver.Init();
}
#endif

#ifndef SIMULATION
public:void Init(ServoDriver _driver) {
        servo = _driver;
        tofDriver.Init();
        tofDriver.SetMeasurementMode(VL53L0X_driver::mode_hig
h_speed);
}
#endif
           // Get distance of distance sensor
public:float GetDistance() {
#ifdef SIMULATION
        return sim.GetProximitySensorDistance();
#else
        return tofDriver.Read_cm();
#endif // SIMULATION

        //return Vector4(tofPoint).getMagnitude();
}

           // Get 3D point of sensor output
public:Vector4 GetDistancePoint() {
```

```
        float depth = GetDistance();
        Vector4 point = tofHeading;
        point.x *= depth;
        point.y *= depth;
        point.z *= depth;
        point.x -= tofOffset.x;
        point.y -= tofOffset.y;
        point.z -= tofOffset.z;
        point.w = depth;
        return point;
}


private:void SetServoAngle(float angle_pan, float
angle_tilt){
        if (isnan(angle_pan) || isnan(angle_tilt))
                return;
        tofAngle[0] = angle_pan;
        tofAngle[1] = angle_tilt;
        servo.SetSensorAngle(tofAngle[0], tofAngle[1]);
}



public:void SetAngle(float pan, float tilt)
{
        if (isnan(pan) || isnan(tilt))
                return;
        pan = Mathf::constrain(pan, -90, 90);
        tilt = Mathf::constrain(tilt, -90, 90);
        SetServoAngle(pan, tilt);
        pan *= Deg2Rad;
        tilt *= Deg2Rad;
        tofHeading.x = sin(-pan) * cos(tilt);
        tofHeading.y = cos(-pan) * cos(tilt);
        tofHeading.z = sin(tilt);
        tofOffset.x = sin(-pan) * tofLengan1_y;
        tofOffset.y = cos(-pan) * tofLengan1_y;
        tofOffset.z = tofLengan1_z;
        tofOffset.z += cos(tilt) * tofLengan2;
        tofOffset.y += sin(tilt) * tofLengan2;
}
};
```

4. TerminalDisplay.h

```
#include <ncurses.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <string.h>
#include <sstream>

#ifndef TERM_DISP
#define TERM_DISP

namespace Term
{
        int Row();

         int kbhit();

         void Println(std::string s);

         void Println(char* s);


         void Print(int x, int y, std::string s);

         void Print(int x, int y, char* s);

         void ShowCursor();

         void HideCursor();

         void clrDisp();

         void Init();


};

#endif
```

5. MotionData.h

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include "Vector4.h"

#ifndef MOTIONDATA_H
#define MOTIONDATA_H
```

```
class MotionData
{
public:vector<Vector4> frame[6];
public:MotionData(){}

public:MotionData(string filePath, Vector4 * initialPos) {
        string line;
        ifstream dataFile(filePath);
        int i = 0;
        if (dataFile.is_open()) {
                while (getline(dataFile, line)) {
                        if (line.find("dataset::") !=
string::npos) {
                                //cout << "frame[" << i << "]:
\n";

                                line =
line.substr(string("dataset::").length(), string::npos);
                                //cout << line << "\n\n";
                                int index = 0;
                                unsigned int lastIndex =
line.find_first_of(';');


        frame[i].push_back(initialPos[i].withW(0));
                                while (true) {
                                        Vector4 v = Vector4();

                                        string val =
line.substr(index, lastIndex - index);
                                        //cout << val << "\n";
                                        int start = 0;
                                        int stop =
val.find(',', 0);
                                        //cout << "( " <<
val.substr(start, stop - start);
                                        v.x =
stof(val.substr(start, stop - start));

                                        start = stop + 1;
                                        stop = val.find(',',
start);
                                        //cout << "| " <<
val.substr(start, stop - start);
                                        v.y =
stof(val.substr(start, stop - start));

                                        start = stop + 1;
                                        stop = val.find(',',
start);
```

```
                                        //cout << "| " <<
val.substr(start, stop - start);
                                        v.z =
stof(val.substr(start, stop - start));

                                        start = stop + 1;
                                        //cout << "| " <<
val.substr(start, string::npos) << ")\n";
                                        v.w =
stof(val.substr(start, string::npos));

                                        //cout << "\t" <<
(v.w*2) << " - " << v.print() << "\n";

                                        frame[i].push_back(v);
                                        index = lastIndex + 1;

                                        lastIndex =
line.find(';', index);
                                        //cout << "index: " <<
(lastIndex) << "\n";
                                        if (lastIndex ==
string::npos) {
                                                break;
                                        }
                                }

                                i++;
                        }
                }
                dataFile.close();
        }
        else {
                cout << "Unable to open file '" << filePath
<< "'";
        }
}

public:MotionData(string filePath) {
        string line;
        ifstream dataFile(filePath);
        int i = 0;
        if (dataFile.is_open()) {
                while (getline(dataFile, line)) {
                        if (line.find("dataset::") !=
string::npos) {
                                //cout << "frame[" << i << "]:
\n";
```

```
                                line =
line.substr(string("dataset::").length(), string::npos);
                                //cout << line << "\n\n";
                                int index = 0;
                                unsigned int lastIndex =
line.find_first_of(';');


                                while (true) {
                                        Vector4 v = Vector4();

                                        string val =
line.substr(index, lastIndex - index);
                                        //cout << val << "\n";
                                        int start = 0;
                                        int stop =
val.find(',', 0);
                                        //cout << "( " <<
val.substr(start, stop - start);
                                        v.x =
stof(val.substr(start, stop - start));

                                        start = stop + 1;
                                        stop = val.find(',',
start);
                                        //cout << "| " <<
val.substr(start, stop - start);
                                        v.y =
stof(val.substr(start, stop - start));

                                        start = stop + 1;
                                        stop = val.find(',',
start);
                                        //cout << "| " <<
val.substr(start, stop - start);
                                        v.z =
stof(val.substr(start, stop - start));

                                        start = stop + 1;
                                        //cout << "| " <<
val.substr(start, string::npos) << ")\n";
                                        v.w =
stof(val.substr(start, string::npos));

                                        //cout << "\t" <<
(v.w*2) << " - " << v.print() << "\n";

                                        frame[i].push_back(v);
                                        index = lastIndex + 1;
```

```
                                        lastIndex =
line.find(';', index);
                                        //cout << "index: " <<
(lastIndex) << "\n";
                                        if (lastIndex ==
string::npos) {
                                                break;
                                        }
                                }

                                i++;
                        }
                }
                dataFile.close();
        }
        else {
                cout << "Unable to open file '" << filePath
<< "'";
        }
}

};

#endif
```

6. KeyToAnimData.py (Blender)

```
import bpy

sce = bpy.context.scene
ob = bpy.context.object

def apply_animations(len):
    i = 0
    for i in range(0,6):
        pbone = ob.pose.bones["IK." + str(i)]
        for f in range(sce.frame_start, len+1, 15):
            sce.frame_set(f)
            pbone.keyframe_insert(data_path="location",
frame = f, options = {'INSERTKEY_VISUAL',
'INSERTKEY_REPLACE'})

pbone.keyframe_insert(data_path="rotation_quaternion", frame
= f, options = {'INSERTKEY_VISUAL', 'INSERTKEY_REPLACE'})



def write_animations(len,xyz_mul, xyz_add, time_offset,
filePath,arr):
    file = open(filePath,'w')
```

```
    apply_animations(len)
    for i in arr:
        pbone = ob.pose.bones["IK." + str(i)]
        s = "dataset::"
        file.write(s);
        for f in range(sce.frame_start, len+1, 15):
            sce.frame_set(f)
            t = str(f/30 + time_offset)
            s =
str(round(pbone.location.x*100*xyz_mul[0]+xyz_add[0], 2)) +
"," + str(round(-pbone.location.y*100*xyz_mul[1]+xyz_add[1],
2)) + "," +
str(round(pbone.location.z*100*xyz_mul[2]+xyz_add[2], 2)) +
"," + t + ";"
            file.write(s);
        suffix = "\n"
        file.write(suffix)
    print_output = "successfully saved '"+
ob.animation_data.action.name +"' animdata to '" + filePath
+ "'"
    print(print_output)
    file.close();



curAction = ob.animation_data.action
curKeyFrame = sce.frame_current;

ob.animation_data.action = bpy.data.actions['Climbing.1']
write_animations(990,[1,1,1],[0,0,0],0,'F:\Tugas
Akhir\Simulation\Blender_sim\Output\Climbing_1.dat', [0, 1,
2, 3, 4, 5])

ob.animation_data.action = bpy.data.actions['Climbing.2_4']
write_animations(315,[1,1,1],[0,0,0],0,'F:\Tugas
Akhir\Simulation\Blender_sim\Output\Climbing_2.dat', [0, 1,
2, 3, 4, 5])

ob.animation_data.action = bpy.data.actions['Climbing.3_3']
write_animations(495,[1,1,1],[0,0,0],0,'F:\Tugas
Akhir\Simulation\Blender_sim\Output\Climbing_3.dat', [0, 1,
2, 3, 4, 5])

ob.animation_data.action = curAction
sce.frame_set(curKeyFrame)
```

VL53L0X

World's smallest Time-of-Flight ranging and gesture detection sensor

Datasheet - production data

Features

- Fully integrated miniature module
  - 940 nm laser VCSEL
  - VCSEL driver
  - Ranging sensor with advanced embedded micro controller
  - 4.4 x 2.4 x 1.0 mm
- Fast, accurate distance ranging
  - Measures absolute range up to 2 m
  - Reported range is independent of the target reflectance
  - Advanced embedded optical cross-talk compensation to simplify cover glass selection
- Eye safe
  - Class 1 laser device compliant with latest standard IEC 60825-1:2014 - 3rd edition
- Easy integration
  - Single reflowable component
  - No additional optics
  - Single power supply
  - I2C interface for device control and data transfer
  - Xshutdown (reset) and interrupt GPIO
  - Programmable I2C address

Applications

- User detection for personal computers/ laptops/tablets and IoT (energy saving)
- Robotics (obstacle detection)
- White goods (hand detection in automatic faucets, soap dispensers etc.)
- 1D gesture recognition.
- Laser assisted autofocus. Enhances and speeds up camera autofocus system performance, especially in difficult scenes (low light levels, low contrast) or fast moving video mode.

Description

The VL53L0X is a new generation Time-of-Flight (ToF) laser-ranging module housed in the smallest package on the market today, providing accurate distance measurement whatever the target reflectances unlike conventional technologies. It can measure absolute distances up to 2m, setting a new benchmark in ranging performance levels, opening the door to various new applications.

The VL53L0X integrates a leading-edge SPAD array (Single Photon Avalanche Diodes) and embeds ST's second generation FlightSense™ patented technology.

The VL53L0X's 940 nm VCSEL emitter (Vertical Cavity Surface-Emitting Laser), is totally invisible to the human eye, coupled with internal physical infrared filters, it enables longer ranging distances, higher immunity to ambient light, and better robustness to cover glass optical crosstalk.

1 Overview

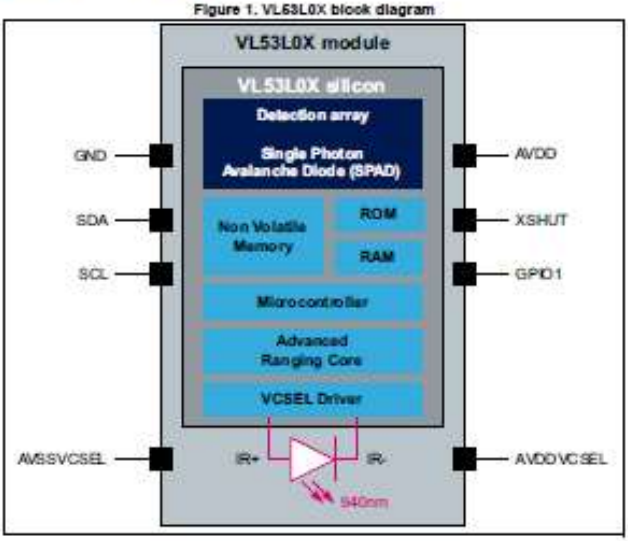
1.1 Technical specification

Table 1. Technical specification

Feature	Detail
Package	Optical LGA12
Size	4.40 x 2.40 x 1.00 mm
Operating voltage	2.6 to 3.5 V
Operating temperature	-20 to 70°C
Infrared emitter	940 nm
I²C	Up to 400 kHz (FAST mode) serial bus Address: 0x52

1.2 System block diagram

Figure 1. VL53L0X block diagram



1.3 Device pinout

Figure 2 shows the pinout of the VL53L0X (see also *Figure 22*).

Figure 2. VL53L0X pinout (bottom view)

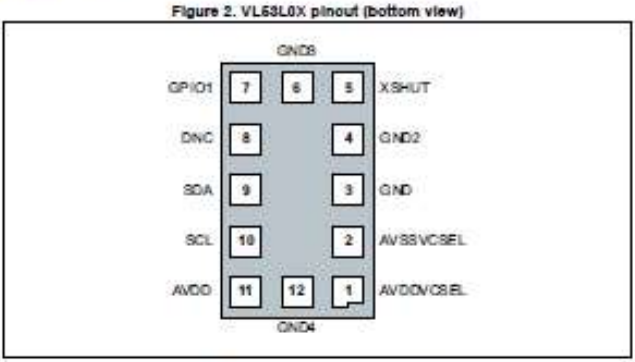
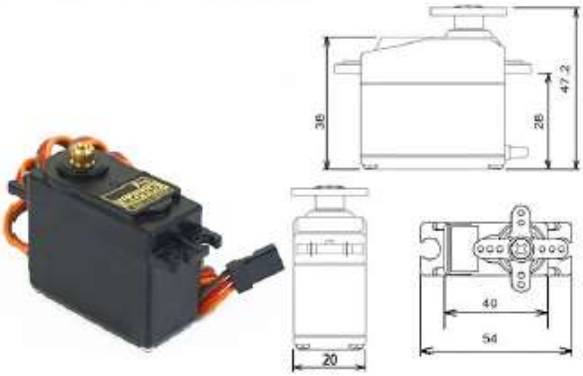


Table 2. VL53L0X pin description

Pin number	Signal name	Signal type	Signal description
1	AVDDVCSEL	Supply	VCSEL Supply, to be connected to main supply
2	AVSSVCSEL	Ground	VCSEL Ground, to be connected to main ground
3	GND	Ground	To be connected to main ground
4	GND2	Ground	To be connected to main ground
5	XSHUT	Digital input	Xahutdown pin, Active LOW
6	GND3	Ground	To be connected to main ground
7	GPIO1	Digital output	Interrupt output. Open drain output
8	DNC	Digital input	Do Not Connect, must be left floating
9	SDA	Digital input/output	I²C serial data
10	SCL	Digital input	I²C serial clock input
11	AVDD	Supply	Supply, to be connected to main supply
12	GND4	Ground	To be connected to main ground

MG995 High Speed
Metal Gear Dual Ball Bearing Servo

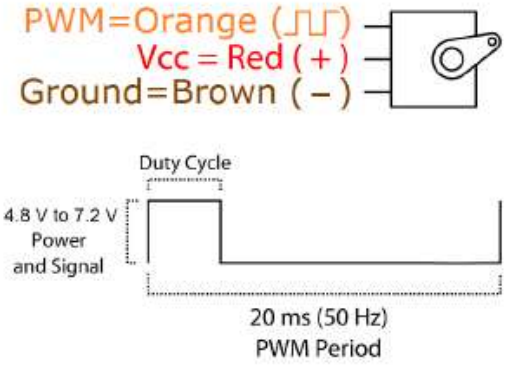


The unit comes complete with 30cm wire and 3 pin 'S' type female header connector that fits most receivers, including Futaba, JR, GWS, Cirrus, Blue Bird, Blue Arrow, Corona, Berg, Spektrum and Hitec.

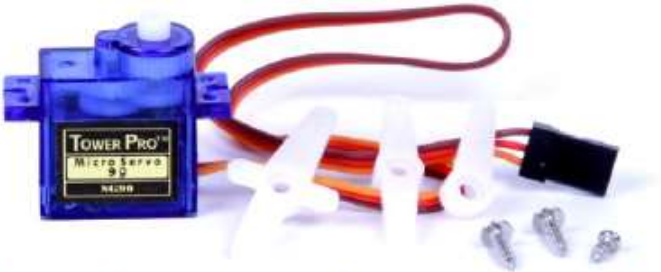
This high-speed standard servo can rotate approximately 120 degrees (60 in each direction). You can use any servo code, hardware or library to control these servos, so it's great for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. The MG995 Metal Gear Servo also comes with a selection of arms and hardware to get you set up nice and fast!

Specifications

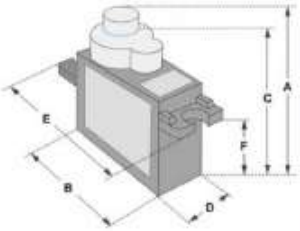
- Weight: 55 g
- Dimension: 40.7 x 19.7 x 42.9 mm approx.
- Stall torque: 8.5 kgf·cm (4.8 V), 10 kgf·cm (6 V)
- Operating speed: 0.2 s/60º (4.8 V), 0.16 s/60º (6 V)
- Operating voltage: 4.8 V a 7.2 V
- Dead band width: 5 µs
- Stable and shock proof double ball bearing design
- Temperature range: 0 ºC – 55 ºC


PWM=Orange
Vcc = Red (+)
Ground=Brown (–)
Duty Cycle
4.8 V to 7.2 V
Power
and Signal
20 ms (50 Hz)
PWM Period

SERVO MOTOR SG90 DATA SHEET

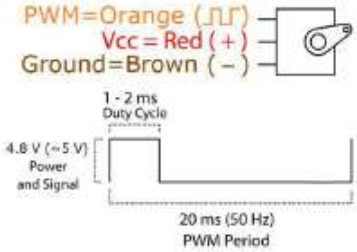


Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but smaller. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.



Dimensions & Specifications
A (mm) : 32
B (mm) : 23
C (mm) : 28.5
D (mm) : 12
E (mm) : 32
F (mm) : 19.5
Speed (sec) : 0.1
Torque (kg-cm) : 2.5
Weight (g) : 14.7
Voltage : 4.8 - 6

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle, is all the way to the right, "-90" (~1ms pulse) is all the way to the left.



PCA9685

16-channel, 12-bit PWM Fm+ I²C-bus LED controller

Rev. 02 — 16 July 2009 Product data sheet

1. General description

The PCA9685 is an I²C-bus controlled 16-channel LED controller optimized for LCD Red/Green/Blue/Amber (RGBA) color backlighting applications. Each LED output has its own 12-bit resolution (4096 steps) fixed frequency individual PWM controller that operates at a programmable frequency from a typical of 40 Hz to 1000 Hz with a duty cycle that is adjustable from 0 % to 100 % to allow the LED to be set to a specific brightness value. All outputs are set to the same PWM frequency.

Each LED output can be off or on (no PWM control), or set at its individual PWM controller value. The LED output driver is programmed to be either open-drain with a 25 mA current sink capability at 5 V or totem pole with a 25 mA sink, 10 mA source capability at 5 V. The PCA9685 operates with a supply voltage range of 2.3 V to 5.5 V and the inputs and outputs are 5.5 V tolerant. LEDs can be directly connected to the LED output (up to 25 mA, 5.5 V) or controlled with external drivers and a minimum amount of discrete components for larger current or higher voltage LEDs.

The PCA9685 is in the new Fast-mode Plus (Fm+) family. Fm+ devices offer higher frequency (up to 1 MHz) and more densely populated bus operation (up to 4000 pF).

Although the PCA9635 and PCA9685 have many similar features, the PCA9685 has some unique features that make it more suitable for applications such as LCD backlighting and Ambilight:

- The PCA9685 allows staggered LED output on and off times to minimize current surges. The on and off time delay is independently programmable for each of the 16 channels. This feature is not available in PCA9635.
- The PCA9685 has 4096 steps (12-bit PWM) of individual LED brightness control. The PCA9635 has only 256 steps (8-bit PWM).
- When multiple LED controllers are incorporated in a system, the PWM pulse widths between multiple devices may differ if PCA9635s are used. The PCA9685 has a programmable prescaler to adjust the PWM pulse widths of multiple devices.
- The PCA9685 has an external clock input pin that will accept user-supplied clock (50 MHz max.) in place of the internal 25 MHz oscillator. This feature allows synchronization of multiple devices. The PCA9635 does not have external clock input feature.
- Like the PCA9635, PCA9685 also has a built-in oscillator for the PWM control. However, the frequency used for PWM control in the PCA9685 is adjustable from about 40 Hz to 1000 Hz as compared to the typical 97.6 kHz frequency of the PCA9635. This allows the use of PCA9685 with external power supply controllers. All bits are set at the same frequency.
- The Power-On Reset (POR) default state of LEDn output pins is LOW in the case of PCA9685. It is HIGH for PCA9635.

The active LOW Output Enable input pin ($\overline{OE}$) allows asynchronous control of the LED outputs and can be used to set all the outputs to a defined I²C-bus programmable logic state. The $\overline{OE}$ can also be used to externally 'pulse width modulate' the outputs, which is useful when multiple devices need to be dimmed or blinked together using software control.

Software programmable LED All Call and three Sub Call I²C-bus addresses allow all or defined groups of PCA9685 devices to respond to a common I²C-bus address, allowing for example, all red LEDs to be turned on or off at the same time or marquee chasing effect, thus minimizing I²C-bus commands. Six hardware address pins allow up to 62 devices on the same bus.

The Software Reset (SWRST) General Call allows the master to perform a reset of the PCA9685 through the I²C-bus, identical to the Power-On Reset (POR) that initializes the registers to their default state causing the outputs to be set LOW. This allows an easy and quick way to reconfigure all device registers to the same condition via software.

2. Features

- 16 LED drivers. Each output programmable at:
  - Off
  - On
  - Programmable LED brightness
  - Programmable LED turn-on time to help reduce EMI
- 1 MHz Fast-mode Plus compatible I²C-bus interface with 30 mA high drive capability on SDA output for driving high capacitive buses
- 4096-step (12-bit) linear programmable brightness per LED output varying from fully off (default) to maximum brightness
- LED output frequency (all LEDs) typically varies from 40 Hz to 1000 Hz (Default of 1Eh in PRE_SCALE register results in a 200 Hz refresh rate with oscillator clock of 25 MHz.)
- Sixteen totem pole outputs (sink 25 mA and source 10 mA at 5 V) with software programmable open-drain LED outputs selection (default at totem pole). No input function.
- Output state change programmable on the Acknowledge or the STOP Command to update outputs byte-by-byte or all at the same time (default to 'Change on STOP').
- Active LOW Output Enable ($\overline{OE}$) input pin. LEDn outputs programmable to logic 1, logic 0 (default at power-up) or 'high-impedance' when $\overline{OE}$ is HIGH.
- 6 hardware address pins allow 62 PCA9685 devices to be connected to the same I²C-bus
- Toggling $\overline{OE}$ allows for hardware LED blinking
- 4 software programmable I²C-bus addresses (one LED All Call address and three LED Sub Call addresses) allow groups of devices to be addressed at the same time in any combination (for example, one register used for 'All Call' so that all the PCA9685s on the I²C-bus can be addressed at the same time and the second register used for three different addresses so that 1/3 of all devices on the bus can be addressed at the same time in a group). Software enable and disable for these I²C-bus address.
- Software Reset feature (SWRST General Call) allows the device to be reset through the I²C-bus

- 25 MHz typical internal oscillator requires no external components
- External 50 MHz (max.) clock input
- Internal power-on reset
- Noise filter on SDA/SCL inputs
- Edge rate control on outputs
- No output glitches on power-up
- Supports hot insertion
- Low standby current
- Operating power supply voltage range of 2.3 V to 5.5 V
- 5.5 V tolerant inputs
- −40 °C to +85 °C operation
- ESD protection exceeds 2000 V HBM per JESD22-A114, 200 V MM per JESD22-A115 and 1000 V CDM per JESD22-C101
- Latch-up testing is done to JEDEC Standard JESD78 which exceeds 100 mA
- Packages offered: TSSOP28, HVQFN28

3. Applications

- RGB or RGBA LED drivers
- LED status information
- LED displays
- LCD backlights
- Keypad backlights for cellular phones or handheld devices

4. Ordering information

Table 1. Ordering information

Type number	Topside mark	Package		
		Name	Description	Version
PCA9685PW	PCA9685PW	TSSOP28	plastic thin shrink small outline package; 28 leads; body width 4.4 mm	SOT361-1
PCA9685PWQ900[1]	PCA9685PW	TSSOP28	plastic thin shrink small outline package; 28 leads; body width 4.4 mm	SOT361-1
PCA9685BS	PCA9685BS	HVQFN28	plastic thermal enhanced very thin quad flat package; no leads; 28 terminals; body 6 × 6 × 0.85 mm	SOT788-1

[1] PCA9685PWQ900 is AEC-Q100 compliant. Contact i2c.support@nxp.com for PPAP.

BIODATA

Saya Mochamad Yusuf lahir di Kutai Kartanegara pada tanggal 30 Juni 1998. Saya memiliki ketertarikan untuk mempelajari dan mendalami beberapa bahasa pemrograman. Mulai dari tahun 2016, saya belajar di Departemen Teknik Elektro, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember.