Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - EE184801

PERANCANGAN SISTEM NAVIGASI LEADER-FOLLOWER BERBASISKAN PEDESTRIAN DEAD-RECKONING (PDR) UNTUK NAVIGASI WHEELED MOBILE ROBOT

Muhammad Farih
NRP 07111540000104

Dosen Pembimbing
Mochammad Sahal, S.T., M.Sc.
Ir. Rusdhianto Effendi A. K., MT.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2019

TUGAS AKHIR - EE184801

PERANCANGAN SISTEM NAVIGASI LEADER-FOLLOWER BERBASISKAN PEDESTRIAN DEAD-RECKONING (PDR) UNTUK NAVIGASI WHEELED MOBILE ROBOT

Muhammad Farih
NRP 07111540000104

Dosen Pembimbing
Mochammad Sahal, S.T., M.Sc.
Ir. Rusdhianto Effendi A. K., MT.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2019

FINAL PROJECT – EE184801

DESIGN OF LEADER-FOLLOWER NAVIGATION SYSTEM BASED ON PEDESTRIAN DEAD-RECKONING FOR WHEELED MOBILE ROBOT NAVIGATION.

Muhammad Farih
NRP 07111540000104

Supervisor
Mochammad Sahal, S.T., M.Sc.
Ir. Rusdhianto Effendi A. K., MT.

DEPARTMENT OF ELECTRICAL ENGINEERING
Faculty of Electrical Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2019

PERNYATAAN KEASLIAN
TUGAS AKHIR

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

"**Perancangan Sistem Navigasi Leader-Follower berbasiskan Pedestrian Dead-Reckoning (PDR) untuk Navigasi Wheeled Mobile Robot**"

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, 28 Mei 2019

Muhammad Farih
Nrp 07111540000104

PERANCANGAN SISTEM NAVIGASI LEADER-FOLLOWER BERBASISKAN PEDESTRIAN DEAD-RECKONING (PDR) UNTUK NAVIGASI WHEELED MOBILE ROBOT

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik
Pada
Bidang Studi Teknik Sistem Pengaturan
Departemen Teknik Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui

Dosen Pembimbing I,	Dosen Pembimbing II,
Mochammad Sahal, S.T., M.Sc.	Ir. Rusdhianto Effendi A. K., MT.
NIP. 197011191998021002	NIP. 195704241985021001

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

SURABAYA
JULI 2019

PERANCANGAN SISTEM NAVIGASI LEDER-FOLLOWER BERBASISKAN PEDESTRIAN DEAD-RECKONING (PDR) UNTUK NAVIGASI WHEELED MOBILE ROBOT

Muhammad Farih
07111540000104

Dosen Pembimbing I: Mochammad Sahal, S.T., M.Sc.
Dosen Pembimbing II: Ir. Rusdhianto Effendi A. K., MT.

ABSTRAK

Sistem navigasi merupakan hal yang wajib dimiliki seluruh wahana tak berawak saat ini baik itu wahana darat, air, maupun udara. Sistem navigasi dapat dibagi menjadi independen dan *leader-follower*. Sistem pemosisian global (GPS) saat ini berperan sangat penting untuk pemosisian setiap wahana. Namun GPS memiliki kekurangan dimana sinyalnya akan terdistorsi ketika penerima GPS berada didalam ruangan. Pada penelitian ini, dirancang sistem navigasi *leader-follower* berbasiskan *pedestrian dead-reckoning* (PDR) dengan *follower* berupa robot beroda. Model *artificial neural network* digunakan sebagai model panjang langkah *leader* sedangkan *leader heading* didapat dari magnetometer. Kedua data tersebut kemudian dikirim ke *follower* melalui *wi-fi*. Model linear digunakan untuk memodelkan jarak tempuh robot dengan *heading* berasal dari magnetometer. Hasil percobaan menunjukkan model terbaik untuk panjang langkah adalah model ANN dengan satu *hidden layer* dengan empat *neuron* dimana memiliki eror *training* 4.65 cm dan eror *testing* 5.04 cm dengan menggunakan total 614 sampel langkah. Kemudian model jarak tempuh robot dengan regresi linear 25 data menghasilkan eror 2.0956 cm. Akhirnya eror terbesar untuk *heading* dari dua magnetometer dengan 28 titik pengujian adalah 39.262°.

Kata Kunci: Sistem Navigasi *Leader-Follower*, *Pedestrian Dead Reckoning*, Magnetometer, *Artificial Neural Network*

DESIGN OF LEADER-FOLLOWER NAVIGATION SYSTEM BASED ON PEDESTRIAN DEAD-RECKONING FOR WHEELED MOBILE ROBOT NAVIGATION

Muhammad Farih
07111540000104

Supervisor I: Mochammad Sahal, S.T., M.Sc.
Supervisor II: Ir. Rusdhianto Effendi A. K., MT.

ABSTRACT

Navigation systems are the first things to have for all types of unmanned vehicles whether it is land, water, or air vehicles. Navigation systems are divided to independent and leader-follower. Today, Global Positioning System (GPS) plays an important role for vehicle positioning. Nevertheless, GPS has a weakness that the signals will be distorted when the GPS receivers are located indoor environments. In this research, leader-follower navigation systems based on pedestrian dead reckoning (PDR) are designed with wheeled mobile robot as follower. Artificial Neural Network (ANN) is used as stride length model and the leader heading is obtained from magnetometer. Both data are then sent to follower through wi-fi. Linear model is used as robot distance model and follower heading are also obtained from it's magnetometer. The test show that the best model for predicting stride length is ANN model with one hidden layer and four neuron units which has 4.65 cm training error and 5.04 cm testing error using 614 stride examples. As for robot distance model shows that the error is 2.0956 cm using 25 data samples. Finally, the biggest heading error is 39.262 ° which is tested from two magnetometers with 28 testing points.

Key Words: *Leader-Follower Navigation Systems, Pedestrian Dead Reckoning, Magnetometer, Artificial Neural Network*

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT, berkat rahmat dan karunia-Nya, penulis dapat menyelesaikan laporan Tugas Akhir berjudul "**Perancangan Sistem Navigasi Leader-Follower berbasiskan Pedestrian Dead-Reckoning (PDR) untuk Navigasi Wheeled Mobile Robot**" untuk memenuhi syarat kelulusan pada Bidang Studi Teknik Sistem Pengaturan, Departemen Teknik Elektro, Fakultas Teknologi Elektro, Institut Teknologi Sepuluh Nopember, Surabaya.

Buku laporan tugas akhir ini dapat terselesaikan berkat bantuan dari Allah SWT. Penulis ingin mengucapkan rasa terima kasih kepada kedua orangtua yang selalu sabar dan mendukung penulis dengan semua bantuan yang dapat diberikan, baik berupa materil maupun non-materil, berkatnya penulis dapat menyelesaikan buku laporan. Terima kasih kepada Bapak Mochammad Sahal dan Bapak Rusdhianto Effendi A. K. selaku dosen pembimbing atas masukan, arahan dan ilmu yang disalurkan kepada penulis dalam menyelesaikan buku laporan ini. Terima kasih juga penulis sampaikan kepada dosen-dosen Bidang Studi Teknik Sistem Pengaturan atas ilmu-ilmu yang disalurkan, baik melalui perkuliahan maupun non-perkuliahan. Terakhir penulis juga mengucapkan terima kasih kepada keluarga Laboratorium Sistem dan Sibernetika AJ204 beserta teman-teman lain yang telah membantu serta berjuang bersama penulis.

Penulis berharap laporan ini dapat memberikan manfaat kepada pembaca, baik secara langsung maupun tak langsung. Laporan ini memang masih jauh dari sempurna, oleh karenanya penulis mengharapkan kritik dan saran yang membangun dari pembaca.

Surabaya, 28 Mei 2019

Penulis

DAFTAR ISI

JUDUL TUGAS AKHIR ..... **i**

PERNYATAAN KEASLIAN TUGAS AKHIR ..... **v**

PENGESAHAN TUGAS AKHIR ..... Error! Bookmark not defined.

ABSTRAK ..... **ix**

ABSTRACT ..... **xi**

KATA PENGANTAR ..... **xiii**

DAFTAR ISI ..... **xv**

DAFTAR GAMBAR ..... **xix**

DAFTAR TABEL ..... **xxiii**

BAB 1 PENDAHULUAN **1**

1.1. Latar Belakang 1
1.2. Perumusan Masalah 2
1.3. Batasan Masalah 2
1.4. Tujuan 2
1.5. Metodologi 2
1.5.1. Desain Penelitian 2
1.5.2. Tipe Penelitian 3
1.5.3. Teknik Penelitian 3
1.5.4. Proses Penelitian 3
1.6. Sistematika 4
1.7. Relevansi 5

BAB 2 TEORI PENUNJANG **7**

2.1. Unit Pengukuran Inersia 7
2.1.1. Akselerometer 7
2.1.2. Giroskop 8
2.1.3. Magnetometer 9
2.2. Message Queuing Telemetry Transport 12
2.3. Platform Elektronik Terintegrasi 12

2.3.1. Arduino 13
2.3.2. ESP8266 14
2.4. Model Panjang Langkah 14
2.4.1. Model Nonlinear 14
2.4.2. Model Frekuensi 15
2.5. Artificial Neural Network 15
2.5.1. Struktur Dasar 16
2.5.2. Standardisasi Data 17
2.5.3. Feed-forward 17
2.5.4. Fungsi Aktivasi 18
2.5.5. Algoritma Optimisasi 19
2.6. Mobile Robot 23
2.6.1. Robot Locomotion 23
2.6.2. Differential Drive Kinematics 24

BAB 3 PERANCANGAN SISTEM 27

3.1. Sistem Leader 27
3.1.1. Komponen 29
3.1.2. Perancangan PCB 30
3.1.3. Algoritma 32
3.2. Sistem *Server* 33
3.2.1. Komponen 34
3.2.2. Perancangan Aplikasi 34
3.2.3. Algoritma 35
3.3. Sistem Follower 41
3.3.1. Komponen 42
3.3.2. Perancangan PCB 44
3.3.3. Algoritma 46
3.4. Perancangan Pengujian 48
3.4.1. Pengujian Giroskop 49
3.4.2. Pengujian Magnetometer 49
3.4.3. Pengujian Algoritma Deteksi Langkah 49
3.4.4. Pengujian Model Panjang Langkah 49
3.4.5. Pengujian Model Jarak Tempuh Follower 50
3.4.6. Pengujian Sistem Navigasi Leader 50
3.4.7. Pengujian Sistem Navigasi Follower 51

3.4.8. Pengujian Sistem Navigasi Leader-Follower 51

BAB 4 HASIL DAN ANALISA 53

4.1. Pengujian Giroskop 53
4.2. Pengujian Magnetometer 54
4.2.1. Pengujian Kalibrasi 54
4.2.2. Pengujian Offset 54
4.3. Pengujian Algoritma Deteksi Langkah 55
4.4. Pengujian Model Panjang Langkah 57
4.4.1. Integral Langsung 57
4.4.2. Model Nonlinear 58
4.4.3. Model Frekuensi 58
4.4.4. Model ANN 59
4.4.5. Perbandingan Model 61
4.5. Pengujian Model Jarak Tempuh Follower 61
4.6. Pengujian Sistem Navigasi Leader 62
4.7. Pengujian Sistem Navigasi Follower 63
4.8. Pengujian Sistem Navigasi Leader-Follower 65
4.9. Analisa Ketercapaian Spesifikasi Desain 66

BAB 5 KESIMPULAN DAN SARAN 69

5.1. Kesimpulan 69
5.2. Saran 69

DAFTAR PUSTAKA 71

LAMPIRAN 73

Lampiran 1. Program Kalibrasi Giroskop [Arduino] 73
Lampiran 2. Program Start *Server* MQTT [Kotlin] 73
Lampiran 3. Program Kalibrasi Otomatis Magnetometer [Kotlin] 74
Lampiran 4. Program Publish rawData IMU *Leader* [ESP8266] .. 74
Lampiran 5. Program Deteksi Langkah [Matlab] 75
Lampiran 5.1. Fungsi detect_start_stride(gyr(i,3)) 76
Lampiran 5.2. Fungsi detect_max_acc(acc(i,1),acc_m1(1)) 76
Lampiran 5.3. Fungsi detect_min_acc(acc(i,1),acc_m1(1)) 77
Lampiran 5.4. Fungsi detect_start_stride(gyr(i,3)) 77
Lampiran 6. Program Tilt Compensated Heading [Kotlin] 77

Lampiran 7. Program Learning Model ANN [Python] 78
Lampiran 8. Program Inference Model ANN [Kotlin].....78
Lampiran 9. Program Subscribe Waypoint Robot [Arduino].....80
Lampiran 10. Program Maju & Belok Robot [Arduino] 81

RIWAYAT HIDUP PENULIS 83

DAFTAR GAMBAR

Gambar 1.1 Diagram Alir Proses Penelitian 3
Gambar 2.1 Struktur Akselerometer tipe MEMS [12] 8
Gambar 2.2 Struktur Mekanik Giroskop tipe MEMS [20]..... 8
Gambar 2.3 Distorsi *Hard Iron* [19] 9
Gambar 2.4 Distorsi *Soft Iron* [19] 10
Gambar 2.5 Koreksi Distorsi [19]..... 11
Gambar 2.6 Struktur Perpesanan MQTT [4]..... 12
Gambar 2.7 Komponen Platform Elektronik Terintegrasi [17]..... 13
Gambar 2.8 Arduino UNO [3] 13
Gambar 2.9 Keluaran Pin ESP8266 [1] 14
Gambar 2.10 Prinsip Kerja Sebuah *Artificial Neuron* [8] 15
Gambar 2.11 Contoh Sederhana Sebuah ANN [8]..... 16
Gambar 2.12 *Graph Model* 3 *Layer* dengan Fungsi Aktivasi [5]..... 16
Gambar 2.13 *Feed-forward* ANN [8] 17
Gambar 2.14 *Plot* Fungsi Aktivasi Sigmoid [5]..... 18
Gambar 2.15 *Plot* Fungsi Aktivasi ReLU [5] 19
Gambar 2.16 *Gradient descent* tanpa momentum (kiri) dan dengan momentum (kanan) [15]..... 20
Gambar 2.17 Sistem Koordinat Global Robot ($\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}$) [6]..... 23
Gambar 2.18 Contoh Skema Robot Jenis *Non-Holonomic Drive* [2] ... 24
Gambar 2.19 Rotasi Robot Mengelilingi Suatu Titik ICC [6] 25
Gambar 3.1 Diagram Blok Sistem Keseluruhan 27
Gambar 3.2 Gambaran Umum Hubungan antar Subsistem 27
Gambar 3.3 Diagram Blok Sistem *Leader* 28
Gambar 3.4 Komponen *Leader* di Sepatu Sebelah Kanan 28
Gambar 3.5 Konfigurasi Komponen Sistem *Leader* 28
Gambar 3.6 Mode *Schematic* PCB Sistem *Leader*..... 30
Gambar 3.7 Mode *Board* PCB Sistem *Leader* 31
Gambar 3.8 Hasil Cetak PCB..... 31
Gambar 3.9 *Flowchart* Sistem *Leader* 32
Gambar 3.10 Diagram Blok Sistem *Server* 33
Gambar 3.11 Doogee BL9000 34
Gambar 3.12 *Server* MQTT telah Aktif 35

Gambar 3.13 *Flowchart* Sistem *Server* (Bagian 1).....36
Gambar 3.14 *Flowchart* Sistem *Server* (Bagian 2).....37
Gambar 3.15 Fase Satu Siklus Berjalan [23].....38
Gambar 3.16 Data Akselerometer dalam Tujuh Langkah 39
Gambar 3.17 Data Giroskop dalam Tujuh Langkah 39
Gambar 3.18 Konfigurasi Komponen Sistem *Follower* 41
Gambar 3.19 Diagram Blok Sistem *Follower* (*Heading*) 42
Gambar 3.20 Diagram Blok Sistem *Follower* (Gerakan Maju).....42
Gambar 3.21 Mode *Schematic* PCB Sistem *Follower*.....44
Gambar 3.22 Mode *Board* PCB Sistem *Leader*.....45
Gambar 3.23 Hasil Cetak PCB 45
Gambar 3.24 *Flowchart* Sistem *Follower* (Bagian 1).....46
Gambar 3.25 *Flowchart* Sistem *Follower* (Bagian 2).....47
Gambar 3.26 Data Jarak Tempuh Berdasarkan Waktu Tempuh.....48
Gambar 3.27 Denah Pengujian (*Indoor*).....50
Gambar 3.28 Denah Pengujian (*Outdoor*) 51
Gambar 4.1 Giroskop Sebelum dan Sesudah Kalibrasi 53
Gambar 4.2 Sinyal Akselerometer.....55
Gambar 4.3 Fase Satu Siklus Berjalan.....56
Gambar 4.4 Jumlah Langkah Terdeteksi 56
Gambar 4.5 Sinyal Giroskop 57
Gambar 4.6 Perbandingan Hasil Integral Langsung dengan Nilai Sebenarnya.....57
Gambar 4.7 Perbandingan Hasil Estimasi Model Nonlinear dengan Nilai Sebenarnya.....58
Gambar 4.8 Perbandingan Hasil Estimasi Model Frekuensi dengan Nilai Sebenarnya.....58
Gambar 4.9 MSE dari *Training* Data Model ANN.....59
Gambar 4.10 Perbandingan Hasil Estimasi Model ANN dengan Nilai Sebenarnya.....60
Gambar 4.11 Struktur Model ANN Satu *Hidden Layer* Empat *Neuron* 60
Gambar 4.12 Hasil Regresi Linear Model Jarak Tempuh *Follower* 62
Gambar 4.13 Lintasan Hasil Sistem Navigasi *Leader* (*Indoor*).....62
Gambar 4.14 Lintasan Hasil Sistem Navigasi *Leader* (*Outdoor*) 63
Gambar 4.15 Lintasan Hasil Sistem Navigasi *Follower* (*Indoor*) 64
Gambar 4.16 Lintasan Hasil Sistem Navigasi *Follower* (*Outdoor*).....64

Gambar 4.17 Lintasan Hasil Sistem Navigasi *Leader-Follower* (*Indoor*) 65
Gambar 4.18 Lintasan Hasil Sistem Navigasi *Leader-Follower* (*Outdoor*) 66

DAFTAR TABEL

Tabel 3.1 Komponen Sistem *Leader* 29
Tabel 3.2 Komponen Sistem *Follower* 43
Tabel 3.3 Spesifikasi Desain Sistem 48
Tabel 4.1 Magnetometer Sebelum dan Setelah Kalibrasi 54
Tabel 4.2 Offset Magnetometer 54
Tabel 4.3 RMSE *Offset* Magnetometer 55
Tabel 4.4 Uji ANN 59
Tabel 4.5 Perbandingan RMSE Model Panjang Langkah 61
Tabel 4.6 Uji Model Jarak Tempuh 61
Tabel 4.7 RMSE Pengujian Sistem Navigasi *Leader* 63
Tabel 4.8 RMSE Pengujian Sistem Navigasi *Follower* 65
Tabel 4.9 Hasil Pengujian Sistem Navigasi *Leader-Follower* 65
Tabel 4.10 Perbandingan Hasil dengan Spesifikasi Desain Sistem 66

BAB 1
PENDAHULUAN

1.1. Latar Belakang

Di era Revolusi Industri 4.0 ini, robot menjadi sebuah teknologi yang semakin dibutuhkan manusia dalam kehidupan sehari-hari. Definisi robot sendiri merupakan sebuah mesin yang mampu melakukan sesuatu sesuai dengan kontrol/perintah pengguna yang diberikan baik itu secara manual maupun otomatis. Dikatakan sebagai robot manual apabila seluruh pergerakannya dikontrol pengguna dan dikatakan otomatis apabila pergerakan robot dikontrol robot itu sendiri dengan/tanpa adanya kontrol pengguna secara terus-menerus. Untuk robot manual tidak perlu adanya sistem navigasi internal, karena seluruh pergerakannya dikendalikan oleh pengguna. Namun untuk robot otomatis, tentu memerlukan sistem navigasi internal sehingga memungkinkan untuk melakukan pergerakan dari satu titik ke titik yang lain.

Sistem navigasi robot dapat berjalan karena terdapat masukan dari berbagai sensor misalnya akselerometer, giroskop, magnetometer, *global positioning system* (GPS), bahkan wi-fi dan *bluetooth*. Selain itu, sistem navigasi juga dibagi menjadi dua yakni independen dan *leader-follower*. Navigasi *leader-follower* dapat dimanfaatkan untuk berbagai hal misalnya saja robot pembawa barang-barang berat militer dan juga robot pembawa bagasi di bandara. Apabila robot diaplikasikan di bandara, tentu sensor GPS tidak dapat berfungsi dengan baik karena gelombang mikro akan dilemahkan dan tersebar secara acak oleh atap, dinding, maupun objek lain.

Oleh karena itu, pada tugas akhir ini akan dirancang sistem navigasi *leader-follower* menggunakan sensor *Inertial Measurement Unit* (IMU) 9-*Degree of Freedom* (*DoF*) untuk menentukan pergerakan relatif *leader* yang kemudian akan di pantau secara kontinyu oleh *follower*. Sensor IMU 9-DoF sendiri terdiri dari akselerometer, giroskop, dan magnetometer. Akselerometer tidak bersifat drift untuk waktu yang lama akan tetapi tidak stabil untuk perubahan secara tiba-tiba untuk waktu yang singkat sedangkan giroskop bersifat drift untuk waktu yang lama. Dengan demikian pada tugas akhir ini juga akan dirancang sebuah filter yang akan menggabungkan berbagai sensor untuk meningkatkan akurasi pengukuran.

1.2. Perumusan Masalah

Masalah yang akan dibahas pada tugas akhir ini adalah sebagai berikut:

1. Bagaimana IMU 9-DoF pada *leader* mengirim data ke *server* secara *wireless* dan *real-time*.
2. Bagaimana *server* mengestimasi panjang lintasan dan *heading* berdasarkan data IMU 9-DoF.
3. Bagaimana *server* mengirim data hasil estimasi ke *follower*.
4. Bagaimana *follower* melaksanakan perintah berdasarkan data hasil estimasi dan sensor yang tersedia pada *follower*.

1.3. Batasan Masalah

Beberapa batasan masalah pada tugas akhir ini adalah sebagai berikut:

- Sensor yang digunakan hanya sensor IMU 9-DoF dan magnetometer.
- Keluaran magnetometer dibatasi hanya berisi nilai pengukuran yang terdistorsi oleh *soft iron* dan *hard iron*
- Lintasan tiap langkah diasumsikan selalu lurus
- Robot memiliki dua penggerak roda dengan satu roda tambahan
- Kecepatan roda robot selalu konstan

1.4. Tujuan

Tujuan yang ingin dicapai pada tugas akhir ini adalah sebagai berikut:

1. IMU 9-DoF pada *leader* dapat mengirim data ke *server* secara *wireless* dan *real-time*.
2. *Server* dapat mengestimasi panjang lintasan dan *heading* berdasarkan data IMU 9-DoF.
3. *Server* dapat mengirim data hasil estimasi ke *follower*.
4. *Follower* dapat melaksanakan perintah berdasarkan data hasil estimasi dan sensor yang tersedia pada *follower*.

1.5. Metodologi

Metodologi dalam penelitian ini meliputi beberapa bagian diantaranya desain, tipe, teknik, dan proses penelitian.

1.5.1. Desain Penelitian

Pada penelitian ini digunakan *true experimental designs*. Desain penelitian ini berupaya untuk mengontrol setiap variabel dan mengamati efek ketika variabel dimanipulasi. Penelitian ini memiliki subjek dan

objek berupa manusia sebagai subjek dan mobile robot sebagai objek yang masing-masing didalamya memiliki beberapa variabel. Hipotesis (prediksi) diformulasikan terlebih dahulu sebelum melakukan eksperimen untuk menentukan variabel-variabel apa saja yang digunakan untuk pengujian dan bagaimana variabel tersebut dikontrol dan diukur. Variabel dalam penelitian ini meliputi sensor akselero, giro, dan magneto. Semua variabel diukur menggunakan sensor IMU 9-DoF dengan ESP8266 sebagai mikrokontroler yang dilengkapi modul wi-fi.

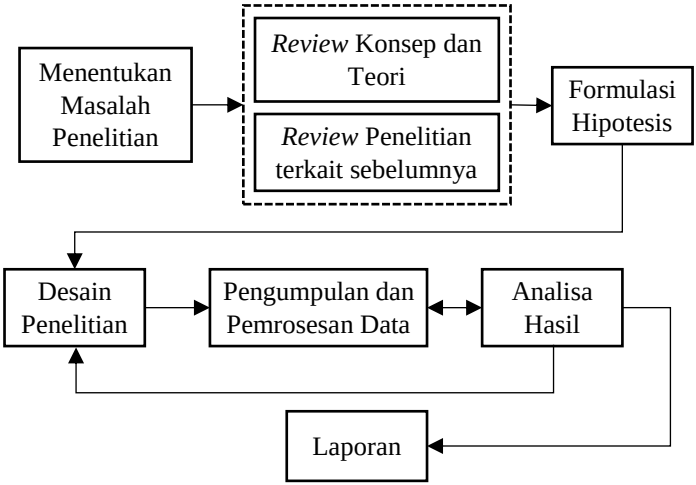
1.5.2. Tipe Penelitian

Penelitian ini memiliki tipe *applied and empirical*. *Applied* karena penelitian ini dapat dimanfaatkan untuk mengatasi masalah yang dihadapi masyarakat umum. *Empirical* karena penelitian ini berbasiskan data yang diukur melalui eksperimen yang kemudian divalidasi dengan teori yang ada.

1.5.3. Teknik Penelitian

Penelitian ini menggunakan teknik penelitian laboratorium yang mana menggunakan perangkat elektronik dan berbagai macam sensor untuk pengambilan data.

1.5.4. Proses Penelitian



Gambar 1.1 Diagram Alir Proses Penelitian

Proses penelitian dapat digambarkan pada diagram alir pada Gambar 1.1 dimana dimulai dari penentuan permasalahan dalam penelitian yang

meliputi permodelan panjang langkah menggunakan IMU 9-DoF, algoritma *heading* menggunakan magnetometer, pengiriman data menggunakan protokol MQTT, serta permodelan jarak tempuh robot. Kemudian dipelajari teori dan penelitian sebelumnya yang terkait dengan permasalahan-permasalahan tersebut. Setelah itu, dapat diformulasikan hipotesis dan perancangan penelitian. Setelah dirancang, dilakukan pengambilan dan pemrosesan data serta analisa hasil data. Terakhir, hasil analisa akan ditulis dalam laporan penelitian.

1.6. Sistematika

Penulisan laporan tugas akhir ini mengikuti sistematika sebagai berikut.

BAB 1 Pendahuluan

Bab ini menguraikan tentang latar belakang dari pemilihan judul tugas akhir. Permasalahan yang ada, tujuan dari pelaksanaan tugas akhir, metodologi penelitian, sistematika penulisan, dan relevansi dari tugas akhir ini.

BAB 2 Teori Penunjang

Pada bab ini dijabarkan teori-teori penunjang dan teori-teori dasar yang digunakan untuk penyelesaian tugas akhir ini meliputi *inertial measurement unit*, *message queuing telemetry transport*, platform elektronik terintegrasi, model panjang langkah, *artificial neural network*, dan juga *mobile robot*.

BAB 3 Perancangan Sistem

Dalam bab ini, dilakukan perancangan dari subsistem-subsistem. Perancangan sistem ini berisi komponen-komponen yang digunakan tiap subsistem, perancangan PCB untuk sistem *leader* dan *follower*, dan perancangan algoritma tiap subsistem. Selain itu, bab ini juga memuat perancangan dari beberapa pengujian diantaranya pengujian komponen, algoritma, dan sistem navigasi.

BAB 4 Hasil dan Analisa

Pada bab ini akan ditampilkan hasil keseluruhan pengujian yang telah dirancang sesuai dengan bab 3.

BAB 5 Kesimpulan dan Saran

Bab ini berisikan kesimpulan yang didapat dari analisa hasil implementasi, serta pemberian saran untuk penelitian berikutnya mengenai topik terkait.

1.7. Relevansi

Hasil dari tugas akhir ini diharapkan dapat memberikan manfaat terhadap khalayak umum terkait metode pengiriman data secara wireless dan real-time serta metode estimasi panjang lintasan dan heading berdasarkan data IMU 9-DoF. Selain itu, tugas akhir ini juga diharapkan dapat diimplementasikan sebagai pelengkap dari sistem pemosisian lokal yang sudah ada.

BAB 2
TEORI PENUNJANG

2.1. Unit Pengukuran Inersia

Unit pengukuran inersia (IMU) 9-DoF adalah satu paket sensor yang terdiri dari tiga sensor yakni akselerometer, giroskop, dan magnetometer yang dapat digunakan untuk melacak perubahan dan orientasi objek. Sekarang ini, IMU 9-DoF yang sering digunakan bertipe *Micro Electro Mechanical Systems* (MEMS). Teknologi MEMS dibangun di atas infrastruktur fabrikasi inti yang dikembangkan untuk sirkuit terintegrasi silikon. Akselerometer mengukur percepatan linear dalam atau yang terjadi pada sensor (termasuk percepatan gravitasi), giroskop mengukur kecepatan angular dalam pada frame sensor, dan magnetometer mengukur kuat medan magnet dalam atau pada frame sensor.

2.1.1. Akselerometer

Akselerometer merupakan sensor yang mengukur percepatan linear yang dialami oleh sensor. Nilai yang terukur pada sensor bukanlah nilai percepatan linear yang benar karena telah ditambah oleh percepatan gravitasi, noise dan juga bias.

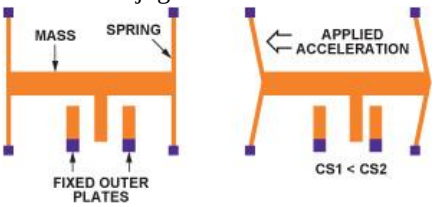
$$\mathrm{a_{meas}} = \mathrm{a_{imeas}} * \mathrm{a_{sfcc}} + \mathrm{a_{bias}} + \eta \tag{2.1}$$

Dimana,

- a_{meas} adalah percepatan terukur
- $a_{imeas} = a_b + \omega_b \times (\omega_b \times d) + (\dot{\omega_b}) \times d - g$
  dengan,
  - a_b adalah percepatan sebenarnya
  - ω_b adalah kecepatan sudut
  - d adalah jarak akselerometer dengan center of gravity
  - g adalah gravitasi
- a_{sfcc} adalah matriks 3x3 dari scale factor pada diagonalnya dan misalignment terms pada selain diagonalnya.
- a_{bias} adalah bias akselerometer
- η adalah *zero-mean Gaussian noise*

Akselerometer memiliki keuntungan pada akurasinya ketika digunakan pada waktu yang lama karena bersifat tidak memiliki *drift* dan pusat gravitasi bumi biasanya tidak berubah, akan tetapi akselerometer

memiliki masalah dimana tidak dapat dipercaya dalam waktu singkat karena gerakan tiba-tiba dan juga memiliki noise.



Gambar 2.1 Struktur Akselerometer tipe MEMS [12]

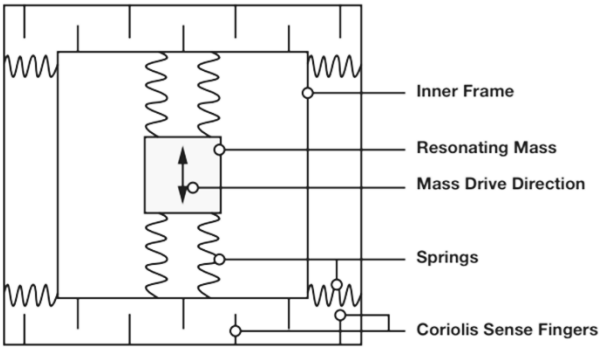
2.1.2. Giroskop

Giroskop merupakan sensor yang mengukur kecepatan sudut yang dialami oleh sensor. Sama seperti akselerometer, giroskop juga tidak mengukur kecepatan sudut sebenarnya melainkan juga bias dan juga noise sistem.

$$\omega_{\text{meas}} = \omega_b * \omega_{\text{sfcc}} + \omega_{\text{bias}} + \text{G}s \times \omega_{gsens} \quad (2.2)$$

dimana

- ω_{meas} adalah kecepatan sudut terukur
- ω_b adalah kecepatan sudut sebenarnya
- ω_{sfcc} adalah matriks 3x3 dari scale factor pada diagonalnya dan misalignment terms pada selain diagonalnya.
- Gs adalah Gs giroskop
- ω_{gsens} adalah bias sensitivitas-g
- η adalah *zero-mean Gaussian noise*



Gambar 2.2 Struktur Mekanik Giroskop tipe MEMS [20]

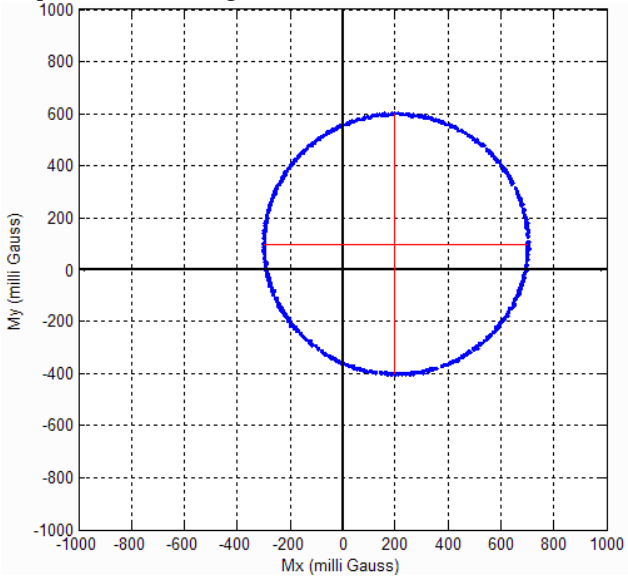
Giroskop memiliki keuntungan pada akurasinya ketika digunakan pada waktu yang singkat, akan tetapi tidak dapat dipercaya dalam waktu yang lama karena bersifat *drift.*

2.1.3. Magnetometer

Magnetometer merupakan sensor yang mengukur medan magnet bumi yang biasa digunakan untuk mengetahui arah mata angin. Magnetometer bersifat distorsi ketika terdapat objek atau elektronik berbahan metal. Selain itu pengukuran magnetometer sangat bergantung dengan kalibrasi yang bagus.

2.1.3.1. Distorsi Hard Iron

Distorsi *hard iron* adalah distorsi yang disebabkan oleh benda-benda yang menghasilkan medan magnetik. Ketika sebuah benda yang demikian berada didekat magnetometer maka akan menyebabkan sebuah bias permanen pada keluaran magnetometer.



Gambar 2.3 Distorsi *Hard Iron* [19]

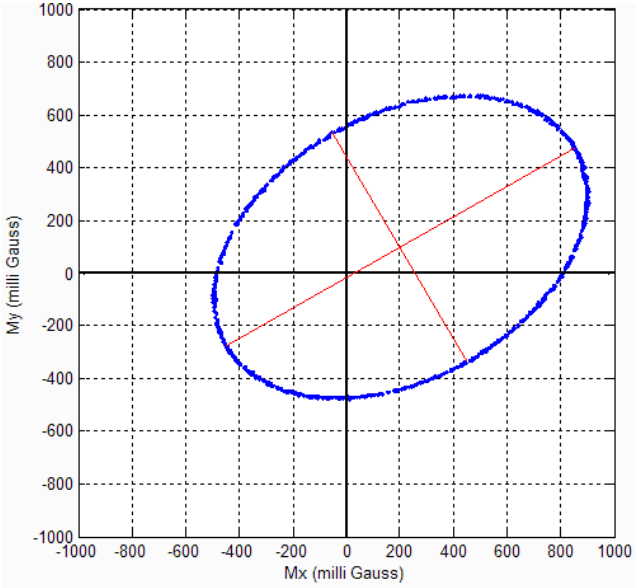
Distorsi *hard iron* dapat dieliminasi dengan mencari offset pada masing-masing sumbu magnetometer. Offset dapat dicari dengan mencari

nilai maksimum dan minimum tiap sumbu. *Pseudocode* untuk mencari offset distorsi *hard iron* dapat dilihat pada persamaan (2.3) dibawah ini.

$$offset_{m_x} = \frac{\max(m_x) + \min(m_x)}{2}$$
$$offset_{m_y} = \frac{\max(m_y) + \min(m_y)}{2} \quad (2.3)$$
$$offset_{m_z} = \frac{\max(m_z) + \min(m_z)}{2}$$

2.1.3.2. Distorsi Soft Iron

Distorsi *soft iron* dapat dikatakan sebagai defleksi terhadap medan magnet. Distorsi ini umumnya disebabkan oleh benda-benda berbahan besi atau nikel.



Gambar 2.4 Distorsi *Soft Iron* [19]

Distorsi *soft iron* tidak dapat dieliminasi secara mudah seperti menghilangkan konstanta pada distorsi *hard iron*. Koreksi *soft iron* umumnya bersifat *computation expensive* dan melibatkan matriks berukuran 3×3. Terdapat algoritma untuk koreksi yang lebih *computation cheap* dengan menggunakan faktor skala [21]. Sebelum

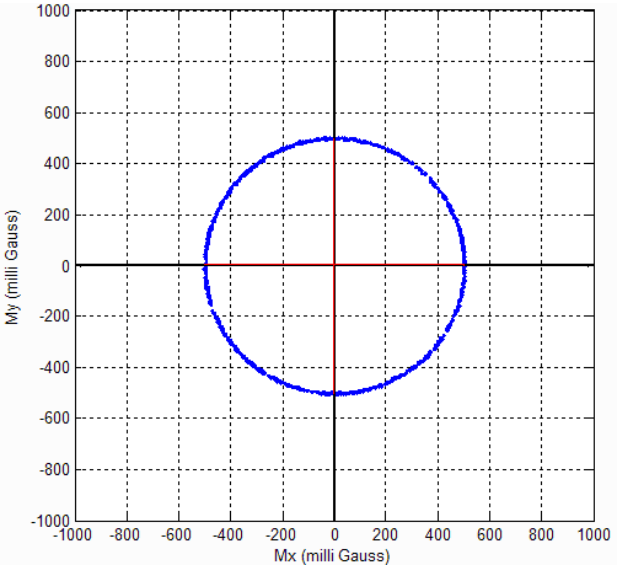
mencari faktor skala, perlu untuk mencari nilai maksimum dan minimum masing-masing sumbu seperti pada persamaan (2.4).

$$\begin{aligned} avg_{\delta m_x} &= \frac{\max(m_x) - \min(m_x)}{2} \\ avg_{\delta m_y} &= \frac{\max(m_y) - \min(m_y)}{2} \\ avg_{\delta m_z} &= \frac{\max(m_z) - \min(m_z)}{2} \\ avg_{\delta} &= \frac{\left(avg_{\delta m_x} + avg_{\delta m_y} + avg_{\delta m_z}\right)}{3} \end{aligned} \tag{2.4}$$

Setelah didapat variabel-variabel pada (2.4), faktor skala *soft iron* dapat dicari menggunakan persamaan berikut:

$$\begin{aligned} scale_{m_x} &= \frac{avg_{\delta}}{avg_{\delta m_x}} \\ scale_{m_y} &= \frac{avg_{\delta}}{avg_{\delta m_y}} \\ scale_{m_z} &= \frac{avg_{\delta}}{avg_{\delta m_z}} \end{aligned} \tag{2.5}$$

2.1.3.3. Koreksi Distorsi

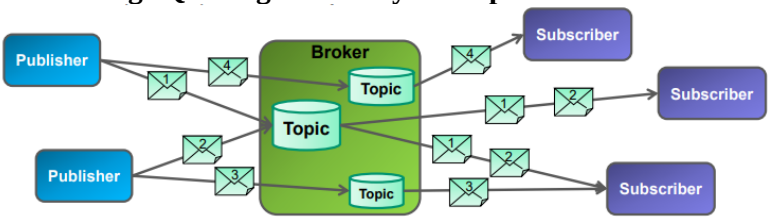


Gambar 2.5 Koreksi Distorsi [19]

Setelah didapatkan *offset* dan faktor skala, pengukuran magnetometer dapat dikoreksi dengan persamaan berikut:

$$\begin{aligned} \hat{m}_x &= \left(m_x - offset_{m_x}\right) * scale_{m_x} \\ \hat{m}_y &= \left(m_y - offset_{m_y}\right) * scale_{m_y} \\ \hat{m}_z &= \left(m_z - offset_{m_z}\right) * scale_{m_z} \end{aligned} \tag{2.6}$$

2.2. Message Queuing Telemetry Transport



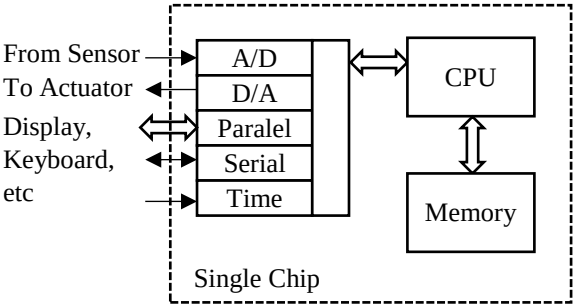
Gambar 2.6 Struktur Perpesanan MQTT [4]

Message Queuing Telemetry Transport (MQTT) adalah protokol ringan yang digunakan dalam transportasi dan antrian perpesanan. Sesuai dengan namanya MQTT cocok untuk digunakan dalam transportasi data *telemetry* (data sensor dan aktor). MQTT sangatlah ringan sehingga cocok untuk aplikasi pada IoT dimana sensor dan aktor berkomunikasi melalui *server* perpesanan MQTT (biasa disebut MQTT *broker*).

MQTT memiliki beberapa karakteristik diantaranya merupakan protokol perpesanan yang ringan, cocok digunakan untuk *bandwidth* rendah, model perpesanan berbasiskan *publish/subscribe* dengan pemisahan data melalui topik.

2.3. Platform Elektronik Terintegrasi

Platform elektronik terintegrasi adalah sebuah perangkat elektronik yang terdiri dari sekumpulan komponen terintegrasi sehingga dapat digunakan secara mandiri untuk kebutuhan pemantauan dan pengaturan. Komponen didalamnya terdiri dari inti prosesor, memori, digital I/O, analog I/O, *interrupt*, dan *timer*. Untuk antarmuka komunikasi, mikrokontroler secara umum memiliki tiga antarmuka komunikasi, diantaranya *serial communication interface* (SCI), *serial peripheral interface* (SPI), dan *inter-IC* (IIC).



Gambar 2.7 Komponen Platform Elektronik Terintegrasi [17]

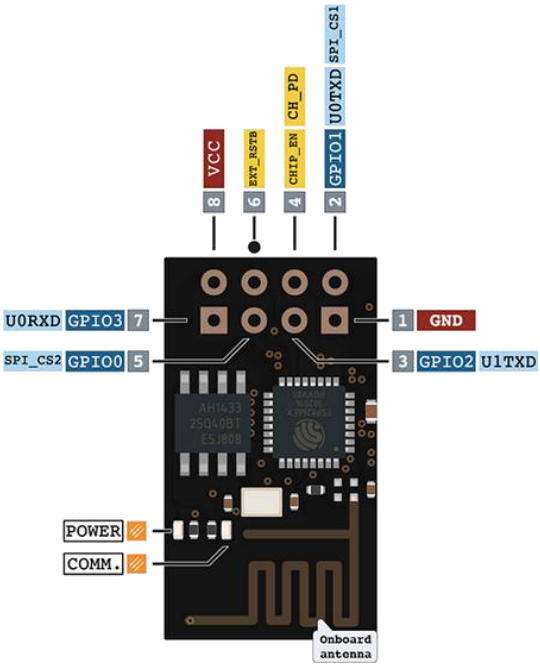
2.3.1. Arduino

Arduino merupakan platform elektronik yang mudah dan populer digunakan pada bidang robotika. Arduino memiliki beberapa jenis diantaranya Arduino Uno, Leonardo, Mega ADK, Ethernet, Due, Yun, Mega 2560, Mini, dan Nano.

Arduino memiliki dua jenis keluaran pin berupa pin digital dan analog. Pin digital adalah sebuan pin keluaran arduino yang dapat mengeluarkan sinyal yang bernilai *HIGH* atau *LOW*. Selain itu, terdapat juga beberapa pin digital yang dapat mengeluarkan sinyal *pulse width modulation* (PWM) yang sering digunakan untuk mengontrol kecepatan motor. Sedangkan pin analog adalah pin yang mengandung 10-bit *analog-to-digital conversion* (ADC).

Gambar 2.8 Arduino UNO [3]

2.3.2. ESP8266



Gambar 2.9 Keluaran Pin ESP8266 [1]

ESP8266 adalah sebuah solusi s*ystem on a chip* (SoC) integrasi tinggi yang memiliki penggunaan daya efisien, desain kompak, dan performa andal yang biasa digunakan pada industri *internet of things* (IoT). ESP8266 memiliki kapabilitas sama seperti mikrokontroler yang dilengkapi dengan konektivitas wi-fi, akan tetapi pin yang dimilikinya terbatas dibandingkan mikrokontroler pada umumnya.

2.4. Model Panjang Langkah

Berdasarkan [23], terdapat dua model yang biasanya digunakan untuk estimasi panjang langkah, yaitu model nonlinear dan model frekuensi.

2.4.1. Model Nonlinear

Model nonlinear merupakan model yang sederhana karena hanya melibatkan satu koefisien akan tetapi memerlukan transformasi sinyal

akselerometer menjadi frame lokal untuk mencari nilai maksimum dan minimum akselerasi vertikal sesuai dengan Persamaan (2.7).

$$L = K \times \sqrt[4]{a_{u_{max}} - a_{u_{min}}} \quad (2.7)$$

dimana L merupakan panjang langkah, $a_{u_{max}}$ (atau $a_{u_{min}}$) adalah akselerasi vertikal maksimum (atau akselerasi vertikal minimum) dalam satu langkah, dan K adalah parameter yang diubah-ubah.

2.4.2. Model Frekuensi

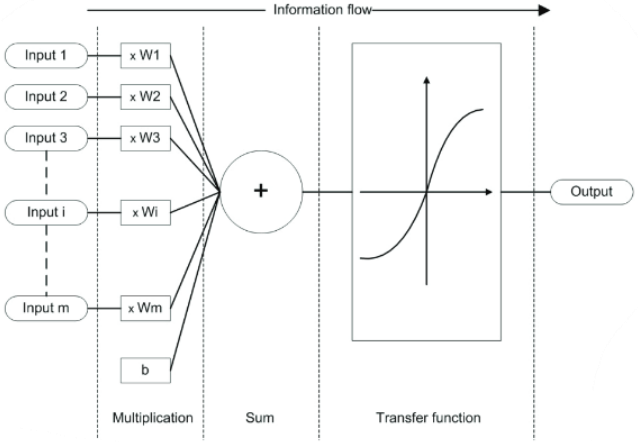
Model frekuensi adalah model sederhana yang melibatkan dua koefisien dan variabel frekuensi dalam satu langkah sesuai dengan Persamaan (2.8).

$$L = a \times f + b \quad (2.8)$$

dimana f adalah frekuensi dalam satu langkah dan a dan b adalah suatu parameter.

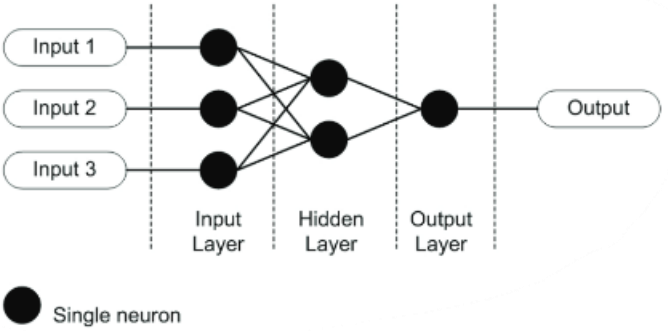
2.5. Artificial Neural Network

Artificial neural network (ANN) adalah sebuah model matematika yang mencoba menyimulasikan struktur dan fungsionalitas dari jaringan syaraf biologis. Blok dasar dari ANN adalah *artificial neuron* yang mana merupakan model matematika sederhana seperti perkalian, penjumlahan, dan aktifasi.



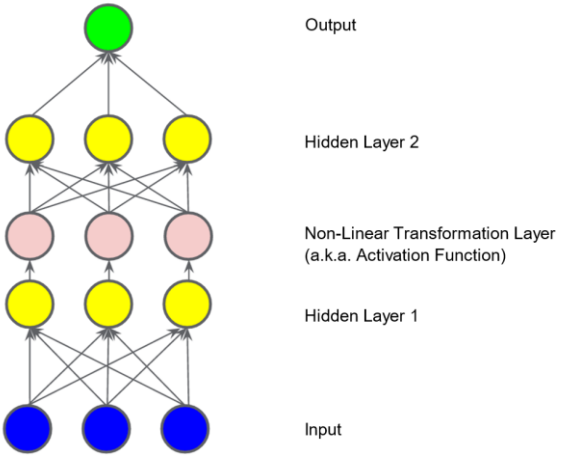
Gambar 2.10 Prinsip Kerja Sebuah *Artificial Neuron* [8]

Walaupun terlihat sederhana, model dari *artificial neuron* akan menjadi sangat efektif ketika *artificial neuron* mulai saling dihubungkan sehingga menjadi sebuah ANN.



Gambar 2.11 Contoh Sederhana Sebuah ANN [8]

2.5.1. Struktur Dasar



Gambar 2.12 *Graph Model* 3 *Layer* dengan Fungsi Aktivasi [5]

ANN memiliki struktur dasar berupa *input layer*, *output layer*, *hidden layer*, dan *non-linear transformation layer* (fungsi aktivasi). *Input layer* adalah *layer* yang berisikan masukan-masukan (*features)*. *Output layer* berisi jumlah dari pembobotan *layer* sebelumnya. *Hidden layer* juga berisi jumlah dari pembobotan *layer* sebelumnya. *Non-linear*

transformation layer merupakan *layer* yang digunakan untuk memodelkan masalah non-linear. *Layer* ini biasa disebut fungsi aktivasi dan tidak ditampilkan didalam *graph*.

2.5.2. Standardisasi Data

Dalam sebuah *multilayer perceptron*, memang tidak diharuskan masukan berada pada rentang 0—1. Akan tetapi terdapat keuntungan ketika data di standardisasi, tetapi bukan pada rentang 0—1 melainkan dengan nilai tengah mendekati 0. Standardisasi memiliki keuntungan akan mempercepat proses *learning* dengan meningkatkan kondisi numerik dari masalah optimisasi. Ketika satu masukan data memiliki rentang 0—1 sedangkan data kedua memiliki rentang 0—1000000, maka sangatlah esensial data untuk distandardisasi, karena kontribusi masukan kesatu akan ditenggelamkan oleh masukan kedua.

Terdapat dua cara yang paling efektif dalam standardisasi data:

- Memiliki rata-rata 0 dan standar deviasi 1

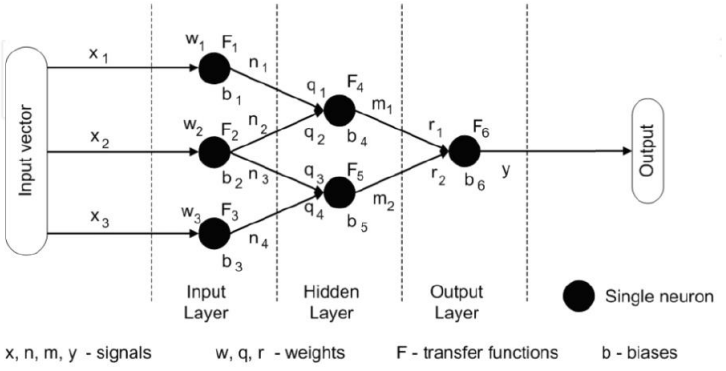
$$data_i = \frac{data_i - mean(\boldsymbol{data})}{std(\boldsymbol{data})} \tag{2.9}$$

  Dimana $data_i$ adalah data ke-i dan $\boldsymbol{data}$ adalah semua sampel *dataset training*.

- Memiliki nilai tengah 0 dan rentang 2 (misal minimum -1 dan maksimal +1)

$$data_i = \frac{data_i - middle(\boldsymbol{data})}{range(\boldsymbol{data})/2} \tag{2.10}$$

2.5.3. Feed-forward



Gambar 2.13 *Feed-forward* ANN [8]

ANN dengan topologi *feed-forward* memiliki satu kondisi dimana informasi harus mengalir dari *input* ke *output* dalam satu arah tanpa umpan balik. *Feed forward* merupakan proses pertama setelah bobot diinisialisasi yang kemudian dilakukan revisi bobot. Dari Gambar 2.13, dapat diperoleh persamaan untuk *feed-forward* sebagai berikut:

$$\begin{aligned} n_1 &= F_1(w_1 x_1 + b_1) \\ n_2 &= F_2(w_2 x_2 + b_2) \\ n_3 &= F_2(w_2 x_2 + b_2) \\ n_4 &= F_3(w_3 x_3 + b_3) \end{aligned} \tag{2.11}$$

$$\begin{aligned} m_1 &= F_4(q_1 n_1 + q_2 n_2 + b_4) \\ m_2 &= F_5(q_3 n_3 + q_4 n_4 + b_5) \end{aligned} \tag{2.12}$$

$$y = F_6(r_1 m_1 + r_2 m_2 + b_6) \tag{2.13}$$

2.5.4. Fungsi Aktivasi

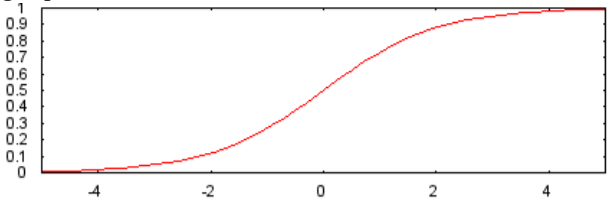
Fungsi aktivasi dapat dipilih dari beberapa fungsi sebagai berikut:

- Sigmoid

  Sigmoid mengubah jumlah dari pembebanan menjadi suatu nilai antara 0—1.

$$F(x) = \frac{1}{1 + e^{-x}} \tag{2.14}$$

  Dengan *plot* berikut ini,



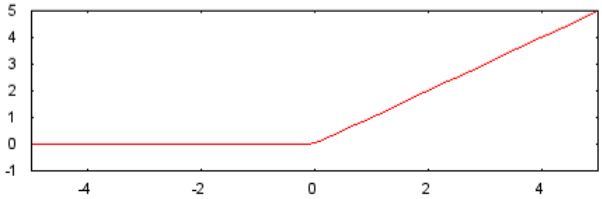
Gambar 2.14 *Plot* Fungsi Aktivasi Sigmoid [5]

- Rectified Linear Unit

  Fungsi aktivasi *Rectified Linear Unit* (ReLU) terkadang bekerja lebih baik dibanding yang lain dengan komputasi yang lebih sederhana.

$$F(x) = \max(0, x) \tag{2.15}$$

  Plot ReLU dapat dilihat pada Gambar 2.15 dibawah ini.



Gambar 2.15 *Plot* Fungsi Aktivasi ReLU [5]

2.5.5. Algoritma Optimisasi

Gradient descent merupakan salah satu algoritma populer untuk melakukan optimisasi pada *neural network*. Di waktu yang sama, banyak *library deep learning* yang mencantumkan berbagai macam algoritma untuk mengoptimalkan *gradient descent* misalnya pada dokumentasi Keras.

Gradient descent merupakan suatu cara untuk meminimumkan fungsi objektif $J(\theta)$ yang diparameterisasikan oleh model parameter $\theta \in \mathbb{R}^d$ dengan memperbarui parameter pada arah sebaliknya dari gradien fungsi objektif $\nabla_\theta J(\theta)$ yang mengacu pada parameter-parameternya. *Learning rate* η menentukan ukuran dari langkah yang diambil untuk mencapai sebuah minimum (lokal).

2.5.5.1. Variasi Gradient Descent

Terdapat tiga variasi *gradient descent* yang dibedakan dari berapa banyak data yang diambil untuk perhitungan gradien fungsi objektif. Berdasarkan jumlah data, dapat ditentukan akurasi pembaruan parameter dan waktu yang digunakan untuk melakukan pembaruan.

1. *Batch Gradient Descent*

   Vanilla gradient descent atau biasa disebut *batch gradient descent* menghitung gradien fungsi objektif tiap parameter menggunakan keseluruhan *dataset*:

   $$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.16}$$

   Karena perlu menghitung gradien untuk seluruh *dataset* hanya untuk memperbarui parameter sekali, *batch gradient descent* menghasilkan komputasi yang lambat. *Batch gradient descent* menjamin akan konvergen ke minimum global untuk permasalahan konveks dan minimum lokal untuk permasalahan non-konveks.

2. *Stochastic Gradient Descent*

   Stochastic gradient descent (SGD) berkebalikan dengan *batch gradient descent* dimana melakukan pembaruan parameter untuk setiap contoh dalam *dataset*:

   $$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^i; y^i) \quad (2.17)$$

   SGD mengatasi komputasi lambat dengan melakukan permbaruan tiap contoh sehingga *training* akan lebih cepat. Dengan iterasi yang cukup, SGD dapat berfungsi namun akan sangat *noisy* [5].

3. *Mini-Batch Gradient Descent*

   Mini-batch gradient descent pada akhirnya mengambil keuntungan dari kedua variasi diatas yakni dengan melakukan pembaruan parameter dalam setiap *mini-batch* dari *n* contoh *training*:

   $$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{i:n}; y^{i:n}) \quad (2.18)$$

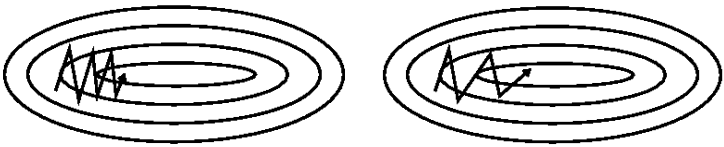
   Mini-batch gradient descent akan mengurangi noise saat pembaruan parameter dan menjadikan konvergen lebih stabil. Selain itu, waktu komputasi juga lebih cepat karena tidak memproses keseluruhan *dataset* dalam sekali pembaruan parameter.

2.5.5.2. Optimisasi Gradient Descent

Pada *gradient descent*, sering terjadi fungsi eror bersifat non-konveks sehingga menyebabkan algoritma optimisasi menhasilkan titik minimum lokal. Terdapat beberapa algoritma yang sering digunakan komunitas *deep learning* untuk menyelesaikan permasalahan tersebut.

1. Momentum

   Gradient Descent memiliki permasalahan pada saat melintasi jurang dimana area lengkungan sangatlah tajam dalam satu dimensi dibandingkan dengan yang lain. Dalam permasalahan ini, *gradient descent* akan berosilasi melalui *slope* jurang yang hanya akan membuat proses lambat untuk menuju optimum lokal seperti pada



Gambar 2.16 *Gradient descent* tanpa momentum (kiri) dan dengan momentum (kanan) [15]

Momentum merupakan sebuah metode yang membantu mempercepat *gradient descent* seperti pada Gambar 2.16 (kanan). Momentum bekerja dengan menambahkan sedikit γ vektor pembaruan dari langkah pada waktu lampau ke vektor pembaruan sekarang:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t \tag{2.19}$$

Dengan menambahkan γ , *gradient descent* akan lebih cepat mengalami konvergen dan mengurangi osilasi.

2. Adagrad

Adagrad merupakan sebuah algoritma untuk optimisasi berbasiskan gradien yang bekerja dengan adaptasi *learning rate* pada parameter-parameter, dimana memberikan *learning rate* rendah pada parameter terkait fitur yang sering terjadi dan *learning rate* tinggi pada parameter terkait fitur yang jarang terjadi. Untuk menyederhanakan, gradien pada waktu t dinotasikan dengan g_t. Kemudian $g_{t,i}$ adalah turunan parsial dari fungsi objektif mengacu pada parameter θ_i pada waktu t:

$$g_{t,i} = \nabla_\theta J(\theta_{t,i}) \tag{2.20}$$

Adagrad memodifikasi *learning rate* η umum tiap waktu t untuk tiap parameter θ_i berdasarkan gradiennya pada masa lampau:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i} \tag{2.21}$$

dimana $G_t \in \mathbb{R}^{d \times d}$ disini merupakan matriks diagonal dimana tiap elemen diagonal i, i merupakan penjumlahan dari kuadrat dari gradient mengacu pada θ_i pada waktu t , yang mana ϵ adalah paramater penghalus yang mencegah pembagian oleh nol.

Kelemahan utama Adagrad adalah akumulasi kuadrat gradien pada penyebutnya. Hal ini menyebabkan *learning rate* mengecil dan pada akhirnya akan bernilai sangat kecil sehingga algoritma tidak lagi dapat memperoleh tambahan pengetahuan.

3. Adadelta

Adadelta merupakan pengembangan dari Adagrad yang mengurangi penurunan monoton *learning rate*. Adadelta membatasi jumlah akumulasi gradien masa lampau pada sebuah ukuran tetap w.

Disamping menyimpan gradien masa lampau sebanyak w, Adadelta mendefinisikan jumlah dari gradien-gradien tersebut sebagai rata-rata dari keseluruhan gradien masa lampau. Rata-rata berjalan $E[g^2]_t$ pada waktu t yang bergantung hanya pada rata-rata sebelumnya dan gradien sekarang.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \quad (2.22)$$

Karena penyebut hanyalah kriteria akar kuadrat rata-rata eror (RMSE) dari gradien, Persamaan (2.22) dapat menjadi:

$$\theta_{t+1} = \theta_t - \frac{\eta}{RMS[g]_t} g_t \quad (2.23)$$

Pembaruan ini masih memiliki perbedaan satuan sehingga dilakukan penyamaan satuan pada parameter sehingga Persamaan (2.23) memiliki persamaan pembaruan akhir:

$$\theta_{t+1} = \theta_t - \frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t \quad (2.24)$$

dimana $\Delta\theta_t = \frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_{t,i}}$.

4. RMSprop

   RMSprop adalah metode optimisasi *gradient descent* yang tidak dipublikasikan dengan *adaptive learning rate* yang diusulkan oleh Geoff Hinton pada *Lecture 6e* di kelas Coursera. RMSprop dan Adadelta keduanya dikembangkan secara independen diwaktu yang sama dengan tujuan menyelesaikan masalah penurunan *learning rates* pada Adagrad. Faktanya, RMSprop identik dengan vektor pembaruan pertama dari Adadelta pada Persamaan (2.22).

5. Adam

   Adaptive Moment Estimation (Adam) adalah metode lain yang menghitung *adaptive learning rates* untuk tiap parameter. Dibandingkan dengan menyimpan rata-rata kuadrat gradien waktu lampau seperti Adadelta dan RMSprop, Adam menyimpan rata-rata gradien lampau m_t, sama seperti Momentum. Dimana Momentum dianalogikan sebagai bola yang menuruni sebuah lengkungan, Adam menganalogikan sebagai sebuah bola berat dengan gesekan sehingga akan lebih mudah menuju titik datar minimum. Persamaan untuk rata-

rata m_t dan kuadrat rata-rata v_t gradien masa lampau adalah sebagai berikut:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \quad (2.25)$$

m_t dan v_t adalah estimasi dari momen pertama (rata-rata) dan momen kedua (varian) dari gradien secara berurutan. Karena m_t dan v_t diinisialisasi sebagai vektor dari nol, m_t dan v_t cenderung menuju nilai nol, terutama ketika *decay rates* bernilai kecil (β_1 dan β_2 mendekati 1). Untuk mengatasinya, dihitung momen pertama dan kedua yang terkoreksi sebagai berikut:

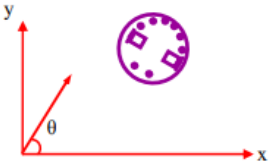
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \quad (2.26)$$

Kemudian, Persamaan (2.26) digunakan untuk pembaruan parameter seperti pada Adadelta dan RMSprop:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \quad (2.27)$$

2.6. Mobile Robot

Mobile robot adalah sebuah sistem robot yang memiliki karakteristik fungsionalitas berupa kemampuan bergerak, kemampuan merasakan dan beraksi, serta kemampuan berkomunikasi terhadap keadaan sekitar. *Mobile robot* dalam permukaan dua dimensi memiliki tiga derajat kebebasan (x, y, θ), dimana (x, y) menyatakan posisi dan θ menyatakan *heading*. Normalnya, ketiga variabel tersebut dinyatakan dalam sistem koordinat global.

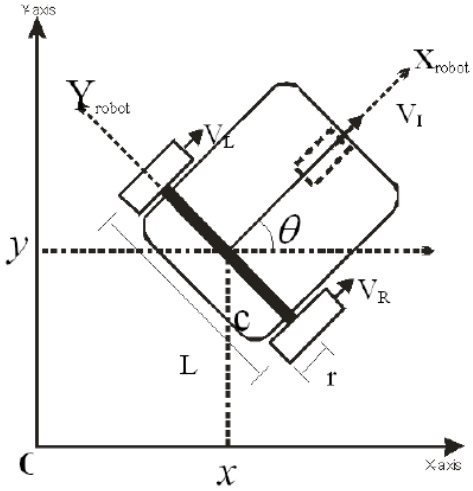


Gambar 2.17 Sistem Koordinat Global Robot $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$ [6]

2.6.1. Robot Locomotion

Berdasarkan cara bergeraknya, robot dapat diklasifikasikan atas robot *Holonomic* dan *Non-Holonomic*. *Holonomic* mengacu pada

hubungan antara derajat kebebasan yang dapat dikontrol dan jumlah derajat kebebasan dari sebuah robot. Jika derajat kebebasan yang dapat dikontrol sama dengan jumlah derajat kebebasan robot, maka dapat dikatakan sebagai robot berjenis *Holonomic.* Sedangkan apabila derajat kebebasan yang dapat dikontrol kurang dari jumlah derajat kebebasan robot, maka dapat diklasifikasikan sebagai robot *Non-Holonomic*.



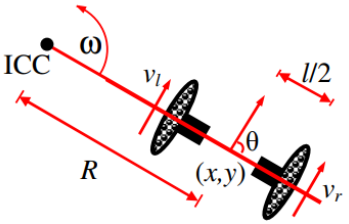
Gambar 2.18 Contoh Skema Robot Jenis *Non-Holonomic Drive* [2]

2.6.2. Differential Drive Kinematics

Kebanyakan *mobile robot* menggunakan mekanisme *differential drive* dimana terdiri dari dua roda yang dapat diatur putarannya secara independen. Untuk *differential drive mobile robot*, gerakan robot dikontrol melalui kecepatan masing-masing motor secara terpisah. Dari pengaturan kecepatan masing-masing motor, akan dapat diketahui posisi dan *heading* relatif melalui persamaan kinematika maju (*forward kinematics equation*).

2.6.2.1. Forward Kinematics Equations

Konsep pusat dari persamaan kinematika adalah kecepatan sudut ω robot. Dalam menentukan ω diperlukan suatu titik yang menjadi pusat rotasi dari robot yang biasa disebut *Instantaneous Center of Curvature* (ICC).



Gambar 2.19 Rotasi Robot Mengelilingi Suatu Titik ICC [6]

Didapatkan penyelesaian untuk ω dan R

$$R = \frac{0.5l(v_l + v_r)}{v_r - v_l} \tag{2.28}$$

$$\omega = \frac{v_r - v_l}{l} \tag{2.29}$$

Dengan diketahuinya ω, maka *heading* dapat diselesaikan dengan:

$$\theta_k = \theta_{k-1} + \omega\delta t \tag{2.30}$$

Dimana pusat rotasi ICC dapat didapatkan melalui trigonometri dasar sebagai berikut:

$$ICC = \begin{bmatrix} ICC_x \\ ICC_y \end{bmatrix} = \begin{bmatrix} x - R\sin\theta \\ y + R\sin\theta \end{bmatrix} \tag{2.31}$$

Apabila diketahui posisi awal (x_{k-1}, y_{k-1}) maka posisi baru (x_k, y_k) dapat dihitung dengan matriks rotasi dengan rotasi mengelilingi ICC:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \cos\omega\delta t & -\sin\omega\delta t \\ \sin\omega\delta t & \cos\omega\delta t \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \end{bmatrix} \tag{2.32}$$

Oleh karena itu, posisi baru (x_k, y_k, θ_k) dapat dihitung melalui persamaan (2.30) dan (2.32).

Kecepatan roda dapat diukur menggunakan sensor yang biasa disebut *wheel encoders*. Sensor ini biasanya dikopling dengan motor dan mengirim sinyal biner untuk setiap langkah dari perputaran roda. Sehingga kecepatan dapat dihitung dengan:

$$v = \frac{n_{step}}{\delta t} \tag{2.33}$$

Dengan mensubtitusikan persamaan (2.33) ke persamaan (2.28) dan (2.29), akan didapat:

$$R = \frac{0.5l(v_l + v_r)}{v_r - v_l} = \frac{0.5l(n_l + n_r)}{n_r - n_l} \tag{2.34}$$

$$\omega\delta t = \frac{v_r - v_l}{l}\delta t = \frac{(n_r - n_l)_{step}}{l} \quad (2.35)$$

Sehingga persamaan (2.30) dan (2.32) dapat dijadikan:

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} \cos\omega\delta t & -\sin\omega\delta t & 0 \\ \sin\omega\delta t & \cos\omega\delta t & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} - ICC_x \\ y_{k-1} - ICC_y \\ 0 \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{bmatrix} \quad (2.36)$$

2.6.2.2. Inverse Kinematics Equations

Dengan *forward kinematics equations*, posisi baru dapat diketahui berdasarkan perhitungan putaran motor. *Inverse kinematics* membalikkan permasalahan tersebut dimana menentukan v_l dan v_r berdasarkan posisi baru. Permasalahan ini sering tidak memiliki solusi karena robot tidak dapat mencapai suatu posisi dengan mengatur kecepatan v_l dan v_r.

Dengan mensubtitusikan beberapa kasus berikut ini ke (2.31)-(2.36), maka:

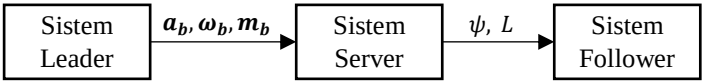
1. $v_r = v_l \rightarrow n_r = n_l \rightarrow R = \infty, \omega\delta t = 0$: Robot bergerak lurus dan θ tetap konstan
2. $v_r = -v_l \rightarrow n_r = -n_l \rightarrow R = 0, \omega\delta t = \frac{2n_{l_{step}}}{l}m, [ICC_x, ICC_y] = [x, y] \rightarrow x_k = x_{k-1}, y_k = y_{k-1}, \theta_k = \theta_{k-1} + \omega\delta t$: Robot berputar ditempat

Dengan mensubtitusikan kedua operasi diatas, algoritma berikut akan dapat digunakan untuk mencapai segala posisi target dari segala posisi awal:

1. Putar robot sampai orientasi robot sejajar dengan garis dari posisi awal sampai posisi target:
$$v_r = -v_l = v_{rot}$$
2. Jalan lurus sampai posisi robot sama dengan posisi target:
$$v_r = v_l = v_{ahead}$$

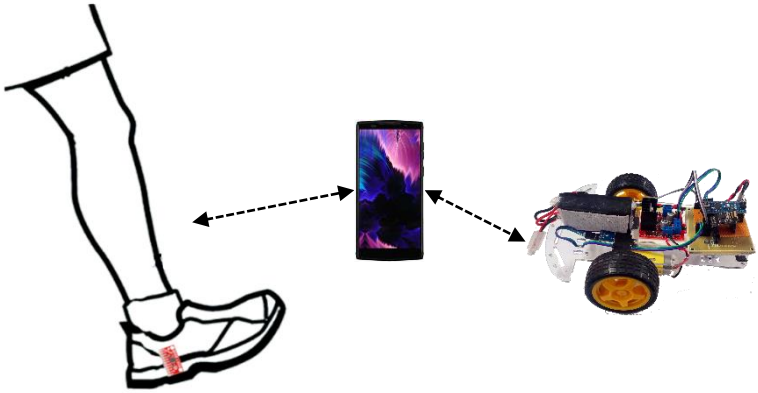
BAB 3
PERANCANGAN SISTEM

Perancangan sistem pada bab ini terdiri dari sistem *leader-follower* yang dipisah menjadi tiga subsistem yakni sistem *leader*, sistem *server*, dan sistem *follower*.



Gambar 3.1 Diagram Blok Sistem Keseluruhan

Sistem *leader* akan mengukur data percepatan, kecepatan sudut, dan kuat medan magnet dari sensor IMU 9-DoF yang kemudian dikirim melalui wi-fi ke sistem *server*. Sistem *server* akan mengestimasi panjang langkah dan arahnya berdasarkan data-data tersebut dan mengirim hasilnya ke sistem *follower*. Sistem *follower* kemudian akan bergerak sesuai dengan data hasil estimasi *server*. Selain itu, bab ini juga membahasan rancangan pengujian dari masing-masing subsistem.

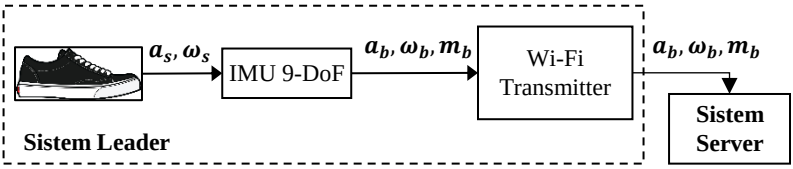


Gambar 3.2 Gambaran Umum Hubungan antar Subsistem

3.1. Sistem Leader

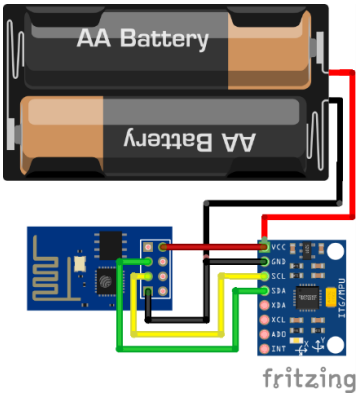
Sistem *leader* merupakan integrasi dari tiga komponen berupa IMU 9-DoF, modul wi-fi, dan baterai yang diletakkan pada sepatu bagian

kanan. IMU 9-DoF bertugas untuk mengukur pergerakan kaki berupa percepatan linear, kecepatan sudut, dan juga kuat magnet. Data dari IMU 9-DoF tersebut kemudian dikirim ke modul wi-fi melalui protokol komunikasi I2C. Frekuensi pengambilan data IMU 9-DoF diatur sebesar 25 Hz. Hal ini dikarenakan data IMU 9-DoF dikirim secara *realtime* dan *wireless* ke *server*.



Gambar 3.3 Diagram Blok Sistem *Leader*

Gambar 3.4 Komponen *Leader* di Sepatu Sebelah Kanan



Gambar 3.5 Konfigurasi Komponen Sistem *Leader*

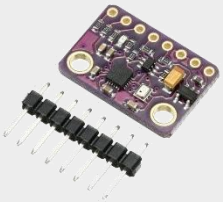
Data dari IMU 9-DoF tersebut kemudian dikirim ke modul wi-fi melalui protokol komunikasi I2C. Frekuensi pengambilan data IMU 9-DoF diatur 25 Hz. Hal ini dikarenakan data IMU 9-DoF dikirim secara *realtime* dan *wireless* ke *server* serta pemrosesan data di*server* bersifat *synchronous.*

Proses keseluruhan sistem *leader* dijelaskan pada Gambar 3.3 dimana $\boldsymbol{a_s, \omega_s}$ adalah percepatan linear dan kecepatan sudut yang dialami oleh sepatu *leader* secara berurutan. Kemudian $\boldsymbol{a_b, \omega_b, m_b}$ merupakan percepatan linear, kecepatan sudut, dan medan magnet yang terukur oleh IMU 9-DoF pada sepatu *leader* secara berurutan.

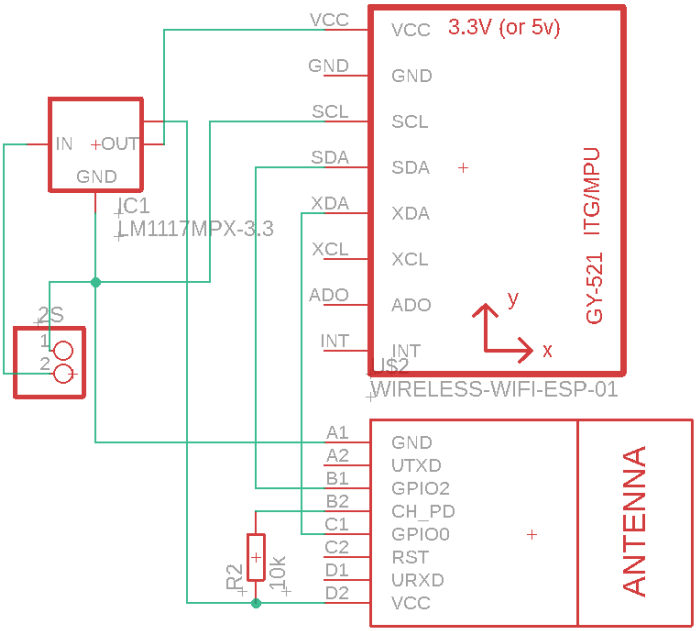
3.1.1. Komponen

Sistem *leader* memiliki tiga komponen utama yang dirincikan pada Tabel 3.1 dibawah ini.

Tabel 3.1 Komponen Sistem *Leader*

No	Nama	Gambar
1	IMU 9-DoF GY-91	
2	Modul Wi-fi ESP8266-01	
3	Baterai Lipo	
4	PCB	Gambar 3.8

3.1.2. Perancangan PCB



Gambar 3.6 Mode *Schematic* PCB Sistem *Leader*

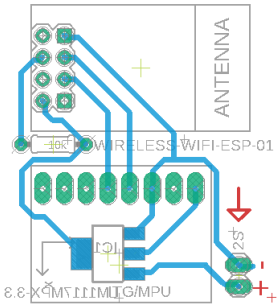
Dirancang sebuah *Printed Circuit Board* (PCB) untuk sistem *leader* sesuai dengan konfigurasi komponen sistem pada Gambar 3.5 menggunakan software Autodesk Eagle. Perancangan diawali dengan menghubungkan beberapa pin tiap komponen diantaranya:

- Pin VCC pada IMU 9-DoF dan ESP8266 masuk pada regulator tegangan baterai 3.3 volt.
- Semua pin GND terhubung menjadi satu *node*.
- Pin SCL pada IMU 9-DoF masuk pin GPIO2 pada ESP8266.
- Pin SDA pada IMU 9-DoF masuk pin GPIO0 pada ESP8266.
- Pin masukan regulator tegangan terhubung dengan pin positif (+) baterai.
- Pin CH-PD pada ESP8266 masuk pin VCC dengan resistor sebagai penghubung.

Koneksi keseluruhan komponen digambarkan pada Gambar 3.6.

Koneksi IMU 9-DoF tidak sesuai dengan keterangan diatas karena IMU 9-DoF yang digunakan bertipe GY-91, sedangkan pada *library* Eagle tidak memiliki komponen IMU 9-DoF bertipe GY-91, sehingga digunakan IMU 6-DoF tipe GY-521 dengan penyesuaian pin sebagai berikut:

- VCC → VCC
- SCL → GND
- SDA → SCL
- XDA → SDA



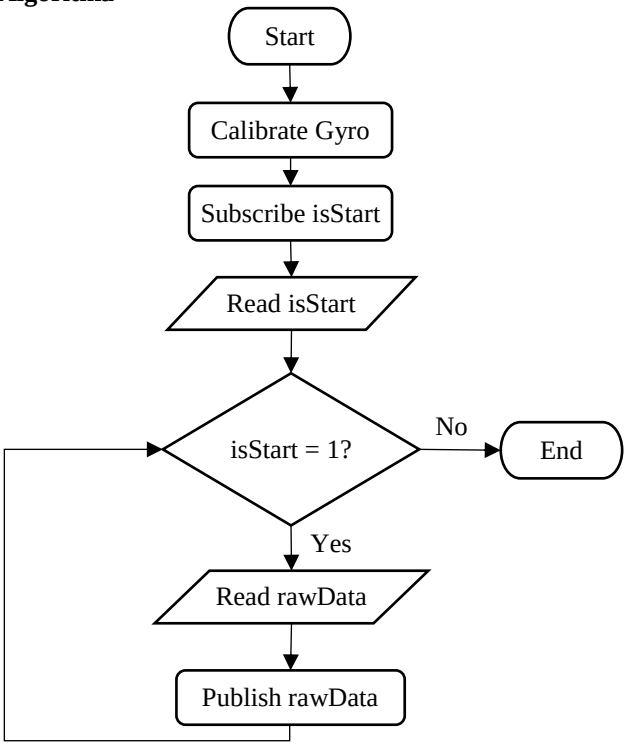
Gambar 3.7 Mode *Board* PCB Sistem *Leader*

Setelah menghubungkan seluruh pin, perancangan PCB dilanjutkan dengan mengubah mode menjadi mode *board*. Pada mode *board*, penyusunan komponen dilakukan dengan menghubungkan pin dimana mode inilah nanti yang akan tercetak pada PCB. Perancangan *board* PCB dapat dilihat pada Gambar 3.7.

Gambar 3.8 Hasil Cetak PCB

Setelah perancangan *board* selesai, dilakukan cetak PCB dengan memanfaatkan jasa Panut PCB di Surabaya. PCB hasil cetak dapat dilihat pada Gambar 3.8. Setelah PCB dicetak, seluruh komponen disolder sesuai dengan desainnya. Setelah selesai, PCB dipasang di sepatu sebelah kanan seperti pada Gambar 3.4.

3.1.3. Algoritma

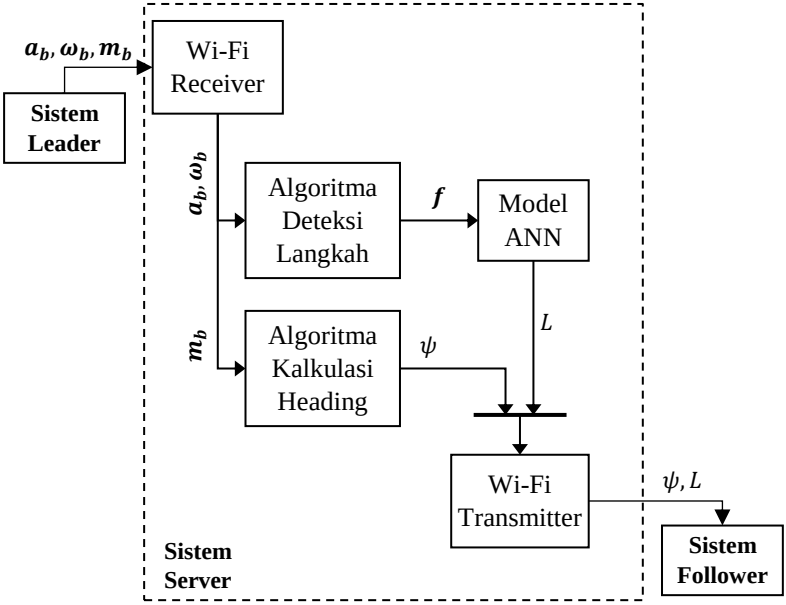


Gambar 3.9 *Flowchart* Sistem *Leader*

Algoritma sistem *leader* dijelaskan melalui *flowchart* pada Gambar 3.9. Akan tetapi sebelum menjalankan algoritma tersebut, magnetometer perlu dikoreksi untuk distorsi *hard* dan *soft iron* seperti pada Persamaan (2.6). Sistem dimulai dengan menyalakan komponen (*Start*) dan mendiamkan beberapa saat untuk kalibrasi giroskop (*Calibrate Gyro*). Kemudian menunggu instruksi dari *server* untuk menyalakan pengukuran

(isStart) dimana jika instruksi diterima (isStart=1) maka sistem akan memulai mengukur data sensor (*read rawData*) dan mengirimkannya ke *server* (*publish rawData*) secara terus-menerus sampai instruksi mematikan diterima (isStart=0) dan algoritma sistem *leader* berhenti (End).

3.2. Sistem *Server*



Gambar 3.10 Diagram Blok Sistem *Server*

Sistem *server* merupakan sebuah komputer berupa smartphone yang akan menerima data IMU 9-DoF dari *leader* dan mengestimasi panjang langkah dan arahnya yang kemudian mengirim hasil estimasi ke *follower*. Proses keseluruhan sistem *server* dijelaskan pada Gambar 3.10 dengan $\boldsymbol{f}$ merupakan fitur ANN yang terdiri dari percepatan maksimum, percepatan minimum, percepatan rata-rata, standar deviasi percepatan, kecepatan sudut maksimum, kecepatan sudut minimum, kecepatan sudut rata-rata, dan standar deviasi kecepatan sudut dari masing-masing sumbu akselerometer dan giroskop. Selain itu terdapat dua fitur yang lain yakni

jumlah data dalam satu langkah (frekuensi langkah) dan temperatur. *L* didefinisikan sebagai panjang langkah dan ψ sebagai *heading*.

3.2.1. Komponen

Sistem *server* memiliki komponen utama berupa smartphone yang dalam hal ini digunakan smartphone android bertipe Doogee BL9000.

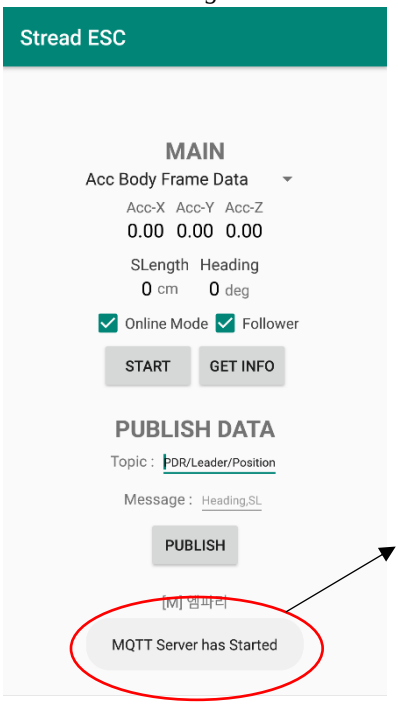
Gambar 3.11 Doogee BL9000

3.2.2. Perancangan Aplikasi

Dikembangkan sebuah aplikasi android untuk menerima, mengirim, dan mengolah data dari *leader* yang kemudian dikirim ke *follower*. Selain itu, aplikasi juga berfungsi membuka *port* untuk *server* MQTT. Aplikasi dikembangkan dengan menggunakan software Android Studio dengan bahasa pemrograman Kotlin. Aplikasi diberi nama "Stread ESC" yang merupakan akronim dari "*Stread Estimation and Controller*".

Aplikasi dapat digunakan setelah *server* MQTT aktif. Untuk mengaktifkan *server*, *wireless access point* (hotspot) android perlu diaktifkan. Kemudian saat aplikasi dibuka, aplikasi akan otomatis mengaktifkan *server* MQTT. *Server* aktif ditandai dengan munculnya *toast* bertuliskan "MQTT *Server* has Started" seperti pada Gambar 3.12. Setelah *server* MQTT aktif, aplikasi dapat memberi perintah ke *leader* untuk memulai pengambilan data melalui tombol "START". Apabila perintah sukses dikirim ke *leader*, tombol "START" akan berganti menjadi tombol "STOP" yang berfungsi untuk menghentikan pengambilan data. Tombol "Get Info" berfungsi untuk mendapatkan informasi *follower* berupa informasi *heading*.

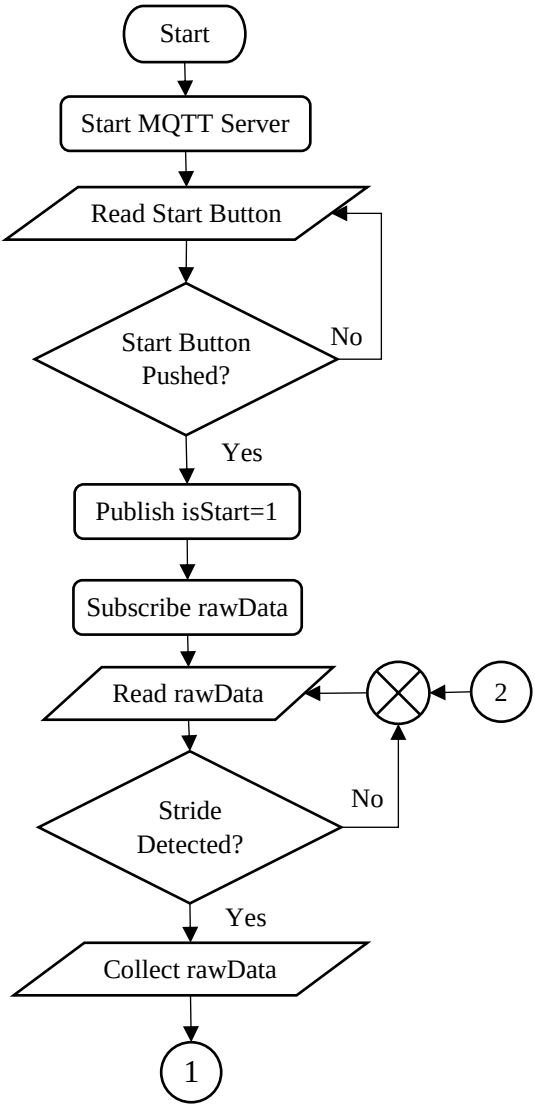
Aplikasi dapat menampilkan data berupa data *raw* secara *realtime* dari pengukuran *leader* serta hasil estimasi panjang langkah dan *heading* tiap langkah. Aplikasi memiliki dua mode, *online* dan *offline*. Mode *online* digunakan saat aplikasi sudah memiliki data bobot ANN sehingga dapat mengestimasi panjang langkah. Sedangkan mode *offline* digunakan untuk pengambilan data untuk *learning* model ANN.



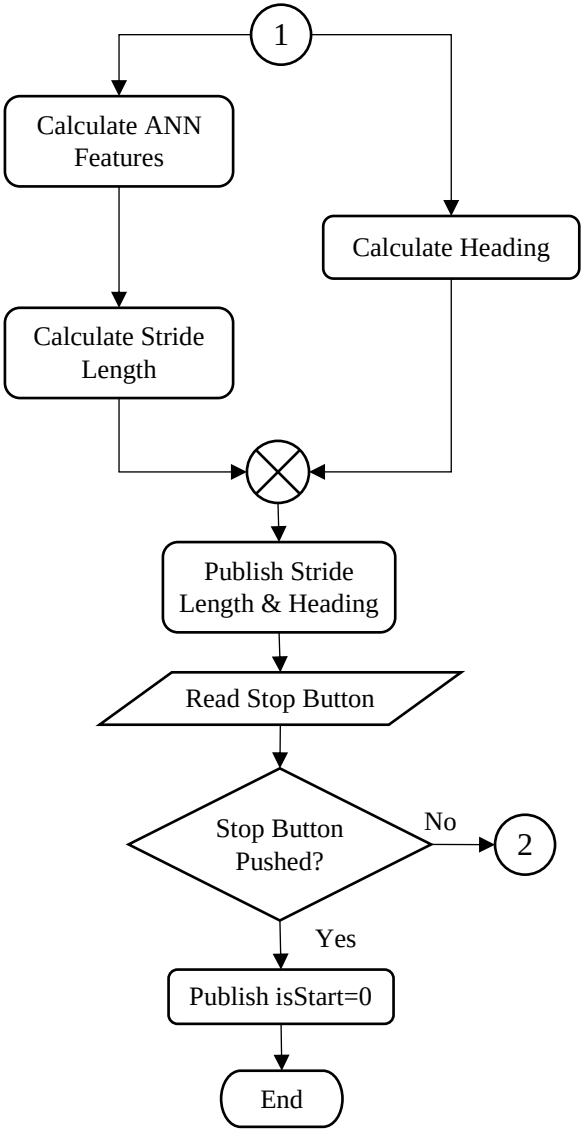
Gambar 3.12 *Server* MQTT telah Aktif

3.2.3. Algoritma

Algoritma sistem *server* dibagi menjadi beberapa bagian diantaranya deteksi langkah berdasarkan data akselerometer dan giroskop, estimasi panjang langkah menggunakan ANN, estimasi heading menggunakan data magnetometer yang telah dikompensasi kemiringannya.



Gambar 3.13 *Flowchart* Sistem *Server* (Bagian 1)

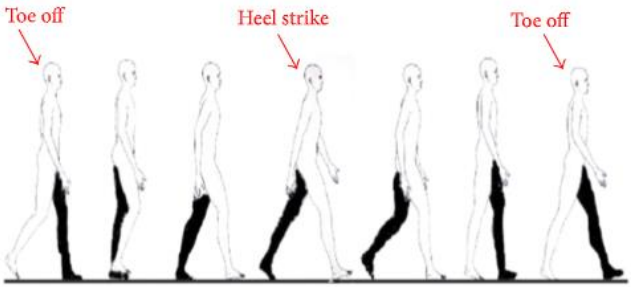


Gambar 3.14 *Flowchart* Sistem *Server* (Bagian 2)

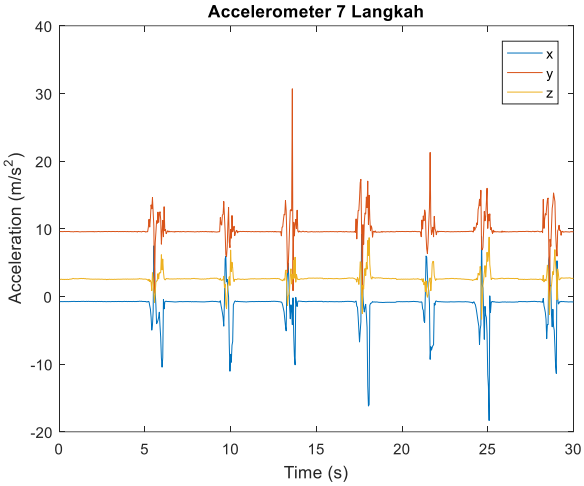
Algoritma dimulai ketika aplikasi dibuka (Start). Saat aplikasi dibuka, aplikasi akan membuka *server* MQTT (Start MQTT *Server*). Setelah *server* berhasil dibuka, aplikasi menunggu instruksi melalui tombol START (Read Start Button). Ketika tombol START ditekan (Start Button Pushed), aplikasi akan mengirim instruksi berupa isStart=1 (Publish isStart=1) serta mendaftarkan topik rawData (Subscribe rawData). rawData akan diterima dari IMU 9-DoF *leader* dan dilakukan deteksi langkah secara terus-menerus (Read rawData). Ketika langkah terdeteksi (Stride Detected), aplikasi akan mulai menyimpan rawData dan mencari nilai maksimum, minimum, rata-rata, dan standar deviasi dalam satu langkah yang kemudian dijadikan sebagai masukan ANN (Calculate ANN Features). Setelah itu, menggunakan model yang telah di *learning* sebelumnya, dapat dihitung panjang langkahnya (Calculate Stride Length). Selain itu, aplikasi juga akan menghitung arah langkah berdasarkan rawData magnetometer (Calculate Heading). Setelah didapatkan panjang dan arah langkah, kedua nilai tersebut dikirim ke *server* (Publish Stride Length & Heading). Setelah itu, aplikasi akan mengulangi instruksi diatas terus-menerus dan menunggu tombol Stop ditekan (Read Stop Button). Ketika tombol Stop ditekan, aplikasi akan mengirim isStart=0 ke *server* (Publish isStart=0) sehingga algoritma sistem *server* berakhir (End).

3.2.3.1. Stride Detection

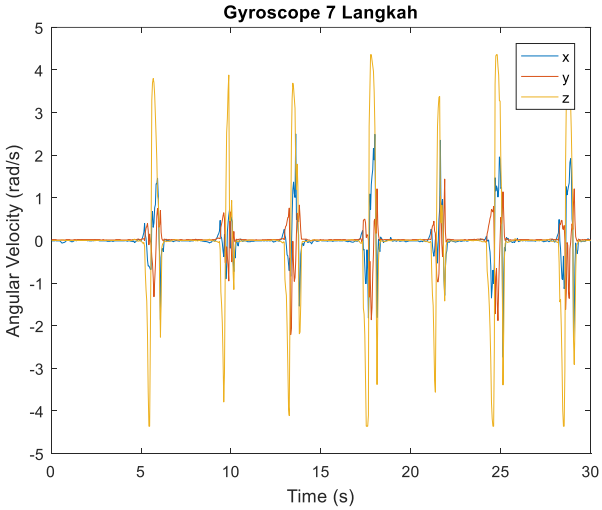
Dalam satu siklus berjalan terdapat dua fase yakni saat kaki akan diangkat (*heel strike*) dan saat kaki selesai diturunkan (*toe off*) sesuai Gambar 3.15.



Gambar 3.15 Fase Satu Siklus Berjalan [23]



Gambar 3.16 Data Akselerometer dalam Tujuh Langkah



Gambar 3.17 Data Giroskop dalam Tujuh Langkah

Saat berjalan, data akselerometer dan giroskop menunjukkan karakteristik siklus yang berulang. Dari siklus berulang ini dapat disimpulkan bahwa setiap langkah kaki, akselerometer dan giroskop akan

membangkitkan suatu sinyal tiap langkah. Dari sinyal tersebut kemudian dapat dijadikan sebagai fitur untuk deteksi dari terangkatnya kaki sampai turun kembali.

Dari Gambar 3.16 dan Gambar 3.17 dapat dibuat sebuah algoritma deteksi langkah yang dibagi menjadi empat fase. Fase satu didefinisikan saat kaki akan terangkat dimana sinyal giroskop sumbu-z akan menuju nilai negatif. Kemudian fase dua terjadi ketika data akselerometer sumbu-x bernilai maksimum. Fase tiga terjadi ketika data akselerometer sumbu-x bernilai minimum. Terakhir fase empat dimana kaki menyentuh tanah yang terjadi ketika sinyal giroskop sumbu-z kembali ke nilai nol.

3.2.3.2. Artificial Neural Network

Untuk estimasi panjang langkah, digunakan metode ANN dengan total 26 variabel masukan yang diperkirakan mempengaruhi panjang langkah yakni berupa percepatan maksimum $\boldsymbol{a_{max}}$, percepatan minimum $\boldsymbol{a_{min}}$, percepatan rata-rata $\boldsymbol{a_{mean}}$, standar deviasi percepatan $\boldsymbol{a_{std}}$, kecepatan sudut maksimum $\boldsymbol{\omega_{max}}$, kecepatan sudut minimum $\boldsymbol{\omega_{min}}$, kecepatan sudut rata-rata $\boldsymbol{\omega_{mean}}$, standar deviasi kecepatan sudut $\boldsymbol{\omega_{std}}$, jumlah data dalam satu langkah (frekuensi langkah) f_{stride}, dan temperatur t.

3.2.3.3. Tilt Compensated Heading

Untuk estimasi arah langkah digunakan sensor magnetometer, namun karena pada implementasinya perangkat tidak diposisikan sesuai sumbunya maka diperlukan kompensasi kemiringan (*tilt compensation*) pada magnetometer. Transformasi frame memerlukan *roll* dan *pitch* dari perangkat. Estimasi *roll* dan *pitch* diperoleh dari gabungan data akselerometer dan giroskop dengan menggunakan *complementary filter* (CF).

$$\begin{aligned} \hat{r} &= K * \left(\hat{r} + r_{gyro}\right) + (1-k) * r_{acc} \\ \hat{p} &= K * \left(\hat{p} + p_{gyro}\right) + (1-k) * p_{acc} \end{aligned} \tag{3.1}$$

dimana:

- $\hat{r}$ dan $\hat{p}$ adalah estimasi *roll* dan *pitch* CF secara berurutan
- K adalah gain CF
- r_{gyro} dan p_{gyro} adalah *roll* dan *pitch* dari giroskop yang diperoleh dari:

$$r_{gyro} = \omega_x * \Delta t \tag{3.2}$$

$$p_{gyro} = \omega_y * \Delta t$$

- r_{acc} dan p_{acc} adalah *roll* dan *pitch* dari akselerometer yang diperoleh dari:

$$r_{acc} = atan2(a_y, a_z)$$
$$p_{acc} = atan2\left(-a_x, \sqrt{a_y^2 + a_z^2}\right) \quad (3.3)$$

Setelah didapat *roll* dan *pitch* hasil CF, dilakukan kompensasi kemiringan dengan formula sebagai berikut:

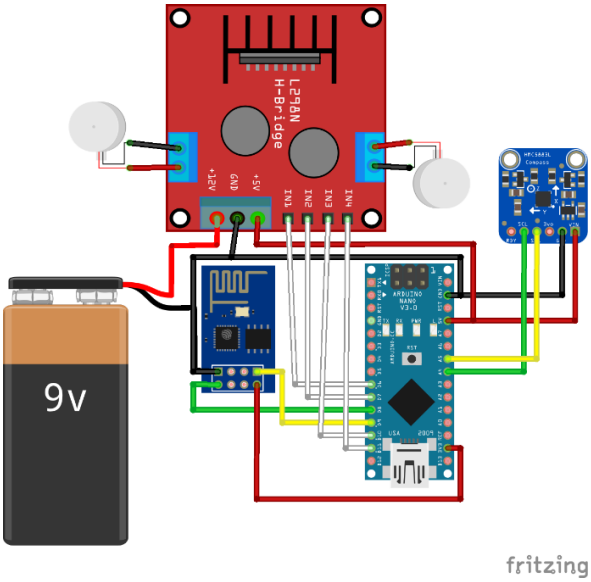
$$y = atan2(-M_y, M_x) \quad (3.4)$$

dimana:

$$M_x = m_x * \cos\hat{p} + m_z * \sin\hat{p}$$
$$M_y = m_x * \sin\hat{r} * \sin\hat{p} + m_y * \cos\hat{r} - m_z * \sin\hat{r} * \cos\hat{p} \quad (3.5)$$

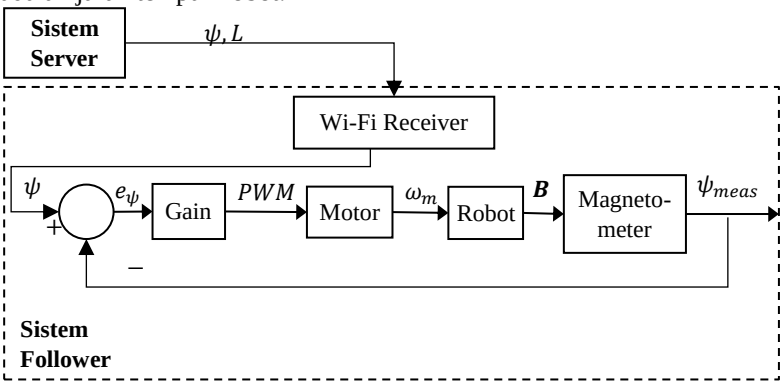
dengan m_x, m_y, m_z merupakan kuat medan magnet yg diukur magnetometer pada sumbu x,y, dan z secara berurutan.

3.3. Sistem Follower

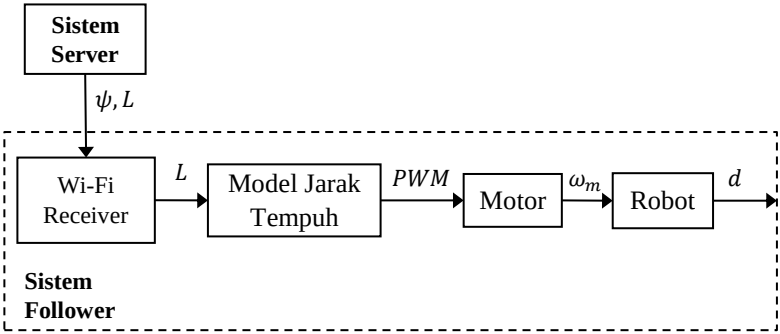


Gambar 3.18 Konfigurasi Komponen Sistem *Follower*

Sistem *follower* terdiri dari beberapa integrasi komponen diantaranya Arduino, modul wi-fi, magnetometer, driver motor, serta baterai. Konfigurasi komponen dapat dilihat di Gambar 3.18. Terdapat dua diagram blok pada sistem *follower* yaitu diagram blok pengaturan *heading* pada Gambar 3.19 dan pengaturan jarak tempuh robot pada Gambar 3.20 dimana ω_m adalah kecepatan putar motor, $\boldsymbol{B}$ adalah kerapatan fluks magnetik disekitar robot, *en* adalah saklar *enable*, dan d adalah jarak tempuh robot.



Gambar 3.19 Diagram Blok Sistem *Follower* (*Heading*)

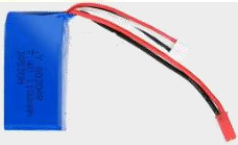
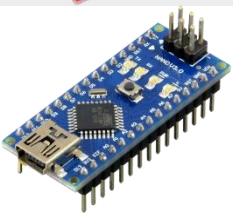


Gambar 3.20 Diagram Blok Sistem *Follower* (Gerakan Maju)

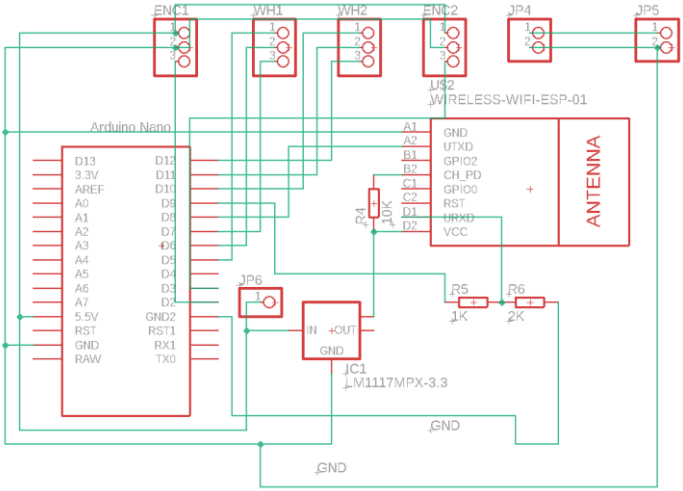
3.3.1. Komponen

Sistem *follower* memiliki komponen utama yang dirincikan pada Tabel 3.2 dibawah ini.

Tabel 3.2 Komponen Sistem *Follower*

No	Nama	Gambar
1	Magnetometer HMC5883L	
2	Kit 2 WD Robot Car	
3	Baterai Lipo	
4	Modul Wi-fi ESP8266-01	
5	Driver Motor L298N	
6	Arduino Nano	

3.3.2. Perancangan PCB

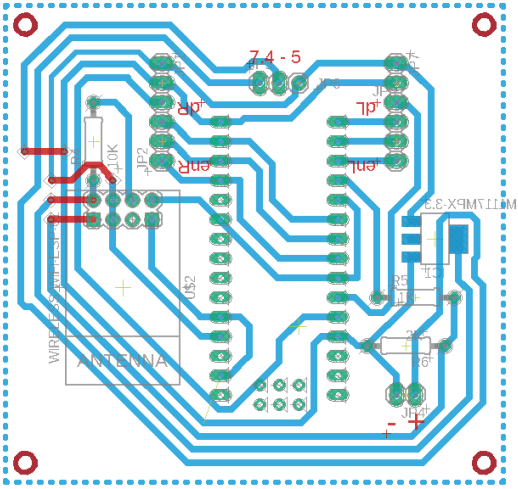


Gambar 3.21 Mode *Schematic* PCB Sistem *Follower*

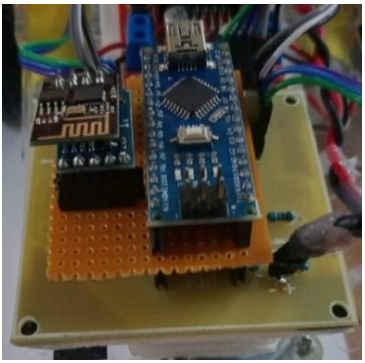
Dirancang sebuah *Printed Circuit Board* (PCB) untuk sistem *follower* sesuai dengan konfigurasi komponen sistem pada Gambar 3.18 menggunakan software Autodesk Eagle. Perancangan diawali dengan menghubungkan beberapa pin tiap komponen diantaranya:

- Pin 5V pada L298N dihubungkan pada pin 5v Arduino Nano, dan juga masuk sebagai masukan regulator 3.3 volt.
- Semua pin GND terhubung menjadi satu *node*.
- Pin WH1 masuk pada pin D5-D7 Arduino secara berurutan.
- Pin WH2 masuk pada pin D10-D12 Arduino secara berurutan.
- Pin keluaran regulator dihubungkan pada pin VCC ESP8266.
- Pin CH-PD pada ESP8266 masuk pin VCC dengan resistor sebagai penghubung.
- Pin URXD ESP8266 masuk pada pin D9 Arduino dengan pembagi tegangan sebagai penghubungnya.
- Pin UTXD ESP8266 masuk pada pin D8 Arduino.
- Pin SDA HMC5883L masuk pada pin A4 Arduino.
- Pin SCL HMC5883L masuk pada pin A5 Arduino.

Komponen HCM5883L berada pada PCB terpisah sehingga tidak terdapat pada Gambar 3.21.



Gambar 3.22 Mode *Board* PCB Sistem *Leader*

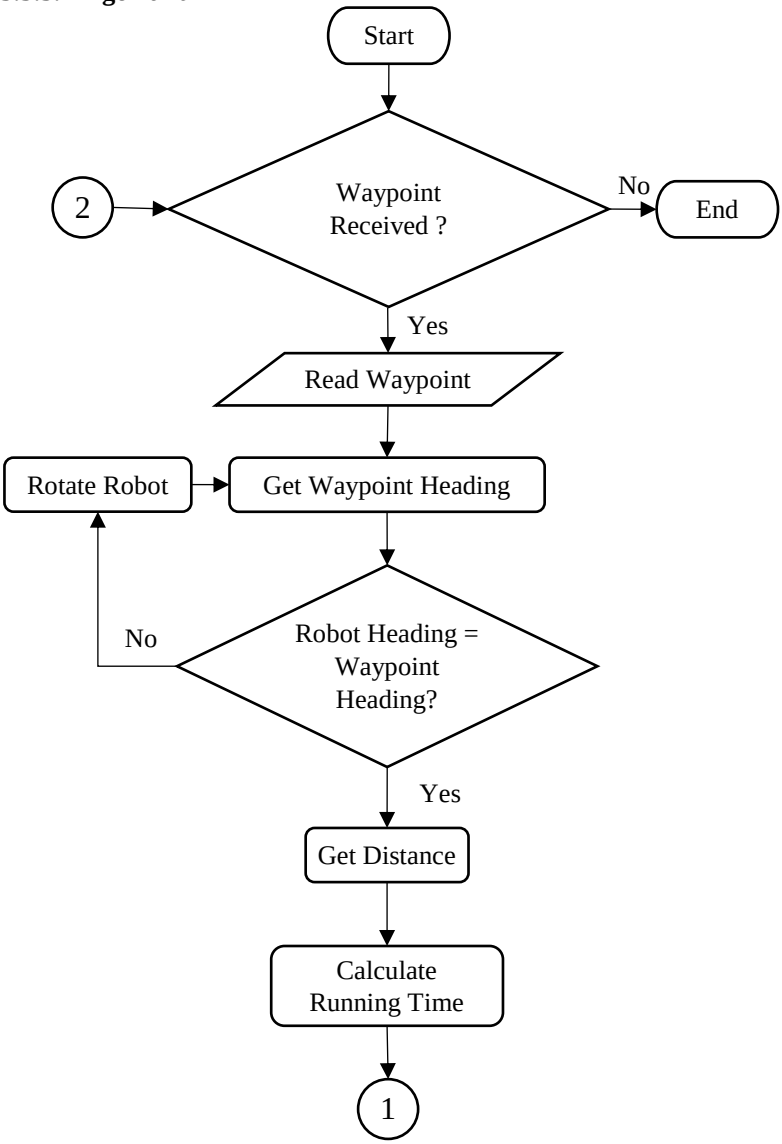


Gambar 3.23 Hasil Cetak PCB

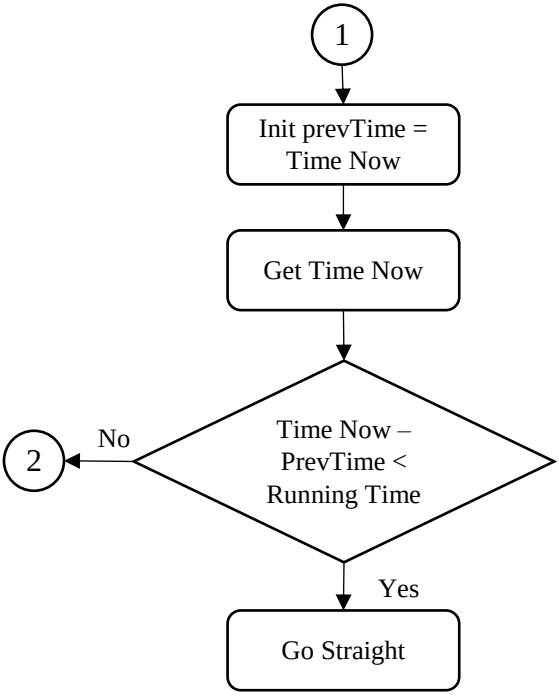
Setelah menghubungkan seluruh pin, perancangan PCB dilanjutkan dengan mengubah mode menjadi mode *board*. Pada mode *board*, penyusunan komponen dilakukan dengan menghubungkan pin dimana mode inilah nanti yang akan tercetak pada PCB. Perancangan *board* PCB dapat dilihat pada Gambar 3.22.

Setelah perancangan *board* selesai, dilakukan cetak PCB dengan memanfaatkan jasa Panut PCB di Surabaya. PCB hasil cetak dapat dilihat pada Gambar 3.23.

3.3.3. Algoritma



Gambar 3.24 *Flowchart* Sistem *Follower* (Bagian 1)

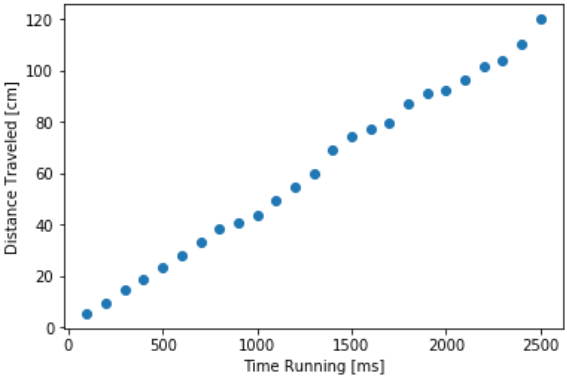


Gambar 3.25 *Flowchart* Sistem *Follower* (Bagian 2)

Sistem *follower* memiliki algoritma yang diringkas pada *flowchart* Gambar 3.24. Pertama, sistem dinyalakan (Start) dengan menghubungkan baterai ke robot. Kemudian robot menunggu instruksi berupa *waypoint* dari *server*, apabila terdapat *waypoint* yang dikirim maka waypoint akan diterima (Waypoint Received?, Yes) sistem akan melakukan pembacaan *waypoint* (Read Waypoint). Setelah *waypoint* dibaca, sistem akan mendapatkan *heading waypoint* (Get Waypoint Heading) dan apabila *heading* robot sama dengan *heading* target (Robot Heading = Heading Target?, Yes) maka robot akan mendapatkan jarak dari *waypoint* yang diterima (Get Distance), sedangkan apabila *heading* robot tidak sama dengan *heading* target, maka robot akan berputar (Rotate Robot) sampai *heading* robot sama dengan *heading* target. Setelah *heading* robot sama dengan *heading* target, robot mengkalkulasi waktu untuk melakukan gerakan maju (Calculate Running Time). Setelah diketahui waktu gerakan

maju, robot akan berjalan sesuai dengan waktu yang telah dikalkulasi (Go Straight). Setelah robot selesai bergerak, robot akan menunggu *waypoint* selanjutnya dan sistem akan selesai (End) ketika robot tidak menerima *waypoint*.

3.3.3.1. Model Jarak Tempuh



Gambar 3.26 Data Jarak Tempuh Berdasarkan Waktu Tempuh

Dari *waypoint* yang diterima *follower* akan didapatkan jarak tempuh. Untuk menempuh jarak tersebut, *follower* diharuskan mengetahui waktu tempuh untuk menggerakkan rodanya. Dilakukan permodelan jarak tempuh dengan masukan waktu bergerak sebagai model linear. Dilakukan pengambilan data sebanyak 25 data untuk memodelkan jarak tempuh.

Dari Gambar 3.26 didapatkan model jarak tempuh menggunakan regresi linear adalah sebagai berikut:

$$y = 0.047x + 0.313 \quad (3.6)$$

dengan x adalah waktu tempuh dan y adalah jarak tempuh.

3.4. Perancangan Pengujian

Berdasarkan [7] dan [23] dirancang spesifikasi desain untuk *heading*, panjang langkah, dan jarak tempuh robot seperti pada Tabel 3.3.

Tabel 3.3 Spesifikasi Desain Sistem

Variabel	Spesifikasi Desain
Heading	RMSE 2°
Panjang langkah	2% dari panjang lintasan pengujian
Jarak Tempuh	2% dari panjang lintasan pengujian

3.4.1. Pengujian Giroskop

Saat inisialisasi, dilakukan kalibrasi giroskop untuk menghilangkan *offset.* Kalibrasi dilakukan dengan memanggil fungsi calibrateGyro() pada Lampiran 1. Saat kalibrasi, giroskop didiamkan selama beberapa saat. Saat diam tersebut giroskop akan mengumpulkan data kemudian akan menjadikan data yang terkumpul tersebut sebagai bias sampai data yg diukur giroskop mendekati nol.

3.4.2. Pengujian Magnetometer

3.4.2.1. Pengujian Kalibrasi

Magnetometer diuji dengan memutar perangkat dalam sumbu-z untuk melihat eror setiap putaran. Dilakukan uji dengan memutar magnetometer dengan variasi putaran 0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°, *dan* 360°.

3.4.2.2. Pengujian Offset

Karena Magnetometer memiliki *offset* yang berbeda-beda di tiap tempat, Dilakukan uji dengan mengukur *offset* magnetometer pada 28 titik yang memiliki jarak tiap titik 1 meter.

3.4.3. Pengujian Algoritma Deteksi Langkah

Deteksi langkah merupakan algoritma pertama yang dijalankan *server* untuk mengaktifkan pengambilan data sebagai masukan ANN. Untuk dapat mendeteksi langkah, diperlukan tuning parameter *threshold* untuk sinyal akselerometer dan giroskop. Sinyal yang digunakan untuk deteksi langkah ada dua dengan asumsi paling berpengaruh dalam gerakan langkah yakni akselerometer sumbu x dan giroskop sumbu z. Dilakukan uji deteksi langkah dengan melangkah sebanyak 25 langkah dan menjalankan algoritma dari deteksi langkah. Program *stride detection* dilampirkan pada Lampiran 4.

3.4.4. Pengujian Model Panjang Langkah

Dilakukan pengambilan 614 sampel langkah beserta *label*-nya untuk *learning* dan *testing* Model ANN dan mencari model terbaik berdasarkan MSE yang terjadi. Sampel tersebut kemudian dibagi menjadi 90% sampel untuk *training* dan 10% sampel untuk *testing*. Dari RMSE didapatkan model ANN terbaik yang kemudian digunakan untuk estimasi panjang langkah.

Selain pengujian ANN, dilakukan juga uji metode estimator panjang langkah umum yang lain, diantaranya integral langsung data akselerometer seperti pada Persamaan (3.7), model non-linear, dan model frekuensi sesuai dengan Subbab 2.4.2. Pengujian ANN dilakukan dengan melakukan *learning* berdasarkan data yang telah diambil dimulai dengan satu *hidden layer* dan satu *neuron* yang kemudian terus ditambah sampai hasil RMSE *testing* membesar yang menandakan model terlalu *overfitting*.

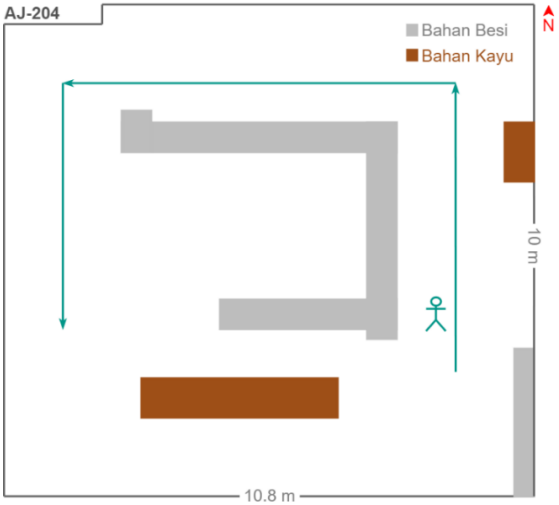
$$r(t) = r_0 + \int_0^t \int_0^t a(t)\, dt\, dt \tag{3.7}$$

3.4.5. Pengujian Model Jarak Tempuh Follower

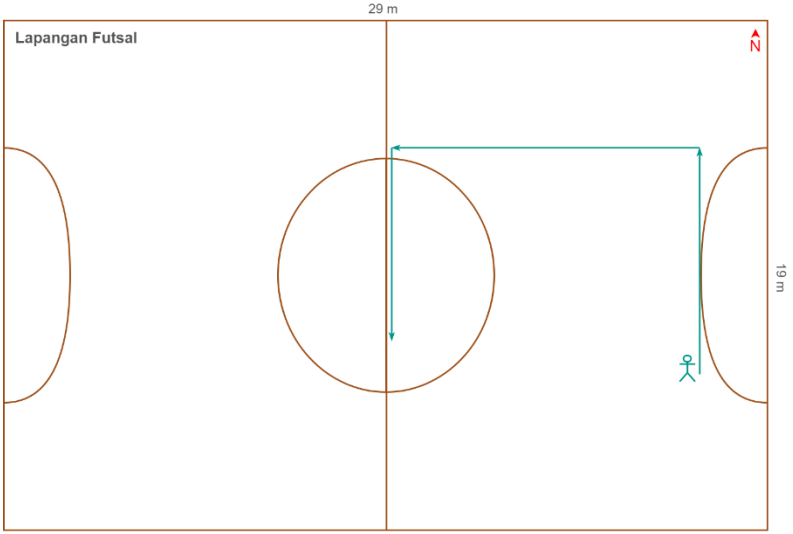
Follower memerlukan pengetahuan berapa lama untuk bergerak dalam suatu jarak tempuh tertentu. Oleh karena itu dilakukan pengujian model jarak tempuh sesuai dengan Persamaan (3.6) dengan memberikan masukan x 100, 200, sampai 2500.

3.4.6. Pengujian Sistem Navigasi Leader

Sistem navigasi *leader* diuji dengan cara pejalan kaki berjalan mengikuti lintasan yang telah ditentukan. Pengujian dilakukan dengan lintasan berbentuk U dengan panjang lintasan per sisi 6-8-5 meter seperti pada Gambar 3.27 dan Gambar 3.28.



Gambar 3.27 Denah Pengujian (*Indoor*)



Gambar 3.28 Denah Pengujian (*Outdoor*)

3.4.7. Pengujian Sistem Navigasi Follower

Sistem navigasi *follower* diuji dengan cara pengiriman data panjang langkah dan heading secara manual dari *server* ke *follower*. Lintasan untuk pengujian *follower* sama seperti lintasan pengujian *leader* pada Gambar 3.27 dan Gambar 3.28 dimana untuk pengujian panjang langkah dikirim tiap satu meter dengan *heading* tetap untuk lintasan lurus dan ditambah 90° pada waktu lintasan belok U.

3.4.8. Pengujian Sistem Navigasi Leader-Follower

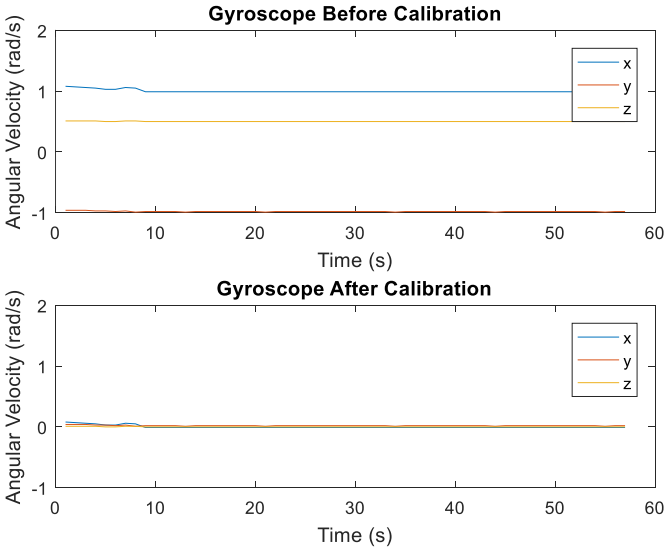
Sistem navigasi *leader-follower* diuji dari gabungan kedua pengujian independen dimana *leader* akan berjalan sesuai lintasan yang ditentukan kemudian *leader* akan mengirim hasil estimasi panjang langkah dan *heading* ke *follower*.

BAB 4
HASIL DAN ANALISA

Pada bab ini, hasil dan analisa dari perancangan pengujian bab 3 akan dibahas. Pengujian ini meliputi pengujian *giroskop*, *magnetometer*, deteksi langkah, model panjang langkah, model jarak tempuh *follower*, sistem navigasi *leader*, sistem navigasi *follower*, serta sistem keseluruhan.

4.1. Pengujian Giroskop

Pengujian giroskop dimaksudkan supaya pengukuran giroskop tereliminasi dari *offset/bias* karena sinyal giroskop akan digunakan sebagai masukan algoritma deteksi langkah.



Gambar 4.1 Giroskop Sebelum dan Sesudah Kalibrasi

Dari gambar diatas, dapat disimpulkan bahwa kalibrasi giroskop berhasil dilakukan dibuktikan dengan sangat kecilnya *offset* yang terjadi ketika giroskop dalam keadaan diam.

4.2. Pengujian Magnetometer

4.2.1. Pengujian Kalibrasi

Magnetometer diuji dengan memutar perangkat dalam sumbu-z untuk melihat eror setiap putaran. Dilakukan uji dengan memutar magnetometer dengan variasi putaran 0°, 45°, 90°, 135°, 180°, 225°, 270°, dan 315°.

Tabel 4.1 Magnetometer Sebelum dan Setelah Kalibrasi

Referensi (°)	**Sebelum Kalibrasi** (°)	**Setelah Kalibrasi** (°)
0	33	0
45	47	50
90	61	93
135	69	132
180	68	175
225	48	218
270	19	257
315	23	306

Dari tabel diatas, didapatkan RMSE untuk magnetometer sebelum kalibrasi sebesar 91.288° dan setelah magnetometer sebesar 6.773°. Sehingga dapat disimpulkan kalibrasi magnetometer akan meminimalkan eror secara signifikan.

4.2.2. Pengujian Offset

Magnetometer memiliki *offset* yang berbeda-beda di tiap tempat. Dilakukan uji dengan mengukur *offset* pada 28 titik yang memiliki jarak tiap titik 1 meter. Didapatkan data pengukuran magnetometer pada Tabel 4.2 dibawah ini.

Tabel 4.2 Offset Magnetometer

No	**Referensi** (°)	**Leader** (°)	**Follower** (°)
1	0	5	1
2	0	13	7
3	0	7	−18
⋮	⋮	⋮	⋮
25	0	30	−2
26	0	14	1
27	0	20	0
28	0	13	10

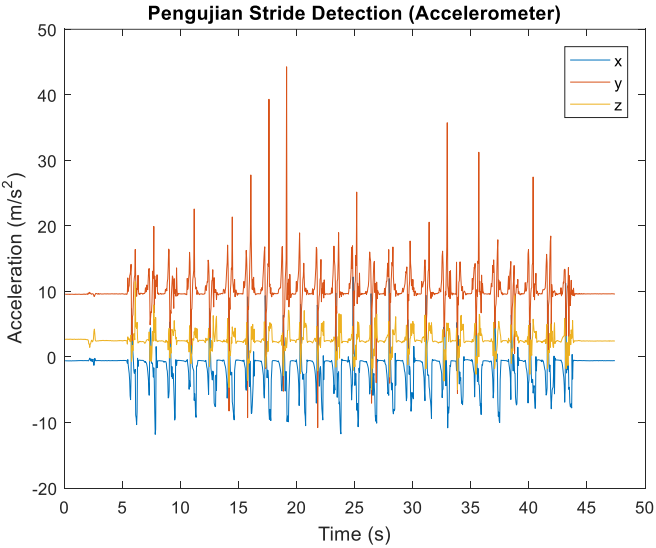
Dari tabel diatas, dapat dihitung RMSE antara *leader* dan referensi, *follower* dan referensi, dan *leader* dan *follower* sesuai pada Tabel 4.3 berikut ini.

Tabel 4.3 RMSE *Offset* Magnetometer

RMSE		
Leader & Referensi	**Follower & Referensi**	**Leader & Follower**
20.648 °	39.262 °	37.275 °

4.3. Pengujian Algoritma Deteksi Langkah

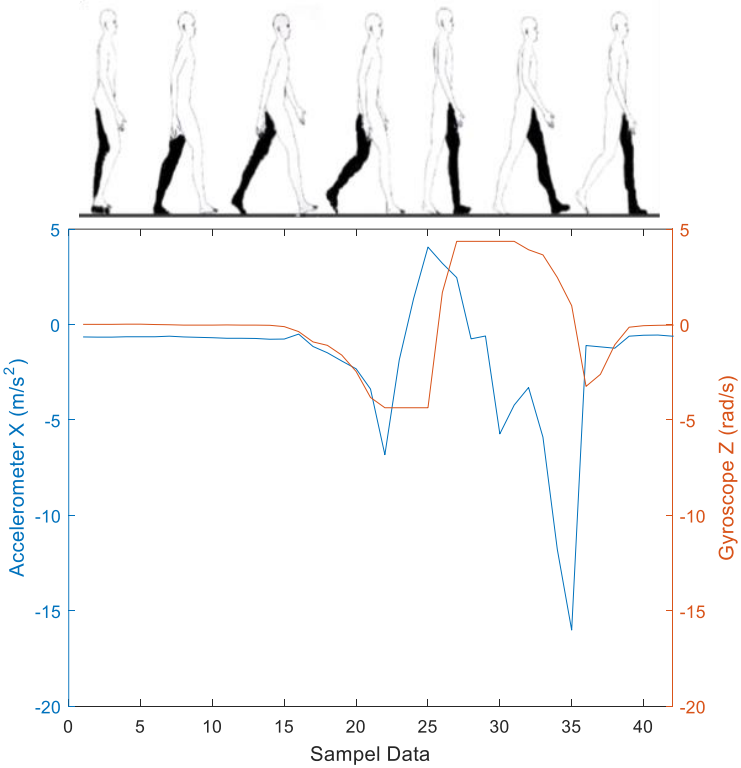
Dilakukan tuning parameter *threshold* untuk menguji keberhasilan algoritma deteksi langkah. Digunakan sinyal giroskop sumbu-z dan didapatkan parameter *threshold* untuk deteksi mulainya langkah adalah nilai absolut dari $2\,rad/s$ dan $0.1\,rad/s$ untuk deteksi selesainya langkah. Untuk meningkatkan performa deteksi langkah digunakan juga sinyal akselerometer sumbu x sebagai tanda langkah sedang terjadi.



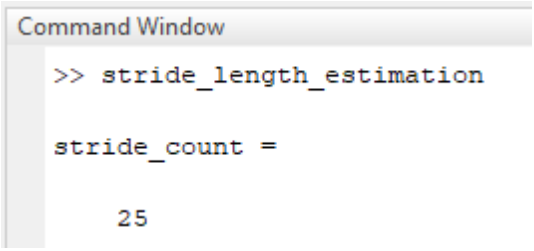
Gambar 4.2 Sinyal Akselerometer

Pengujian algoritma deteksi langkah dimaksudkan supaya semua data yang dibutuhkan untuk masukan estimasi panjang langkah dapat

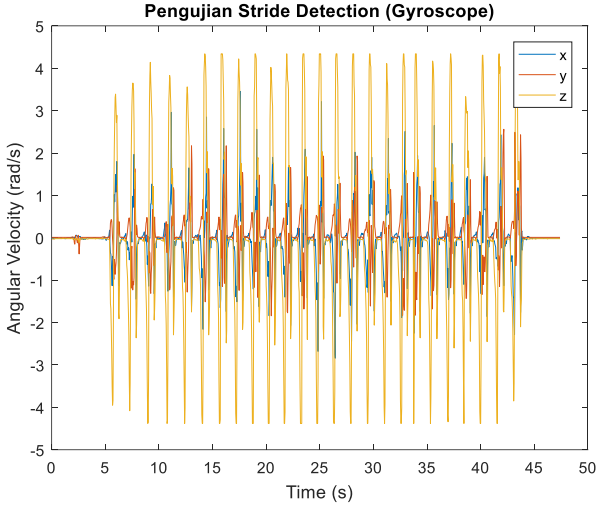
dipenuhi dengan benar. Pengujian dilakukan dengan melangkah sebanyak 25 langkah secara kontinyu.



Gambar 4.3 Fase Satu Siklus Berjalan



Gambar 4.4 Jumlah Langkah Terdeteksi

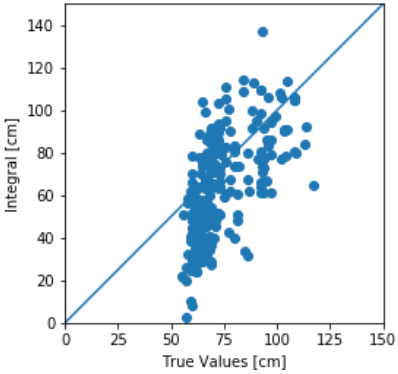


Gambar 4.5 Sinyal Giroskop

Dari Gambar 4.2 dan Gambar 4.5 dapat diamati bahwa semua sumbu akselerometer dan giroskop mengalami siklus yang berulang sebanyak 25 siklus serta dari Gambar 4.4 didapatkan hasil deteksi langkah berjumlah 25 langkah sehingga disimpulkan parameter *threshold* sudah optimal.

4.4. Pengujian Model Panjang Langkah

4.4.1. Integral Langsung

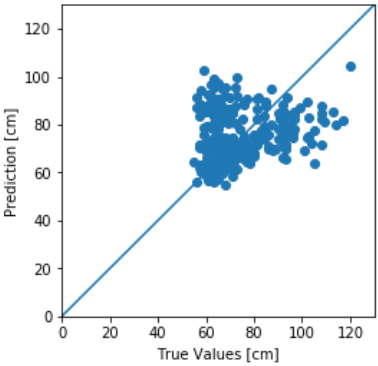


Gambar 4.6 Perbandingan Hasil Integral Langsung dengan Nilai Sebenarnya

Dilakukan kalkulasi integral secara langsung terhadap akselerometer pada sumbu-x untuk mendapatkan perpindahan posisi relatif. Didapatkan RMSE dari integral langsung bernilai 24.907 cm.

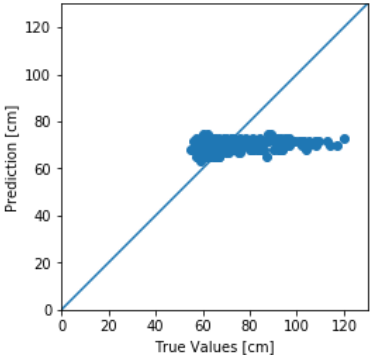
4.4.2. Model Nonlinear

Digunakan data *training* berjumlah 614 sampel untuk mencari konstanta K pada Persamaan (2.7). Didapatkan nilai konstanta K adalah 36.92. Kemudian model digunakan untuk estimasi pada seluruh data sampel dan didapatkan hasil RMSE bernilai 15.553 cm.



Gambar 4.7 Perbandingan Hasil Estimasi Model Nonlinear dengan Nilai Sebenarnya

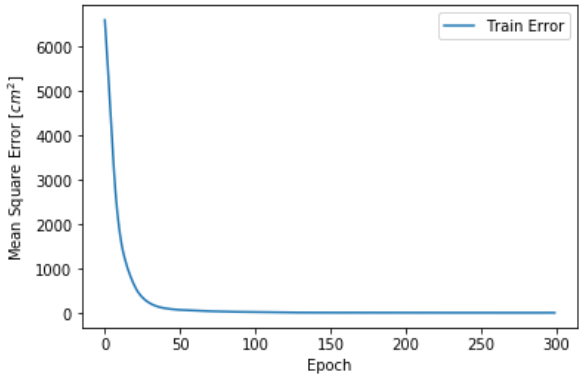
4.4.3. Model Frekuensi



Gambar 4.8 Perbandingan Hasil Estimasi Model Frekuensi dengan Nilai Sebenarnya

Sama seperti model nonlinear, diambil 614 sampel data *training* digunakan untuk mencari konstanta a dan b pada Persamaan (2.8). Didapatkan nilai konstanta a adalah 1.635 dan b adalah 33.835. Kemudian model digunakan untuk prediksi seluruh sampel data dan didapatkan hasil RMSE bernilai 14.664 cm.

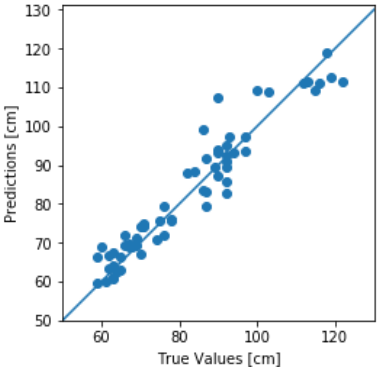
4.4.4. Model ANN



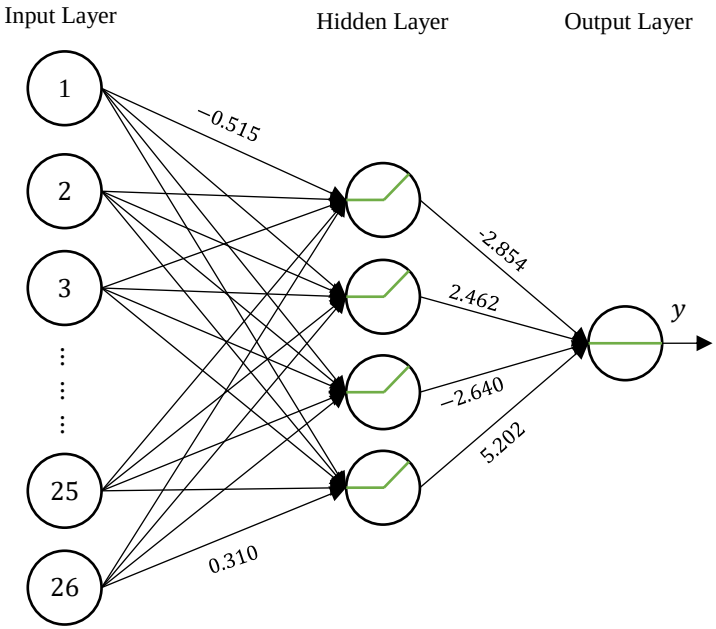
Gambar 4.9 MSE dari *Training* Data Model ANN

Tabel 4.4 Uji ANN

Hidden Layer	Neuron	RMSE Train (cm)	RMSE Test (cm)
1	1	5.83	5.55
	2	4.84	5.71
	3	4.94	5.11
	4	4.65	5.04
	5	4.16	5.75
	6	4.24	6.47
	⋮	⋮	⋮
	26	3.85	6.70
2	1, 1	5.98	5.53
	2, 1	5.72	5.41
	2, 2	5.59	5.60
	3, 2	5.42	5.04
	3, 3	5.24	6.27
	⋮	⋮	⋮
	26, 26	4.50	9.11



Gambar 4.10 Perbandingan Hasil Estimasi Model ANN dengan Nilai Sebenarnya



Gambar 4.11 Struktur Model ANN Satu *Hidden Layer* Empat *Neuron*

Dilakukan uji coba *training* dengan jumlah *hidden layer* beserta *neuron* yang berbeda-beda. RMSprop *optimizer* digunakan sebagai

optimisasi *learning* karena dari hasil pengujian memiliki performa terbaik dibandingkan SGD, Momentum, Adagrad, Adadelta, dan Adam. Selain itu, fungsi aktivasi ReLU dipilih karena dari pengujian *learning* eror ReLU lebih kecil dibandingkan Sigmoid. Dari hasil pengujian, didapatkan hasil uji seperti pada Tabel 4.4

Dari Tabel 4.4, dapat diamati bahwa semakin banyak *neuron* dan *hidden layer*, RMSE *training* akan semakin kecil, namun hasil RMSE *testing* menunjukkan sebaliknya. Hal ini kemungkinan disebabkan model menjadi *overfitting* ketika terlalu banyak *neuron* atau *hidden layer*. Dari hasil tersebut dipilihlah model dengan satu *hidden layer* dan empat *neuron* sebagai model panjang langkah.

4.4.5. Perbandingan Model

Dari subbab 4.4.1 sampai subbab 4.4.4 dapat disimpulkan bahwa model ANN adalah model terbaik untuk model panjang langkah dengan RMSE 5.04 cm sehingga model ANN akan digunakan sebagai model panjang langkah.

Tabel 4.5 Perbandingan RMSE Model Panjang Langkah

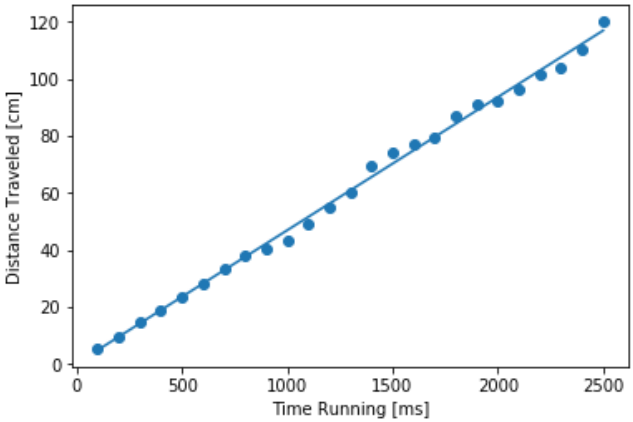
Integral Langsung	Model Non-linear	Model Frekuensi	Model ANN
24.907 cm	15.553 cm	14.664 cm	5.04 cm

4.5. Pengujian Model Jarak Tempuh Follower

Dari model pada Persamaan (3.6) dilakukan pengujian model dengan memberikan masukan 100, 200, sampai 2500. Didapatkan hasil pengujian pada Tabel 4.6 dan Gambar 4.12 dibawah ini.

Tabel 4.6 Uji Model Jarak Tempuh

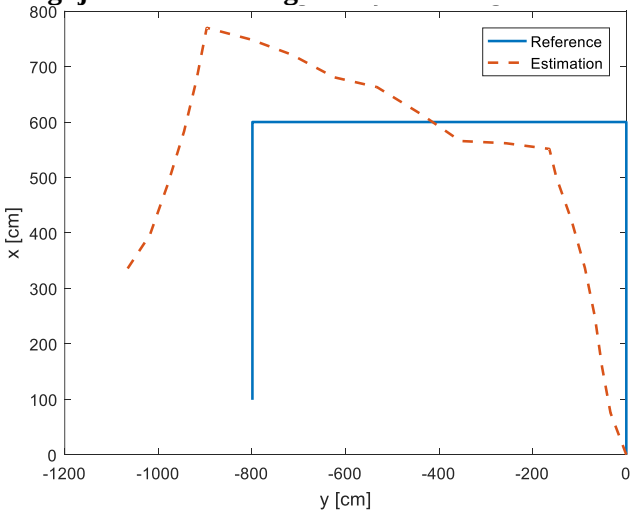
Running Time (ms)	Jarak Tempuh Sebenarnya (cm)	Jarak Tempuh Model (cm)
100	5.5	5.0
200	9.5	9.3
300	14.9	14.0
⋮	⋮	⋮
2300	103.8	107.3
2400	110.3	111.9
2500	120.0	116.6



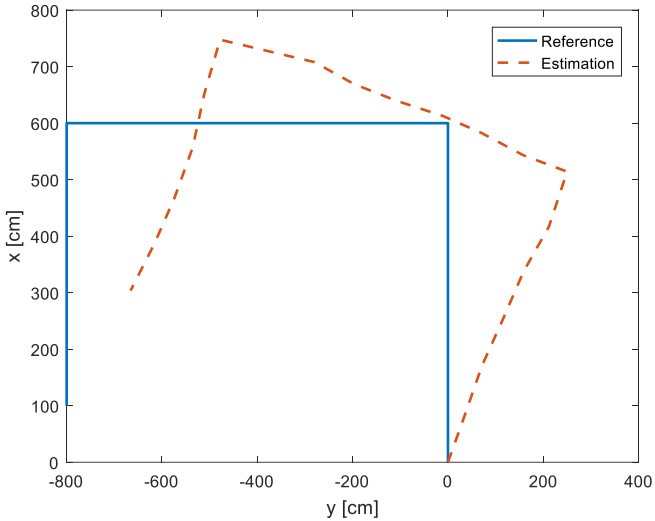
Gambar 4.12 Hasil Regresi Linear Model Jarak Tempuh *Follower*

Dari Tabel 4.6 dan Gambar 4.12 didapatkan nilai RMSE sebesar 2.0956 cm. Meskipun dari nilai RMSE tergolong kecil, pada implementasinya akurasi model dapat menurun dikarenakan salah satunya slip roda dengan lantai yang selalu berubah-ubah.

4.6. Pengujian Sistem Navigasi Leader



Gambar 4.13 Lintasan Hasil Sistem Navigasi *Leader* (*Indoor*)



Gambar 4.14 Lintasan Hasil Sistem Navigasi *Leader* (*Outdoor*)

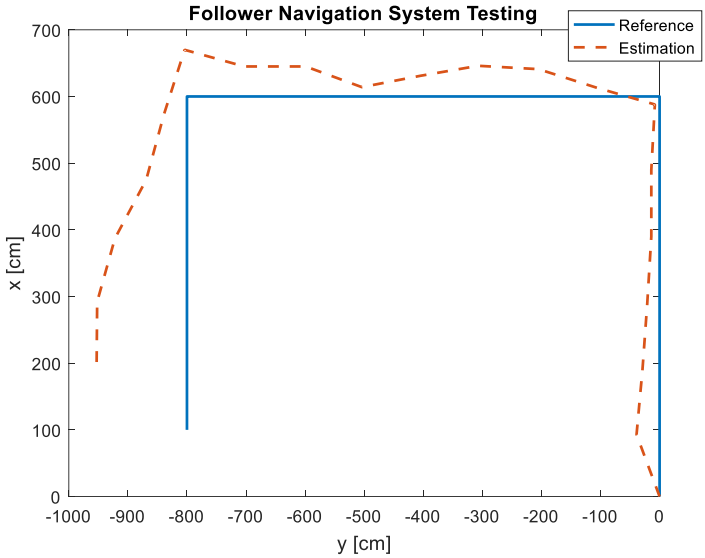
Diambil inisialisasi *heading* dari pengukuran magnetometer smartphone, kemudian *leader* berjalan sesuai dengan lintasan yang telah dirancang pada Gambar 3.27 dan Gambar 3.28. Didapatkan lintasan hasil estimasi sistem navigasi *leader* pada Gambar 4.13 dan Gambar 4.14 dengan RMSE tiap variabel pada Tabel 4.7.

Tabel 4.7 RMSE Pengujian Sistem Navigasi *Leader*

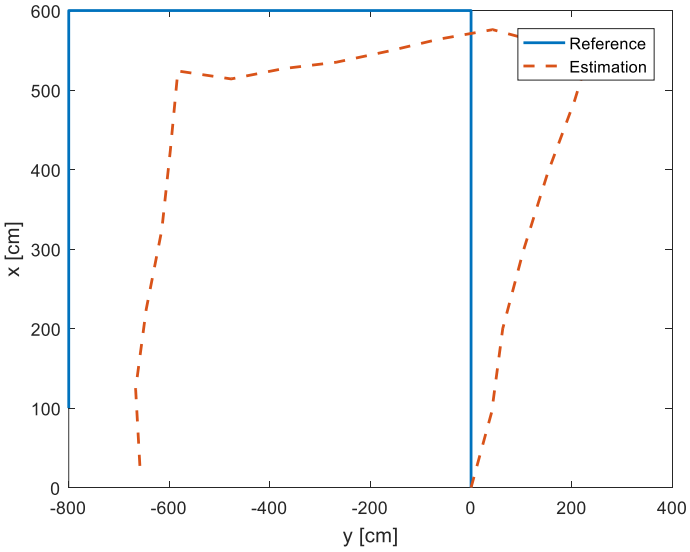
	RMSE Panjang Langkah (cm)	**RMSE Heading (°)**
Indoor	7.787	38.048
Outdoor	5.856	22.664

4.7. Pengujian Sistem Navigasi Follower

Diambil inisialisasi *heading* dari pengukuran magnetometer smartphone, kemudian *follower* dijalankan melalui pengiriman *waypoint* secara manual sesuai dengan Gambar 3.27 dan Gambar 3.28. Didapatkan hasil lintasan estimasi sistem navigasi *follower* pada Gambar 4.15 dan Gambar 4.16 dengan RMSE tiap variabel pada Tabel 4.8**.**



Gambar 4.15 Lintasan Hasil Sistem Navigasi *Follower* (*Indoor*)

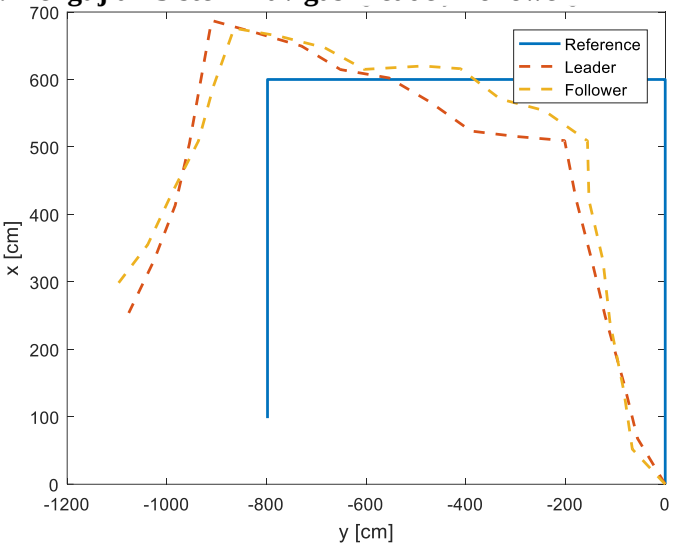


Gambar 4.16 Lintasan Hasil Sistem Navigasi *Follower* (*Outdoor*)

Tabel 4.8 RMSE Pengujian Sistem Navigasi *Follower*

	RMSE Jarak Tempuh (cm)	RMSE Heading (°)
Indoor	2.156	13.620
Outdoor	5.198	15.526

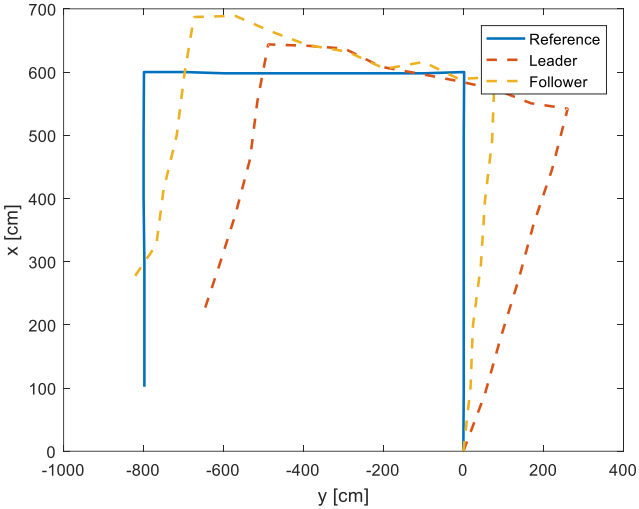
4.8. Pengujian Sistem Navigasi Leader-Follower



Gambar 4.17 Lintasan Hasil Sistem Navigasi *Leader-Follower* (*Indoor*)

Tabel 4.9 Hasil Pengujian Sistem Navigasi *Leader-Follower*

	Kondisi	RMSE Panjang Langkah & Jarak Tempuh (cm)	RMSE Heading (°)
Indoor	*Leader - Reference*	8.00	20.54
	Follower - Leader	5.97	11.00
Outdoor	*Leader – Reference*	6.02	19.31
	Follower - Leader	3.22	12.88



Gambar 4.18 Lintasan Hasil Sistem Navigasi *Leader-Follower* (*Outdoor*)

Pengujian ini dilakukan dengan *leader* berjalan sesuai Gambar 3.27 dan Gambar 3.28 yang kemudian hasil estimasi panjang langkah dan *heading* dikirim tiap langkah ke *follower*. *Follower* akan berjalan sesuai dengan *waypoint* yang diterima dari *leader*. Hasil dari pengujian ini dapat dilihat pada Gambar 4.17 dan Gambar 4.18 dengan RMSE tiap variabel pada Tabel 4.9**.**

4.9. Analisa Ketercapaian Spesifikasi Desain

Diperoleh hasil sistem navigasi *leader-follower* pada Tabel 4.9 dan diambil nilai eror terbesar sebagai perbandingan dengan spesifikasi desain. Dengan panjang lintasan referensi sepanjang 19 meter, hasil estimasi panjang lintasan *leader* sepanjang 17.61 meter, dan hasil estimasi jarak tempuh robot sebesar 17.51 meter diperoleh perbandingan hasil dengan spesifikasi desain sistem pada Tabel 4.10 dibawah ini.

Tabel 4.10 Perbandingan Hasil dengan Spesifikasi Desain Sistem

Variabel	Spesifikasi Desain	Hasil
Heading	2°	20.54°
Panjang langkah	38 cm	139.08 cm
Jarak Tempuh	35.22 cm	9.76 cm

Dari Tabel 4.10, dapat diamati bahwa eror *heading* jauh lebih besar dibandingkan target spesifikasi desain. Berdasarkan [19], perbedaan ini kemungkinan disebabkan dua faktor diantaranya kurangnya variabel kalibrasi magnetometer dan benda-benda yang mendistorsi medan magnet disekitar magnetometer. Kemudian untuk panjang langkah, eror hasil lebih besar dari spesifikasi desain kemungkinan dikarenakan kurangnya *dataset* langkah untuk *learning* model. Terakhir eror jarak tempuh sudah dapat diamati bahwa sudah sesuai bahkan lebih kecil dibandingkan target spesifikasi desain sistem.

BAB 5
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Dari hasil pengujian dapat disimpulkan beberapa poin sebagai berikut:

- Estimasi jarak tempuh adalah satu-satunya variabel yang sesuai atau lebih baik dari target spesifikasi desain sistem.
- Estimasi panjang langkah pada *leader* terbaik menggunakan model ANN dengan satu *hidden layer* dan empat *neuron*. Hal ini dibuktikan dengan eror *testing* RMSE yang bernilai 5.04 cm yang mana terkecil dibandingkan model panjang langkah yang lain. Akan tetapi hasil estimasi variabel ini masih belum sesuai dengan spesifikasi desain dikarenakan kurangnya jumlah *dataset learning* model.
- Estimasi *heading* masih belum sesuai dengan spesifikasi desain sehingga menyebabkan sistem navigasi *leader-follower* masih kurang akurat. Hal ini kemungkinan dikarenakan kurang akuratnya kalibrasi serta sifat magnetometer yang mudah terdistorsi oleh benda-benda disekitarnya.
- Gerakan robot dari satu titik ke titik lain dapat dimodelkan sebagai model linear antara jarak tempuh dengan waktu tempuh robot dengan kesalahan RMSE sebesar 2.0956 cm.

5.2. Saran

Dari hasil pengujian diatas dapat dianalisis berbagai macam kekurangan dalam sistem navigasi *leader-follower* sehingga untuk penelitian selanjutnya terdapat beberapa saran diantaranya:

- Meningkatkan algoritma kalibrasi magnetometer sehingga didapat *heading* yang lebih akurat.
- Menambahkan atau mengganti sensor *heading* lain yang lebih akurat di berbagai lingkungan.
- Meningkatkan akurasi model panjang langkah ANN dengan memodifikasi masukan ataupun jumlah *hidden layer* dan *neuron*.
- Menambahkan sensor posisi pada robot sehingga posisi robot dapat diestimasi lebih akurat.
- Penambahan algoritma dan sensor untuk menghindari gangguan.

DAFTAR PUSTAKA

[1] "ACROBOTIC ESP8266 ESP-01 Serial to Wi-Fi Module." *Acrobotic, ACROBOTIC Industries*, https://acrobotic.com/products/acr-00020.

[2] Al-Araji, Ahmed S., Maysam F. Abbod, and Hamed S. Al-Raweshidy. "Design of an adaptive neural predictive nonlinear controller for nonholonomic mobile robot system based on posture identifier in the presence of disturbance." (2011).

[3] Arduino. https://www.arduino.cc

[4] Egli, Peter R. "An introduction to MQTT, a protocol for M2M and IoT applications." *indigoo.com* (2015).

[5] Google Developers. "Machine Learning Crash Course with TensorFlow APIs", https://developers.google.com/machine-learning/crash-course/

[6] Hellström, Thomas. "Kinematics equations for differential drive and articulated steering". *Department of Computing Science, Umeå University* (2011).

[7] Honeywell. "3-Axis Digital Compass IC HMC5883L", https://www.jameco.com/Jameco/Products/ProdDS/2150248.pdf

[8] Krenker, Andrej, Janez Bešter, and Andrej Kos. "Introduction to the Artificial Neural Networks." *Artificial Neural Networks-Methodological Advances and Biomedical Applications*. *IntechOpen* (2011).

[9] Lima, Pedro, and Maria Isabel Ribeiro. "Mobile robotics." *Course Handouts, Instituto Superior Técnico/Instituto de Sistemas e Robótica* (2002).

[10] Mazur, Matt. "A Step by Step Backpropagation Example", https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ (2015).

[11] Nugroho, Hary, and Sartika Siagian Aprilia. "Esp8266ex Datasheet, 2015, Espressif Systems IOT Team."

[12] O'Reilly, Rob, Alex Khenkin, and Kieran Harney. "Sonic nirvana: Using mems akselerometers as acoustic pickups in musical instruments." *Analog Dialogue 43.02* (2009): 1-4.

[13] Robot Platform. "Robot Locomotion", http://www.robotplatform.com/knowledge/Classification_of_Robots/Holonomic_and_Non-Holonomic_drive.html

[14] Rogers, R. M., "Applied Mathematics in Integrated Navigation Systems", *AIAA Education Series*, 2000.

[15] Ruder, Sebastian. "An overview of gradient descent optimization algorithms." *arXiv preprint arXiv:1609.04747* (2016).

[16] Sarle, Warlen. "comp.ai.neural-nets FAQ, Part 2 of 7: Learning", http://www.faqs.org/faqs/ai-faq/neural-nets/part2/

[17] Shaaban, Muhammad. "Microcontroller Basics" Class Lecture, *EECC250 from Rochester Institute of Technology, New York*, February 16 (2019).

[18] Tran, Long Dinh. "Data Fusion with 9 Degrees of Freedom Inertial Measurement Unit To Determine Object's Orientation." (2017).

[19] VectorNav. "Magnetometer", https://www.vectornav.com/support/library/magnetometer.

[20] Watson, Jeff. "MEMS Giroskop Provides Precision Inertial Sensing in Harsh, High Suhue Environments" *Analog Devices*, *Analog Devices*, https://www.analog.com/en/technical-articles/mems-giroskop-provides-precision-inertial-sensing.html (2016).

[21] Winer, Kris. Simple and Effective Magnetometer Calibration, (2017), *GitHub repository*, https://github.com/kriswiner/MPU6050/wiki/Simple-and-Effective-Magnetometer-Calibration

[22] Wetzstein, Gordon. "Inertial Measurement Unit I" *Class Lecture*, *Virtual Reality from Stanford University*, *California* (2019).

[23] Xing, Haifeng, et al. "Pedestrian stride length estimation from IMU measurements and ANN based algorithm." *Journal of Sensors 2017* (2017).

LAMPIRAN

Lampiran 1. Program Kalibrasi Giroskop [Arduino]

```
int MPU9250::calibrateGyro() {
  _gxbD = 0;
  _gybD = 0;
  _gzbD = 0;
  for (size_t i=0; i < _numSamples; i++) {
    readSensor();
    _gxbD += (getGyroX()+_gxb)/((double)_numSamples);
    _gybD += (getGyroY()+_gyb)/((double)_numSamples);
    _gzbD += (getGyroZ()+_gzb)/((double)_numSamples);
    delay(20);
  }
  _gxb = (float)_gxbD;
  _gyb = (float)_gybD;
  _gzb = (float)_gzbD;
}
void MPU9250::setGyroBiasX_rads(float bias) {
  _gxb = bias;
}
void MPU9250::setGyroBiasY_rads(float bias) {
  _gyb = bias;
}
void MPU9250::setGyroBiasZ_rads(float bias) {
  _gzb = bias;
}
```

Lampiran 2. Program Start *Server* MQTT [Kotlin]

```
val server = Server()
val memoryConfig = MemoryConfig(Properties())
memoryConfig.setProperty(BrokerConstants.PERSIST…
                         ENT_STORE_PROPERTY_NAME,…
                         Environment.getExternalStor…
                         ageDirectory().absolutePath…
                         +      File.separator      +…
                         BrokerConstants.DEFAULT_MO…
                         QUETTE_STORE_MAP_DB_FILENAME
                         )
server.startServer(memoryConfig)
Toast.makeText(applicationContext, "MQTT Server has…
                Started", Toast.LENGTH_LONG).show()
```

```
val clientId = MqttClient.generateClientId()
if(client==null){
 client     =     MqttAndroidClient(applicationContext,
                 "tcp://192.168.43.1:1883", clientId)
}
```

Lampiran 3. Program Kalibrasi Otomatis Magnetometer [Kotlin]

```
val mag=ArrayList<Double>()
val offset=ArrayList<Double>()
val avgDelta=ArrayList<Double>()
mag.add(allSeparatedData[6].toDouble())
mag.add(allSeparatedData[7].toDouble())
mag.add(allSeparatedData[8].toDouble())
for(iMag in 0..2){
 if(mag[iMag]>maxMag[iMag]){
  maxMag[iMag]=mag[iMag]
 }
 if(mag[iMag]<minMag[iMag]){
  minMag[iMag]=mag[iMag]
 }
 offset.add((maxMag[iMag]+minMag[iMag])/2)
 avgDelta.add((maxMag[iMag]-minMag[iMag])/2)
}
val allAvgDelta =(avgDelta[0]+avgDelta[1]+ …
    avgDelta[2])/3
val scale=ArrayList<Double>()
val magCorrected=ArrayList<Double>()
for(iMag in 0..2){
 scale.add(allAvgDelta/avgDelta[iMag])
 magCorrected.add((mag[iMag]-offset[iMag])* …
 scale[iMag])
}
```

Lampiran 4. Program Publish rawData IMU *Leader* [ESP8266]

```
IMU.readSensor();
ax = IMU.getAccelX();
ay = IMU.getAccelY();
az = IMU.getAccelZ();
gx = IMU.getGyroX();
```

```
gy = IMU.getGyroY();
gz = IMU.getGyroZ();
mx = IMU.getMagX();
my = IMU.getMagY();
mz = IMU.getMagZ();
t  = IMU.getTemp();
if(iData<5){
 sacc+=String(ax)+separator+String(ay)+separator+…
       String(az)+";";
 sgyr+=String(gx)+separator+String(gy)+separator+…
       String(gz)+";";
 smag+=String(mx)+separator+String(my)+separator+…
       String(mz)+";";
 stem+=String(t)+";";
 iData++;
 // Send data after 15 data are collected
 if(iData==5){
  sacc.toCharArray(acc,sacc.length()+1);
  sgyr.toCharArray(gyr,sgyr.length()+1);
  smag.toCharArray(mag,smag.length()+1);
  stem.toCharArray(tem,stem.length()+1);
  client.publish("PDR/Data/Acc",acc);
  client.publish("PDR/Data/Gyr",gyr);
  client.publish("PDR/Data/Mag",mag);
  client.publish("PDR/Data/Tem",tem);
  sacc="";
  sgyr="";
  smag="";
  stem="";
  iData=0;
 }
}
```

Lampiran 5. Program Deteksi Langkah [Matlab]

```
if(process==0)
  isStart = detect_start_stride(gyr(i,3));
  if(isStart)
    process = 1;
  end
elseif(process==1)
  isMax = detect_max_acc(acc(i,1),acc_m1(1));
  if(isMax)
```

```
    process=2;
  end
elseif(process==2)
  isMin = detect_min_acc(acc(i,1),acc_m1(1));
  if(isMin)
    process=3;
  end
elseif(process==3)
  isStop = detect_stop_stride(gyr(i,3), gyr_m1(3));
  if(isStop)
    process=4; %ANN
  end
```

---------- program break

Lampiran 5.1. Fungsi detect_start_stride(gyr(i,3))

```
function isStart = detect_start_stride(gyr)
  if(abs(gyr)>2)
    isStart=true;
  else
    isStart=false;
  end
end
```

Lampiran 5.2. Fungsi detect_max_acc(acc(i,1),acc_m1(1))

```
function isMax = detect_max_acc(acc,acc_m1)
  acc_thold = 2;
  if(acc>acc_thold)
    if(acc_m1>acc_thold)
      if(acc<acc_m1)
        isMax=true;
      else
        isMax=false;
      end
    else
      isMax=false;
    end
  else
    if(acc_m1>acc_thold)
      isMax=true;
    else
      isMax=false;
    end
  end
end
```

Lampiran 5.3. Fungsi detect_min_acc(acc(i,1),acc_m1(1))

```
function isMin = detect_min_acc(acc,acc_m1)
  acc_thold=-2;
  if(acc<acc_thold)
    if(acc_m1<acc_thold)
      if(acc>acc_m1)
        isMin=true;
      else
        isMin=false;
      end
    else
      isMin=false;
    end
  else
    if(acc_m1<acc_thold)
      isMin=true;
    else
      isMin=false;
    end
  end
end
```

Lampiran 5.4. Fungsi detect_start_stride(gyr(i,3))

```
function isStop = detect_stop_stride(gyr, gyr_m1)
  if(abs(gyr)<0.1 && abs(gyr_m1)<0.1)
    isStop=true;
  else
    isStop=false;
  end
end
```

Lampiran 6. Program Tilt Compensated Heading [Kotlin]

```
val accY = acc[1]
val accZ = acc[2]
val gyrY = gyr[1]
val accOrientation = arrayListOf( …
                     atan2(accY,accZ),atan2(acc[0], …
                     sqrt((accY*accY)+(accZ*accZ))))
val gyrOrientation = arrayListOf((gyr[0]*0.04), …
                      (gyrY*0.04))
for(_i in 0..1){
 orientation[_i] = 0.98*(orientation[_i]+ …
                   gyrOrientation[_i])+0.02* …
                   accOrientation[_i]
```

```
} // Complementary Filter : Combine Accel and Gyro
val magX = magCorrected[0]*cos(orientation[1]) + …
           magCorrected[2]*sin(orientation[1])
val magY = magCorrected[0]*sin(orientation[0])* …
           sin(orientation[1]) + magCorrected[1]*…
           cos(orientation[0]) - magCorrected[2]* …
           sin(orientation[0])*cos(orientation[1])

val yaw=atan2(magY,magX) //Get Heading in radian
```

Lampiran 7. Program Learning Model ANN [Python]

```
## Build Model
Model = tf.keras.Sequential([
              tf.layers.Dense(3,activation="relu",inpu
       t_shape=([26])),
              tf.layers.Dense(1)
    ])
optimizer = tf.keras.optimizers.RMSprop(0.01)
model.compile(loss='mean_squared_error',
            optimizer=optimizer,
            metrics=['mean_absolute_error',
'mean_squared_error'])
model.summary()

## Model Training
EPOCHS = 300
history = model.fit(
  normed_train_data, train_labels,
  epochs=EPOCHS, validation_split = 0, verbose=0,
  callbacks=[PrintDot()])
```

Lampiran 8. Program Inference Model ANN [Kotlin]

```
// Import Model Weight and Data
val meanFile = File(Environment.getExternalStorage …
               Directory().toString() + "/Documents …
               /TA/Model/mean.csv")
val stdFile = File(Environment.getExternalStorage …
               Directory().toString() + "/Documents …
               /TA/Model/std.csv")
val wih1File = File(Environment.getExternalStorage …
               Directory().toString() + "/Documents …
```

```
                /TA/Model/wih1.csv")
val woFile = File(Environment.getExternalStorage …
                Directory().toString() + "/Documents …
                /TA/Model/wo.csv")
val bih1File = File(Environment.getExternalStorage …
                Directory().toString() + "/Documents …
                /TA/Model/bih1.csv")
val boFile = File(Environment.getExternalStorage …
                Directory().toString() + "/Documents …
                /TA/Model/bo.csv")
mean = CSVReader(FileReader(meanFile.absolute …
       Path)).readAll()
std = CSVReader(FileReader(stdFile.absolute …
      Path)).readAll()
wih1 = CSVReader(FileReader(wih1File.absolute …
       Path)).readAll()
bih1 = CSVReader(FileReader(bih1File.absolute …
       Path)).readAll()
wo = CSVReader(FileReader(woFile.absolute …
     Path)).readAll()
bo = CSVReader(FileReader(boFile.absolute …
     Path)).readAll()
// Get All Raw Features
val featuresRaw = arrayListOf<Double>()
for (j in 0..2){
 featuresRaw.add(accGyrMax[j])
}
for (j in 0..2){
 featuresRaw.add(accGyrMin[j])
}
for (j in 0..2){
 featuresRaw.add(accGyrMean[j])
}
for (j in 0..2){
 featuresRaw.add(accGyrVar[j])
}
for (j in 3..5){
 featuresRaw.add(accGyrMax[j])
}
for (j in 3..5){
 featuresRaw.add(accGyrMin[j])
```

```
}
for (j in 3..5){
 featuresRaw.add(accGyrMean[j])
}
for (j in 3..5){
 featuresRaw.add(accGyrVar[j])
}
featuresRaw.add(strideFrequency.toDouble())
featuresRaw.add(temp)
val features = arrayListOf<Double>()
for((index, featureRaw) in featuresRaw.withIndex()){
 val feature = (featureRaw-mean[index][0].to …
               Double())/std[index][0].toDouble()
               // Standardize Features
 features.add(feature)
}
var sumOutWeight = 0.0
// Inference
for (i in 0..4){
 var sumLayerWeight = 0.0
 for((index,feature) in features.withIndex()){
  sumLayerWeight += feature*wih1[i][index].toDouble()
 }
 val netH1 = bih1[i][0].toDouble() + sumLayerWeight
 val outH1 = Math.max(0.0,netH1)
 sumOutWeight += outH1*wo[i][0].toDouble()
}
val netO = bo[0][0].toDouble() + sumOutWeight
```

Lampiran 9. Program Subscribe Waypoint Robot [Arduino]

```
if((strcmp(topic,"PDR/Leader/Position")==0)){
 for (byte i = 0; i < length; i++) {
  if ((char)payload[i] == ',') {
   char sLengthReceived[i+1];
   char headingReceived[length-i];
   for(byte j =0; j<i;j++){
    sLengthReceived[j] = (char) payload[j];
   }
   byte k=0;
   for(byte j =i+1; j<length;j++){
    headingReceived[k] = (char) payload[j];
```

```
    k++;
   }
   sLengthTarget[iWaypointReceived] = atoi(sLength…
                                      Received);
   headingTarget[iWaypointReceived] = atoi(heading…
                                      Received);
   break;
  }
  client.loop();
 }
 iWaypointReceived++;
 if(iWaypointReceived==25){
  iWaypointReceived=0;
 }
 client.publish("PDR/isReceived", "1"); //Feedback to
                                         server
}
```

Lampiran 10. Program Maju & Belok Robot [Arduino]

```
// If waypoint is coming, start algorithm
// process = 1 > Rotate robot to waypoint heading
// process = 2 > Go to waypoint distance
if(iWaypointTarget<iWaypointReceived){
 float theta_now = getHeading();
 float theta_target = headingTarget[iWaypointTarget];
 if((theta_target-theta_now<5 && theta_target-…
     theta_now>-5) || theta_target-theta_now>355 ||…
     theta_target-theta_now<-355){
  process=2;
 }
 if(process==1){
  float timeRun = getRunningTime("theta",theta_now,…
                  theta_target);
  prevTime=millis();
  if(timeRun>0){
   if(timeRun<50){
    timeRun=50;
   }
   while(millis()-prevTime<timeRun){
    rotate("CCW");
    delay(10);
    client.loop();
   }
```

```
  }
  else{
   if(timeRun>-50){
    timeRun=-50;
   }
   while(millis()-prevTime<-timeRun){
    rotate("CW");
    delay(10);
    client.loop();
   }
  }
  turnOffMotors();
  theta_now = getHeading();
  if((theta_target-theta_now<5 && theta_target-…
      theta_now>-5) || theta_target-theta_now>355 ||…
      theta_target-theta_now<-355){
   process=2;
  }
 }
 else if(process==2){
  float timeRun = 21.239*sLengthTarget[iWaypoint…
                  Target]-6.022; // Robot Distance
                  Linear Model
  prevTime=millis();
  while(millis()-prevTime<timeRun){
   go();
   delay(10);
   client.loop();
  }
  turnOffMotors();
  iWaypointTarget++;
  if(iWaypointTarget==25){
   iWaypointTarget=0;
  }
  process = 1;
 }
}
```

RIWAYAT HIDUP PENULIS

Muhammad Farih panggilan M lahir di Gresik pada tanggal 21 Mei 1997 dari pasangan suami istri Bapak Shun'an dan Ibu Siti Maryam. Penulis adalah anak kelima dari lima bersaudara. Penulis sekarang bertempat di Keputih gg 2 no 11, Sukolilo, Surabaya.

Penulis telah menempuh pendidikan mulai dari SDN Kalirejo yang lulus pada tahun 2009, MTs YKUI Maskumambang lulus pada tahun 2012, MA YKUI Maskumambang lulus pada tahun 2015, dan mulai 2015 menempuh pendidikan S1 Departemen Teknik Elektro, Institut Teknologi Sepuluh Nopember, Surabaya.

Saat menempuh pendidikan di Teknik Elektro ITS, penulis aktif di UKM Robotika dan bergabung dengan Tim Bayucaraka ITS divisi Fixed Wing. Selain itu, pada tahun ke-4 penulis juga bergabung dengan penelitian Laboratorium Sistem dan Sibernetik di bidang *Inertial Navigation Systems*.