NAVIGASI ROBOT KESEIMBANGAN DENGAN VIRTUAL MAP DAN VIRTUAL SENSOR

SKRIPSI

Diajukan sebagai salah satu syarat
untuk memperoleh gelar
Sarjana Teknik

Oleh:

HENRY PROBO SANTOSO
NIM. I0716016

PROGRAM STUDI TEKNIK ELEKTRO
FAKULTAS TEKNIK UNIVERSITAS SEBELAS MARET
SURAKARTA
2020

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN
UNIVERSITAS SEBELAS MARET
FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK ELEKTRO
Jalan. Ir. Sutami nomor 36 A Kentingan Surakarta 57126
Telepon. 0271 647069 psw 438, faksimili: 0271 662118

SURAT TUGAS

Nomor : 063/TA/TE/2020

Kepala Program Studi Teknik Elektro Fakultas Teknik Universitas Sebelas Maret memberikan tugas kepada:

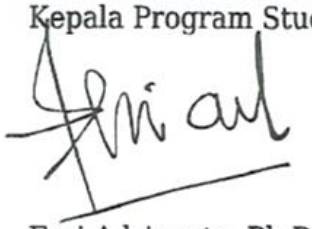
Nama Mahasiswa	:	**Henry Probo Santoso**
NIM	:	**I0716016**
Bidang peminatan	:	**Sistem Mekatronika (SM)**
Pembimbing Utama	:	**Hari Maghfiroh M.Eng.** **NIP. 199104132018031001**
Pembimbing Pendamping	:	**Joko Slamet Saputro, S.Pd., M.T.** **NIP. 198904242019031013**
Mata kuliah pendukung	:	**1. Mekatronika** **2. Sistem Kontrol Terintegrasi** **3. Teknik Robot**

untuk mengerjakan dan menyelesaikan Tugas Akhir dengan judul :

Navigasi Robot Keseimbangan dengan Virtual Map dan Virtual Sensor

Surat tugas ini dibuat untuk dilaksankan dengan sebaik-baiknya.

Surakarta, 16 Maret 2020
Kepala Program Studi



Feri Adriyanto, Ph.D.
NIP. 196801161999031001

Tembusan:

1. Mahasiswa ybs.
2. Dosen Pembimbing TA
3. Koordinator TA
4. Arsip

SURAT PERNYATAAN

ORISINALITAS KARYA ILMIAH

Saya mahasiswa Program Studi Teknik Elektro Universitas Sebelas Maret yang bertanda tangan di bawah ini:

Nama	:	Henry Probo Santoso
NIM	:	I0716016
Judul Tugas Akhir	:	Navigasi Robot Keseimbangan dengan Virtual Map dan Virtual Sensor

Dengan ini menyatakan bahwa Tugas Akhir atau Skripsi yang saya susun tidak mencontoh atau melakukan plagiat dari karya tulis orang lain. Jika terbukti Tugas Akhir yang saya susun tersebut merupakan hasil plagiat dari karya orang lain maka Tugas Akhir yang saya susun tersebut dinyatakan batal dan gelar sarjana yang saya peroleh dengan sendirinya dibatalkan atau dicabut.

Demikian surat pernyataan ini saya buat dengan sebenarnya dan apabila di kemudian hari terbukti melakukan kebohongan maka saya sanggup menanggung segala konsekuensinya.

Surakarta, 29 Juli 2020

Henry Probo Santoso

NIM. I0716016

:

Navigasi Robot Keseimbangan dengan Virtual Map dan Virtual Sensor

Henry Probo Santoso

Abstrak

Kebutuhan akan robot otonom, akhir-akhir ini semakin meningkat, bersamaan dengan semakin majunya perkembangan teknologi robotika. Kemampuan navigasi tanpa perlu campur tangan manusia, menjadi salah satu kelebihan robot otonom. Untuk membuat robot otonom, dibutuhkan robot dengan efisiensi dan fleksibilitas yang tinggi, serta sistem navigasi yang handal. Efisiensi dan fleksibilitas yang tinggi dapat ditangani dengan penggunaan robot kesimbangan beroda dua. Sistem navigasi yang handal dapat dicapai dengan penggunaan *platform* ROS (*robot operating system).* Penelitian menghasilkan virtual robot, virtual map, dan virtual sensor pada simulasi di aplikasi GAZEBO, yang dapat divisualisasikan dalam RVIZ. Pembuatan map dan penggunaan sistem navigasi berjalan dalam sistem ROS, menghasilkan data kecepatan yang dikirimkan ke simulasi GAZEBO dan robot keseimbangan pada dunia nyata. Uji coba *tracking* simulasi menunjukan robot virtual mampu menuju lokasi tujuan dengan error rata-rata -0.052 m di sumbu X dan -0.05 m di sumbu Y. Uji coba *tracking* real menunjukan robot keseimbangan mampu menerima data kecepatan dari sistem ROS, untuk bergerak menuju titik tujuan berdasarkan virtual map dan virtual sensor, dengan error rata-rata -0.044 m pada sumbu X dan 0.38 m pada sumbu Y.

Kata Kunci : Navigasi, Otonom, Robot Keseimbangan Beroda Dua, ROS

Balancing Robot Navigation with Virtual Map and Virtual Sensor

Henry Probo Santoso

Abstract

The need for autonomous robots has recently increased, along with the rapid development of robotics technology. The ability to navigate without the need of human intervention, is one of the advantages of autonomous robots. To make an autonomous robot, robot with high efficiency, flexibility, and a reliable navigation system are needed. High efficiency and flexibility can be handled with the use of a two-wheeled balancing robot. A reliable navigation system can be achieved by using the ROS (robot operating system) platform. The research produces virtual robots, virtual maps, and virtual sensors on simulations in the GAZEBO application, which can be visualized in RVIZ. Map development and navigation system usage, run in the ROS system, producing speed data that sent to GAZEBO simulations and balancing robot in real world. The tracking simulation test shows that the virtual robot can reach the destination location with an average errorrs -0.052 *m on the X axis and* -0.05 m on the *Y axis. The real tracking test shows the balancing robot can receive speed data from the ROS system, to move towards the destination point based on the virtual map and virtual sensor, with an average errors -0.044 m on the X axis and 0.38 m on the Y axis.*

Keywords : Autonomous, Navigation, ROS , Two-Wheeled Balancing Robot

KATA PENGANTAR

Puji syukur kehadirat Tuhan Yang Maha Esa yang telah melimpahkan berkat dan anugerah-Nya sehingga penulis dapat menyelesaikan Skripsi dengan Judul "NAVIGASI ROBOT KESEIMBANGAN DENGAN VIRTUAL MAP DAN VIRTUAL SENSOR" dengan lancar dan tepat waktu. Skripsi ini disusun guna memenuhi persyaratan akademik dalam memperoleh kelulusan studi di Program Studi Teknik Elektro Universitas Sebelas Maret Surakarta.

Dalam penyusunan Skripsi ini penulis telah mendapatkan banyak bimbingan, doa, dukungan, dan bantuan dari berbagai pihak. Penulis mengucapkan terima kasih kepada :

1. Tuhan Yesus Kristus atas segala berkat, anugrah, dan kasih karunianya, sehingga penulis dapat melaksanakan dan menyelesaikan Skripsi ini dengan baik dan penuh penyertaan.
2. Segenap keluarga yang telah memberikan doa, dukungan, dan kasih sayang. Terutama mami, koko, dan om agus.
3. Bapak Hari Maghfiroh, S.T., M.Eng, M.Sc. selaku dosen pembimbing I yang selalu memberikan dukungan, bimbingan, arahan, dan kemudahan selama perkulihan dan pengerjaan skripsi.
4. Bapak Joko Slamet Saputro, S.Pd., M.T. selaku dosen pembimbing II yang selalu memberikan dukungan, ide, dan penyelesaian masalah dalam pengerjaan skripsi.
5. Bapak Feri Adriyanto, Ph.D., selaku Kepala Program Studi Teknik Elektro UNS dan Penguji I, atas bantuan dan arahan yang telah diberikan
6. Bapak Muhammad Hamka Ibrahim, S.T, M.Eng., selaku koordinator Tugas Akhir yang telah memberikan arahan dan dorongan.
7. Bapak Sutrisno, S.T., M.Sc., Ph.D selaku penguji II atas masukan dan koreksi yang telah diberikan, sehingga skripsi ini dapat lebih baik
8. Segenap dosen dan karyawan Program Studi Teknik Elektro UNS.
9. Teman-teman Mekatronika 2016 dan Teknik Elektro 2016.
10. Semua pihak yang tidak dapat disebutkan satu persatu, untuk segala bantuan yang telah diberikan.

11. Kakak-kakak dan adik-adik Tingkat Elektro yang telah memberikan semangat dan dukungan
12. Semua pihak yang tidak dapat disebutkan satu persatu, untuk segala bantuan yang telah diberikan.

Surakarta, 29 Juli 2020

Henry Probo Santoso

NIM.I0716016

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR ISTILAH

API	:	Kumpulan berbagai elemen yang memungkinkan untuk pengembang mengintegrasikan dua bagian dari aplikasi secara bersamaan
autonomous	:	Proses dimana sebuah objek yang menggunakan suatu perangkat elektronik dapat berjalan secara otomatis tanpa menggunakan alat penggerak(*remote control*)
command terminal	:	Sebuah perintah khusus yang diketikan pada terminal di sistem operasi linux
library	:	Kumpulan fungsi program dalam sebuah file yang dapat dipanggil pada suatu program
loop function	:	Fungsi perulangan pada sebuah kode yang digunakan mikrokontroler untuk melakukan suatu proses kerja
mobile robot	:	Sebuah robot yang memiliki kemampuan untuk bergerak disekitarnya
navigasi	:	Kemampuan untuk memandu pergerakan dari suatu posisi ke posisi lain yang dituju melalui penentuan posisi dan arah gerakanya
package	:	Merupakan sebuah direktori yang memuat *library*, *datasheet*, konfigurasi, perangkat lunak pihak ketiga, atau hal hal lain yang berguna
ROS	:	ROS atau *robot operating system* merupakan platform yang digunakan untuk berbagai implementasi robot
setup function	:	Fungsi awal pada sebuah kode yang hanya berjalan satu kali setiap kali mikrokontroler dinyalakan

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi membuat manusia semakin mudah untuk mengembangkan robot. Manusia mampu menciptakan mikroprosesor berbentuk *single board computer,* yang memungkinkan untuk membuat algoritma rumit yang dibutuhkan untuk membentuk sebuah robot [1]. Berbagai jenis robot telah dikembangkan, salah satunya adalah *autonomous mobile robot*. Akhir-akhir ini *autonomous mobile robot* banyak digunakan untuk berbagai tujuan, antara lain sebagai pengantar barang, pemandu wisata, dan lain lain [1], [2].

Navigasi dan pemetaan secara terus menerus, merupakan poin penting bagi sebuah *autonomous mobile robot*. Pada aplikasi sesungguhnya, informasi lingkungan sekitar memiliki berbagai bentuk dan sangat kompleks, robot dituntut untuk bisa beroperasi dengan baik dan efektif pada kondisi tersebut [3], [4]. Namun, hal yang paling mendasar bagi *autonomous mobile robot* adalah bagaimana menemukan jalan terbaik, aman, dan bebas hambatan dari titik awal menuju ke tujuan [4]. Untuk itu pemetaan dan navigasi direalisasikan dengan data hasil dari pengindraan dan peta yang telah dibangun. Semua fungsi tersebut akan diselesaikan oleh *Robot Operating System* atau ROS [5].

Autonomous mobile robot membutuhkan berbagai persyaratan untuk bisa mencapai tujuannya, maka dari itu digunakanlah model *two-wheeled self-balancing robot. Two-wheeled self-balancing robot* memiliki kemampuan untuk menjaga keseimbangan di atas dua roda dan mampu untuk berputar di tempat [6]. Model ini memiliki tingkat efisiensi yang lebih besar dibandingkan dengan robot keseimbangan yang lain, memiliki fitur fitur keamanan, kemudahan kontrol, dan kemampuan manuver yang tinggi [7], [8]. Penggunaan model *two-wheeled self-balancing robot* mampu melengkapi kebutuhan dari sebuah *autonomous mobile robot* dalam pergerakan melakukan pemetaan dan navigasi.

1.2 Rumusan Masalah

a. Bagaimana cara membuat robot keseimbangan beroda dua?
b. Bagaimana cara mengintegrasikan antara robot keseimbangan beroda dua dengan ROS ?
c. Bagaimana cara membuat sistem navigasi pada robot keseimbangan beroda dua ?

1.3 Tujuan Penelitian

a. Mengetahui cara membangun robot keseimbangan beroda dua
b. Mengetahui cara mengintegrasikan antara robot keseimbangan beroda dua dengan ROS
c. Mengetahui cara menggunakan sistem navigasi pada robot keseimbangan beroda dua.

1.4 Manfaat Penelitian

a. Penerapan teori dan simulasi pengontrolan ke dalam sebuah sistem yang nyata
b. Menghasilkan sebuah *Autonomous* robot keseimbangan beroda dua yang dapat dikembangkan untuk membantu pekerjaan manusia.

1.5 Batasan Masalah

Penelitian dibatasi dalama beberapa hal:

a. Pembangunan map dilakukan secara manual berdasarkan pengukuran ruangan milik peneliti pada dunia nyata
b. Kondisi uji coba robot pada dunia nyata sama dengan kondisi pada saat pembangunan map.
c. Represanti robot pada simulasi gazebo dan rviz tidak menggunakan feedback orientasi, kondisi robot dianggap seimbang di setiap waktu.

1.6 Sistematika Penulisan

Susunan dan pembahasan yang direncanakan pada penulisan Skripsi ini adalah sebagai berikut :

BAB I PENDAHULUAN

Bab ini menguraikan tentang latar belakang masalah, rumusan masalah, tujuan dan manfaat penelitian, serta sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Berisi referensi penelitian sebelumnya, menguraikan sistematis dasar teori dan komponen penelitian.

BAB III METODOLOGI PENELITIAN

Uraian metode , tahap tahap penelitian yang digambarkan melalui diagram alir penelitian dan variabel yang akan diteliti.

BAB IV HASIL DAN PEMBAHASAN

Menyajikan dan menjelasakan hasil temuan data yang dianalisis dari perancangan sistem yang telah dibuat.

BAB V KESIMPULAN DAN SARAN

Menyimpulkan dengan pernyataan singkat yang telah dijabarkan dari hasil penelitian serta merupakan jawaban dari tujuan penelitian dan memberikan saran untuk melanjutkan atau mengembangkan penelitian sejenis.

DAFTAR PUSTAKA

BAB II
TINJAUAN PUSTAKA

2.1 *Mobile Robot*

Robot merupakan sebuah alat mekanik yang digunakan untuk membantu pekerjaan manusia dalam berbagai bidang. Robot digunakan untuk pekerjaan sehari-hari hingga kebutuhan industri. Salah satu bidang utama dari ilmu robot adalah *mobile robot. Mobile robot* memiliki kemampuan untuk menavigasi dalam sebuah lingkungan dan berinteraksi dengannya menggunakan sensor dan aktuator [5], [9]. *Mobile robot* dapat diklasifikasikan menjadi dua yaitu *autonomous mobile robots* (AMR) dan *autonomous guided vehicles* (AGV). Perbedaan antara AMR dan AGV terletak pada mekanisme *autonomous* dari tiap robot. AGV bekerja berdasarkan petunjuk fisik dan menavigasi di lingkungan yang telah ditentukan pada jalur yang telah ditentukan. Sedangkan AMR mampu bekerja pada lingkungan yang belum terprediksi, mampu membuat model lingkungan, dan menentukan lokasinya sendiri [9], [10].

2.2 Penelitian Terdahulu

Terdapat beberapa penelitian terdahulu yang berkaitan dengan robot keseimbangan beroda dua, sistem navigasi, dan ROS(*robot operating system)*. Beberapa penelitan terdahulu seperti *The Kinematics Model of a Two-wheeled Self-balancing Autonomous Mobile Robot and Its Simulation* [6] dan *An Experimental Study on the PID and Fuzzy-PID Controllers on a Designed Two-Wheeled Self- Balancing Autonomous Robot* [7] memberikan gambaran yang cukup bagi penulis untuk membangun penelitian kali ini.

Tabel 2. 1 Tabel Penelitian Terdahulu

Nama Peneliti	**Judul**	**Keterangan**
He Bin, Liu Wen Zhen, Lv Hai Feng [6]	The Kinematics Model of a Two-wheeled Self-balancing Autonomous Mobile Robot and Its Simulation	Mendeskripsikan mengenai model kinematik dari *two-wheeled self-balancing robot.*

Rasoul Sadeghian & Mehdi Tale Masoule c	An Experimental Study on the PID and Fuzzy-PID Controllers on a Designed Two-Wheeled Self-Balancing Autonomous Robot	Menyajikan model dinamik dan strategi kontrol untuk *two-wheeled self-balancing robot.* Model dinamik dari *two-wheeled self-balancing robot* dihitung menggunakan metode newtonian dan strategi kontrol didesain berdasarkan perhitungan dari model dinamik.
Chaomin Luo, Gene Eu Jan, Jing Zhang, and Furao Shen [4]	Boundary Aware Navigation and Mapping for a Mobile Automaton	Mengusulkan metode *boundary aware* untuk melakukan navigasi dan pemetaan pada lingkungan yang tidak diketahui.
Li Zhi & Mei Xuesong [5]	Navigation and Control System of Mobile Robot Based on ROS	Mengembangkan seluruh sistem kontrol pada *mobile robot.* Menggunakan ROS sebagai *upper computer* yang dihubungkan ke sistem kontrol melalui USB.
Joko Slamet Saputro, Pranoto H. Rusmin, & Arief Syaichu Rochman [11]	Design and Implementation of Trajectory Tracking Motion in Mobile Robot Skid Steering Using Model Predictive Control	Menjelaskan mengenai desain dan implementasi perencanaan gerak dan pelacakan lintasan *mobile robot.*
Sukkpranhachai Gatesichapakorn, Jun Takamatsu, Miti Ruchanurucks [2]	ROS based Autonomous Mobile Robot Navigation using 2D LiDAR and RGB-D Camera	Mengimplementasikan *autonomous mobile robot* dengan menggunakan *robot operating system* (ROS). Membandingkan antara penggunaan LiDAR dengan gabungan antara

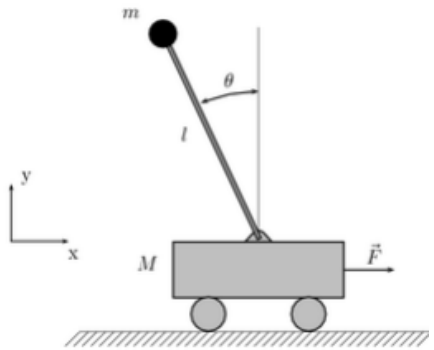
		LiDAR dan RGB-D kamera.

Pada penelitian yang dilakukan, digunakan robot keseimbangan beroda dua sebagai *mobile robot* untuk melakukan navigasi secara otonom. Penelitian terdahulu, penggunaan *mobile robot* juga menggunakan dua roda. Namun, dibantu dengan adanya *caster wheel*. Penggunaan motor stepper juga menjadi pembeda dalam penelitian yang dilakukan.

2.3 Robot Keseimbangan Beroda Dua

Robot keseimbangan beroda dua merupakan *mobile robot* yang menggunakan dua roda di sisi kiri dan kanan. Sebagai robot keseimbangan beroda dua, robot memiliki ciri khas berupa kemampuan untuk menyeimbangkan robot pada dua roda dan dapat berputar di tempat [6].

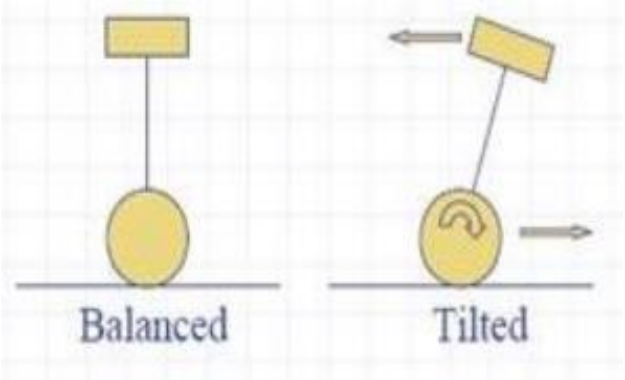
Robot keseimbangan beroda dua menggunakan konsep pendulum terbalik. Pendulum terbalik merupakan pendulum yang terengsel ke kereta beroda yang dapat bergerak di sepanjang lintasan horisontal seperti yang terlihat pada gambar 2.1. Dengan konsep tersebut, dapat diasumsikan dua roda sebagai kereta beroda dan badan robot sebgai pendulum. Konsep ini memiliki sistem yang tidak stabil dimana pendulum tidak dapat terus seimbang karena adanya gravitasi bumi, sehingga diperlukan sebuah kendali khusus untuk mempertahankan keseimbangan.



Gambar 2. 1 Pendulum Terbalik

Sumber: http://www.oocities.org/husni_ums/dsk/bandulterbalik.html

Dasar untuk mengendalikan robot agar tetap seimbang adalah dengan menggerakan roda searah dengan arah jatuhnya badan robot. Gambaran cara kerja robot menjaga keseimbangan dapat dilihat pada gambar 2.2.



Gambar 2. 2 Robot Menjaga Keseimbangan

Sumber: http://www.instructables.com/id/Self-Balancing Robot/

2.4 ROS

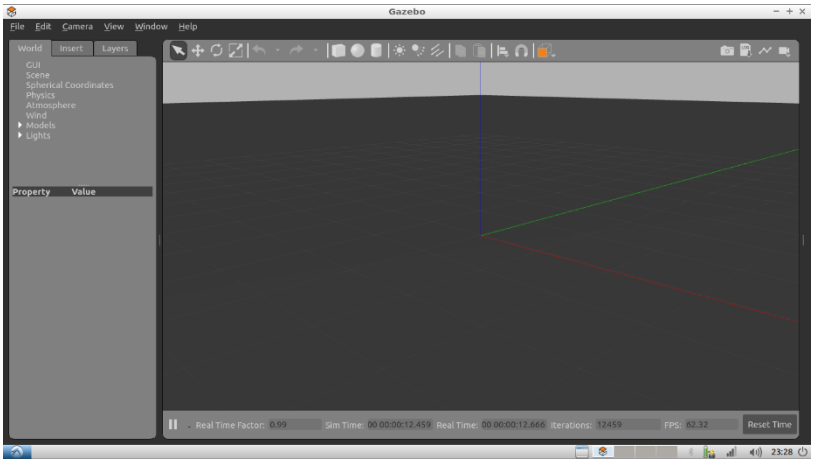
ROS (*Robot Operating System)* merupakan platform yang digunakan untuk berbagai implementasi robot. ROS menjadi sebuah framework yang fleksibel dengan berbagai *tools, library,* dan *conventions.* Lebih jauh lagi, ROS menyediakan API untuk membangun *custom packages* untuk digunakan bersama dengan sistem eksternal [2].

Penggunaan ROS secara umum menggunakan tiga konsep berikut: *filesystem level, computational graph level,* dan *community level* [1]. *Filesystem level* mencakup sumber daya ROS yang ada di *disk* seperti: *packages, metapackages, dan repostories. Computational graph level* merupakan jaringan *peer-to-peer* dari proses ROS yang memproses data bersama. *Community level* menjadi sumber daya ROS yang memungkinkan komunitas terpisah untuk bertukar perangkat lunak dan pengetahuan.

2.5 Gazebo

Gazebo merupakan aplikasi yang digunakan untuk melakukan simulasi. Gazebo bersifat open source dan dapat mengolah perkembangan mesin dinamis. Aplikasi gazebo berjalan pada sistem operasi linux dan memiliki kemampuan grafis yang cukup tinggi [12].

Gazebo menawarkan kemampuan untuk mensimulasikan populasi robot secara akurat dan efisien di lingkungan indoor dan outdoor yang kompleks. Selain itu gazebo memiliki komunitas yang hidup dan berbagai fitur yang menunjang simulasi robot. Gambar 2.3 menunjukan aplikasi gazebo dengan *empty world.*

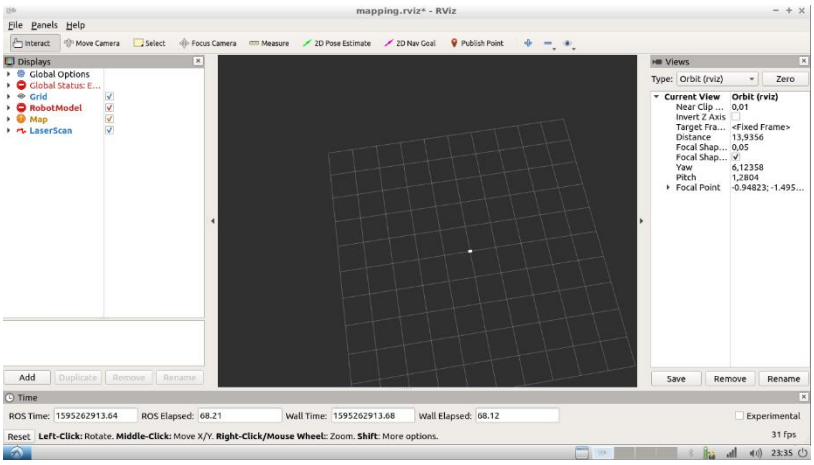


Gambar 2. 3 Gazebo

2.6 RVIZ

RVIZ adalah grafis antarmuka ROS yang memungkinkan untuk memvisualisasikan banyak informasi, menggunakan plugin untuk berbagai jenis topik yang tersedia. RVIZ dapat memvisualisasikan sensor data di lingkungan 3D, memperbaiki kinetik model dari gazebo, dan merepresentasikan data sensor seperti LiDAR [13].

Dalam RVIZ, dapat dilakukan pemilihan berbagai bentuk tampilan dan sudut pandang. Dengan menekan tombol penambahan pada RVIZ, dapat dilakukan visualisasi berbagai data dari berbagai sensor. Gambar 2.4 menunjukan *interface* dari RVIZ.

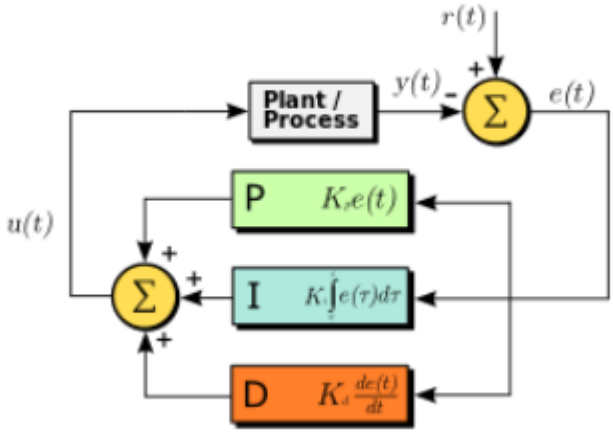


Gambar 2. 4 RVIZ

2.7 PID *Controller*

PID *controller* merupakan metode kontrol yang mengombinasikan proporsional, integral, dan derivatif untuk menghasilkan sinyal kontrol. PID

controller bekerja sebagai pemberi *feedback* (umpan balik) untuk bisa mencapai suatu level yang diinginkan. Skema PID ditunjukkan pada Gambar 2.5.



Gambar 2. 5 Skema PID

Sumber: http://eupdate.weebly.com/

PID *controller* mempertahankan output sedimikian hingga tidak ada kesalahan antara variabel dengan output setpoint yang diinginkan melalui closed loop operations.

2.8 Navigasi

Algoritma navigasi robot yang diketahui saat ini terbagi menjadi tiga bagian, berdasarkan pengetahuan akan lingkungan, yaitu: *completely known, partially known* dan *unknown environment.* Pada lingkungan *completely known,* cukup mudah untuk melakukan konfigurasi parameter dengan membuat peta dan melakukan algoritma pencarian untuk menghasilkan referensi jalur. Di sisi lain, *partially known* atau *unknown environment*, membutuhkan algoritma untuk menghindari rintangan [14].

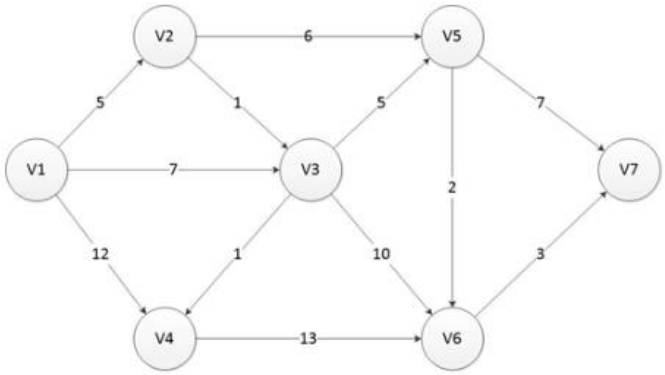
2.9 Algoritma Dijkstra

Algoritma Dijkstra merupakan sebuah algoritma untuk memecahkan masalah pencarian jarak terpendek pada graf. Algoritma Dijkstra bekerja dengan membuat jalur ke satu simpul optimal pada setiap langkah. Jadi pada langkah ke n, setidaknya ada n *node* yang sudah kita tahu jalur terpendek [15].

Langkah-langkah algoritma Dijkstra dapat dilakukan dengan langkah-langkah berikut:

a. Tentukan titik mana yang akan menjadi *node* awal, lalu beri bobot jarak pada *node* pertama ke *node* terdekat satu per satu, Dijkstra akan melakukan pengembangan pencarian dari satu titik ke titik lain dan ke titik selanjutnya tahap demi tahap.
b. Beri nilai bobot (jarak) untuk setiap titik ke titik lainnya, lalu set nilai 0 pada *node* awal dan nilai tak hingga terhadap *node* lain (belum terisi) 2.
c. Set semua *node* yang belum dilalui dan set *node* awal sebagai "*Node* keberangkatan"
d. Dari *node* keberangkatan, pertimbangkan *node* tetangga yang belum dilalui dan hitung jaraknya dari titik keberangkatan. Jika jarak ini lebih kecil dari jarak sebelumnya (yang telah terekam sebelumnya) hapus data lama, simpan ulang data jarak dengan jarak yang baru
e. Saat kita selesai mempertimbangkan setiap jarak terhadap *node* tetangga, tandai *node* yang telah dilalui sebagai "*Node* dilewati". *Node* yang dilewati tidak akan pernah di cek kembali, jarak yang disimpan adalah jarak terakhir dan yang paling minimal bobotnya.
f. Set "*Node* belum dilewati" dengan jarak terkecil (dari *node* keberangkatan) sebagai "*Node* Keberangkatan" selanjutnya dan ulangi langkah e.

Contoh perhitungan dapat dilihat pada gambar 2.6 dan 2.7.



Gambar 2. 6 Contoh Jalur

Sumber: https://mti.binus.ac.id/2017/11/28/algoritma-dijkstra/

Iteration	Unvisited (Q)	Visited (S)	Current	Node : Min = (dist[node], prev[node])iteration						
				V1	V2	V3	V4	V5	V6	V7
	Initialization {V1,V2,V3,V4,V5,V6,V7}	{-}		(0, -)0	(∞, -)0	(∞, -)0	(∞, -)0	(∞, -)0	(∞, -)0	(∞, -)0
1	{V2,V3,V4,V5,V6,V7}	{V1}	V1		(5,V1)1	(7,V1)1	(12,V1)1	(∞,V1)1	(∞,V1)1	(∞,V1)1
2	{V3,V4,V5,V6,V7}	{V1,V2}	V2			(6,V2)2	(12,V1)1	(11,V2)2	(∞,V2)2	(∞,V2)2
3	{V4,V5,V6,V7}	{V1,V2, V3}	V3				(7,V3)3	(11,V3)3	(16,V3)3	(∞,V3)3
4	{V5,V6,V7}	{V1,V2, V3,V4}	V4					(11,V3)3	(16,V3)3	(∞,V3)3
5	{V6,V7}	{V1,V2, V3,V4,V5}	V5						(13,V5)5	(18,V5)5
6	{V7}	{V1,V2, V3,V4,V5.V6}	V6							(16,V6)6

Gambar 2. 7 Hasil Perhitungan Jalur

Sumber: https://mti.binus.ac.id/2017/11/28/algoritma-dijkstra/

2.10 AMCL (*adaptive Monte Carlo localization)*

ACML merupakan metode lokalisasi berdasarkan Monte Carlo. Metode ini dapat direalisasikan melalui rata-rata filter partikel. Ide utama dari penyaringan partikel adalah untuk menyebarkan partikel secara acak di ruang peta. Ketika robot bergerak maka partikel juga ikut bergerak. Dalam setiap periode perhitungan, semakin tinggi tingkat kecocokan data laser pemindai dari posisi ke peta lingkungan, partikel dalam posisi akan memperoleh bobot yang lebih tinggi. Partikel dalam peta akan dilakukan sampling ulang berdasarkan bobot, partikel-partikel pada peta, secara bertahap akan bertemu pada satu titik, maka ini adalah estimasi kemungkinan maksimum dari posisi robot [5].

2.11 Arduino Uno Rev3

Arduino Uno merupakan mikrokontroler berbasis ATmega328P. Memiliki 14 pin digital input/output, 6 analog input, 16MHz *ceramic resonator* (CSTCE16M0V53-R0), koneksi USB, sebuah *power jack*, dan tombol reset. Spesifikasi lebih lengkap dapat dilihat pada tabel 2.2:

Tabel 2. 2 Spesifikasi Arduino Uno

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)

PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g

2.12 ESP8266-01

ESP8266 merupakan mikrokontroler yang memiliki fitur *wifi built-in.* Sebagai mikrokontroler, ESP8266 mampu melakukan kontrol secara terbatas karena hanya memiliki empat pin input/output, dimana dua diantarnya telah digunakan sebagai sarana komunikasi, satu digunakan sebagai trigger untuk melakukan flashing. ESP8266 lebih cocok digunakan sebagai shield yang dipadukan dengan mikrokontroler lain seperti arduino agar mikrokontroler yang tersambung dapat terkoneksi dengan jaringan wifi..

2.13 MPU6050 *Accelerometer* dan *Gyroscope*

MPU6050 merupakan piranti yang menggunakan teknologi MEMS dan *Coriolis Effect* untuk melakukan pengukuran. Memiki sensor 3-axis *accelerometer* dan 3-axis *gyroscope* yang tertanam dalam satu chip. Output dari *accelerometer* berupa akselerasi gravitasi di sepanjang 3-axis dan output dari *gyroscope* berupa `*position.* MPU6050 bekerja pada range tegangan 5V dan menggunakan protokol I2C untuk komunikasi data. MPU 6050 memberikan keluaran berupa akselerasi dan keseimbangan dari robot secara *real time.* Sudut dari robot dapat diperoleh dari *accelerometer* dengan menghitung sudut dari vektor gravitasi dengan menggunakan fungsi trigonometri [16], yang kemudian hasil perhitungan tersebut diproses oleh mikrokontroler untuk digunakan untuk mengatur keseimbangan dari robot.

2.14 Stepper Driver DRV8825

Merupakan driver stepper motor bipolar yang memiliki fitur *adjustable current limiting,* proteksi kelebihan arus, proteksi kelebihan temperatur, dan memiliki enam *microsteps resolution (up to 1/32).* Bekerja pada range tegangan 8,2V – 45V dan kemampuan menyalurkan arus mencapai 1,5A. Driver stepper digunakan untuk menggerakan motor stepper dan sebagai perantara kontrol antara mikrokontroler ke motor stepper. Hal ini dikarenakan mikrokontroler hanya bekerja pada arus rendah (100mA), sehingga dibutuhkannya perantara untuk mengontrol motor stepper yang membutuhkan arus lebih besar (1,7A). Penggunaan driver DRV8825 membutuhkan penambahan komponen kapasitor yang dipasang di inputan Vmot dan GND. Gambar 2.8 menunjukan driver DRV8825 dengan kapasitor.

Gambar 2. 8 Stepper Driver DRV8825

2.15 Stepper Motor

Motor stepper adalah perangkat elektromekanis yang bekerja dengan mengubah pulsa elektronis menjadi gerakan mekanis diskrit. Motor stepper bergerak berdasarkan urutan pulsa yang diberikan kepada motor. Karena itu, untuk menggerakkannya diperlukan pengendali motor stepper yang membangkitkan pulsa-pulsa periodik. Penggunaan motor stepper memiliki beberapa keunggulan dibandingkan dengan penggunaan motor DC biasa, seperti : Sudut rotasi motor proporsional dengan pulsa masukan sehingga lebih mudah diatur, Motor dapat langsung memberikan torsi penuh pada saat mulai bergerak Posisi dan pergerakan repetisinya dapat ditentukan secara presisi memiliki respon yang sangat baik terhadap mulai, stop dan berbalik. Penggunaan motor stepper juga mampu mengeliminasi penggunaan komponen yang mahal seperti *optical encoder* dan *brushless motor DC* [17].

Dengan tingkat kepresisian yang tinggi pada stepper motor. Maka stepper motor akan digunakan sebagai roda utama atau bagian robot yang menyentuh dengan tanah, untuk menggerakan robot. Penggunaan stepper yang memiliki tingkat presisi tinggi akan memberikan kemudahan bagi robot mempertahankan keseimbangan pada sumbu lateral.

BAB III

METODOLOGI PENELITIAN

3.1 Waktu dan Tempat Penelitian

Penelitian telah dilaksanakan dalam waktu 5 bulan terhitung dari bulan Februari 2020 hingga Juni 2020. Lokasi penelitian berada di rumah peneliti.

3.2 Alat dan Bahan

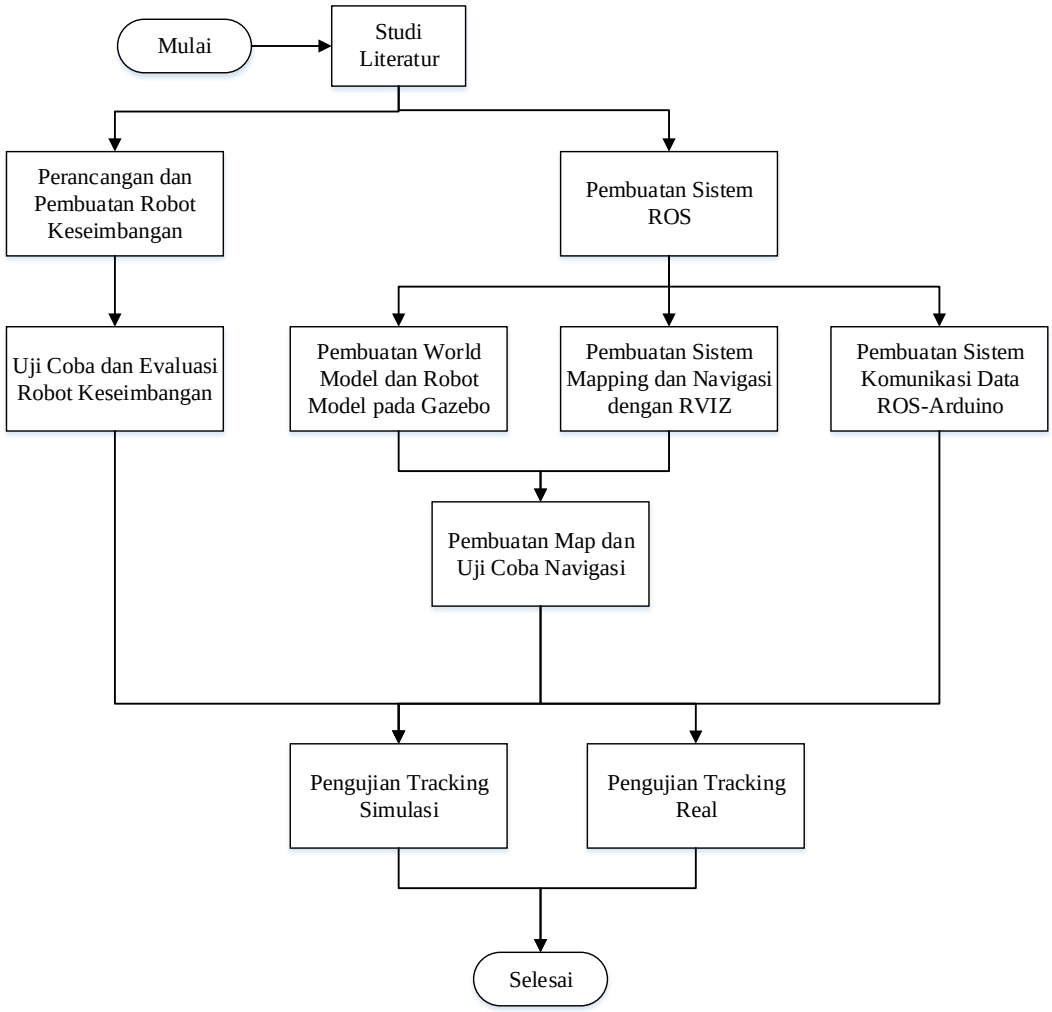
Tabel 3. 1 Tabel Alat dan Bahan

No	Nama alat	Kuantitas	Keterangan
1	Arduino Uno Rev3	1 Buah	Mikrokontroler sebagai pengendali robot
2	MPU6050	1 Buah	Sensor untuk mengetahui sudut
3	Stepper Driver DRV8825	1 Buah	Driver kontrol perantara antara mikrokontroler dengan motor stepper
4	Stepper Motor	1 Buah	Sebagai penggerak robot
5	Solder	1 buah	Penyolder
6	Tenol	1 buah	Timah untuk mengubungkan kompenen atau kabel
7	Multimeter	1 buah	Menguji arus dan tegangan pada sistem

8	Baterai LiPO	1 buah	Penyuplai tegangan 12V
9	Toolset	1 buah	Obeng berbagai ukuran
10	PCB Polos Fiber 10.5x20cm	1 buah	Meletakkan rangkaian driver yang sudah jadi
11	Pin Header Male dan Pin Header Female	10 buah	Penyambung kabel jumper agar terpasang dengan baik
12	ESP8266	1 Buah	Mikrokontroler *built-in* wifi
13	Akrilik	2 buah	Sebagai casing rangka untuk meletakan komponen
14	Roda	2 Buah	Pijakan robot
15	Spacer	10 Set	Menghubungkan antar rangka
16	Laptop Lenovo IdeaPad 320-14AST	1 Buah	Digunakan untuk instalasi ROS

3.3 Metode Penelitian

Pada penulisan skripsi, digunakan beberapa metode untuk mendapatkan data objektif yang dapat dijadikan acuan dalam penyusunan skripsi. Gambar 3.1 menunjukkan metode penelitian dalam bentuk diagram alir.



Gambar 3. 1 Diagram Metode Penelitian

3.3.1 Studi Literatur

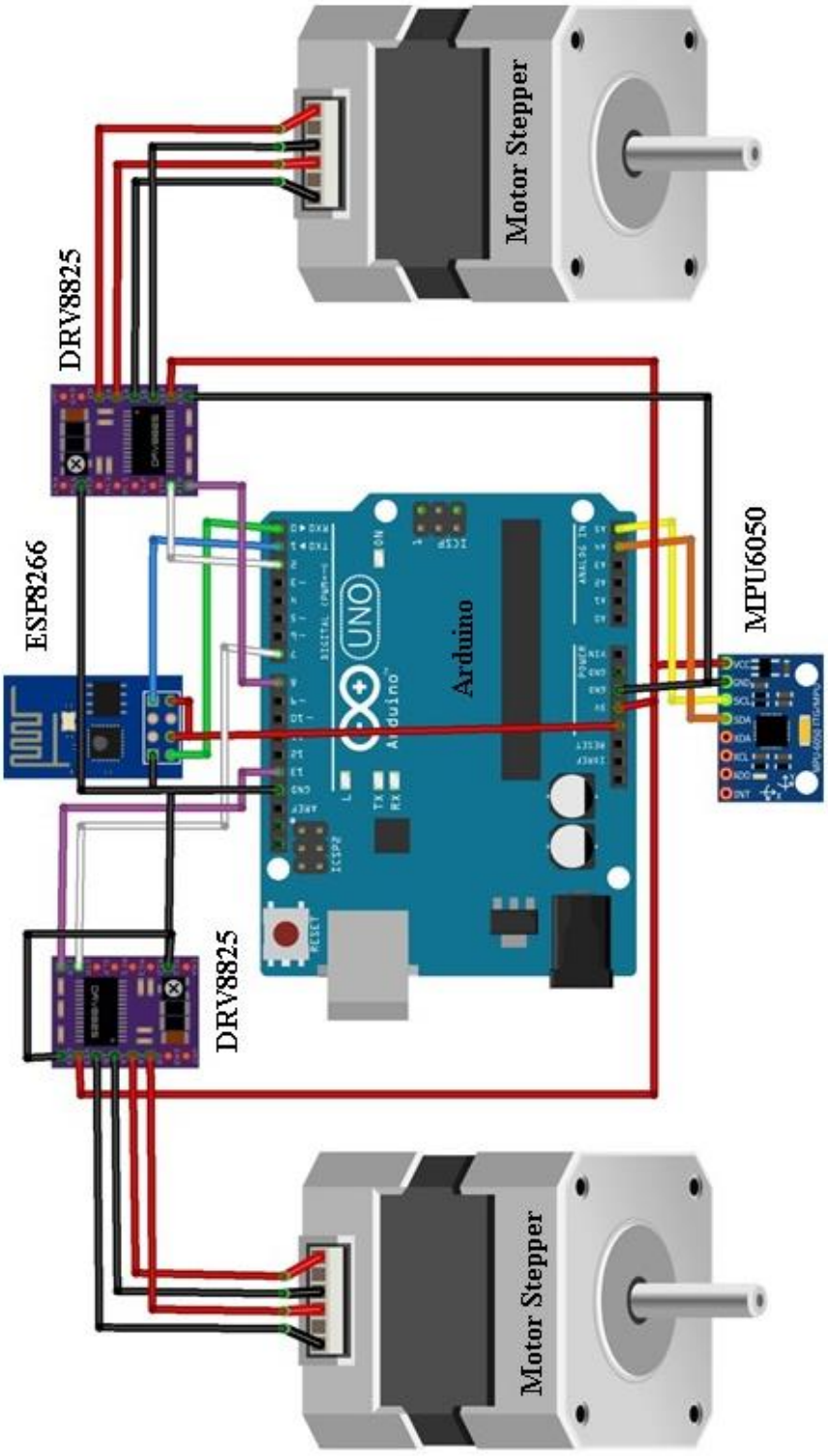
Setelah mengidentifikasi masalah, dilakukan pencarian sumber-sumber referensi berupa jurnal, *paper*, artikel, dan sumber dalam bentuk lain yang berkaitan dengan robot keseimbangan, ROS, Gazebo, RVIZ, sistem navigasi, dan sistem otonom guna memahami dan mengerti bagaimana merancang sistem navigasi otonom pada robot keseimbangan.

3.3.2 Perancangan dan Pembuatan Robot Keseimbangan

a. Desain Casing dan Perancangan Robot

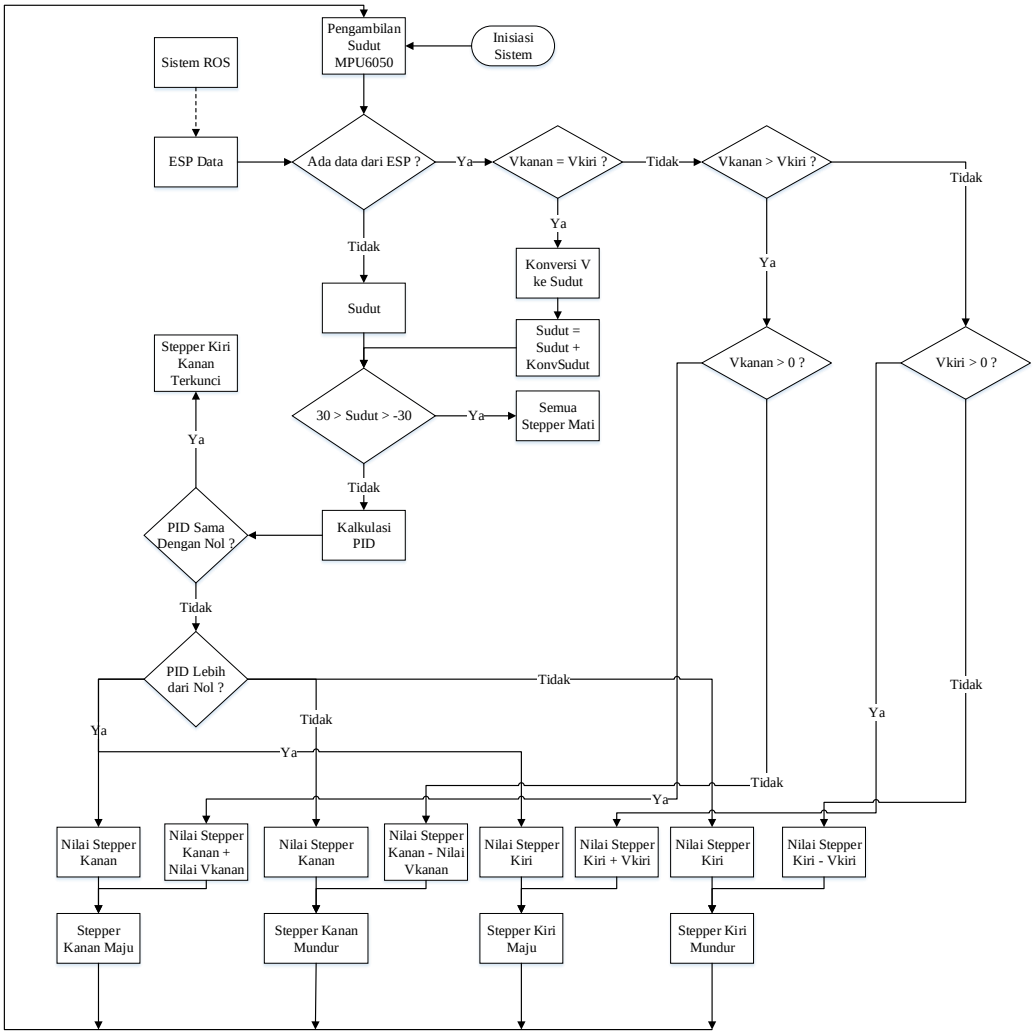
Pembuatan desain casing menggunakan software corel draw. Bahan yang digunakan berupa akrilik bening dengan tebal 2mm. Pemotongan akrilik menggunakan jasa laser cutting. Casing dibuat dari tumpukan layer akrilik yang dipisahkan dengan spacer berukuran 2.5cm. Jumlah layer yang dirancang berjumlah tiga.

Casing didesain agar peletakan antara komponen dan baterai dapat diatur dengan mudah. Untuk perancangan robot digunakan skema rangkaian komponen seperti yang digambarkan pada gambar 3.2.



Gambar 3. 2 Skema Rangkaian Robot Keseimbangan

Kontrol keseimbangan menggunakan algoritma seperti yang terlihat pada gambar 3.3. Algoritma dimulai dengan inisiasi sistem, selanjutnya dilakukan proses pengulangan pengambilan sudut, pemeriksaan data, kalkulasi PID, dan penggerakan motor stepper. Program akan terus mengulang sampai sistem dimatikan.



Gambar 3. 3 Flowchart Cara Kerja Robot Keseimbangan

b. Konfigurasi MPU6050 dengan Arduino Uno

Konfigurasi MPU6050 menggunakan library wire.h yang tersedia di arduino. MPU6050 *gyroscope output* diatur pada skala +/- 250 derajat per detik. MPU6050 *accelerometer output* diatur pada skala +/- 4g.

c. Konfigurasi Driver Stepper DRV8825

Penggunaan driver dengan menyambungkan pin pulse dan pin *direction* pada driver ke output pulse pada arduino. Dengan pemberian

pulsa high dan low dari arduino, maka akan memberikan sinyal kepada driver untuk menggerakan motor stepper. Pemberian sinyal high atau low pada pin *direction* dapat mengubah arah putaran motor stepper.

Dilakukan pengaturan pembatasan arus pada driver, untuk mencegah arus yang melebihi kapasitas motor stepper.

d. Konfigurasi Motor Stepper Nema 17 dengan Arduino Uno

Untuk menggerakan motor stepper diperlukan sinyal high low yang dikirimkan ke driver. Pada arduino sinyal high low dapat dihasilkan melalui beberapa metode seperti: bit banging, PWM, dan timer intterupt.

e. PID Kontroler dalam Arduino

Penggunaan PID kontroler bertujuan untuk memberikan respon yang cepat pada robot untuk kembali menyeimbangkan diri setelah terjadinya *tilted*. Beberapa library PID telah disediakan oleh software arduino. Pemakaian perlu memasukan file library, melakukan deklarasi variabel, dan pemanggilan fungsi.

f. ESP8266

ESP8266 berperan untuk melakukan komunikasi data antara robot dengan sistem ROS melalui wifi. Antara ESP8266 dan arduino sebagai pengontrol robot perlu dibangun komunikasi. Komunikasi serial menjadi pilihan yang tepat untuk melakukan pertukaran data antara ESP8266 dengan arduino. Pada ESP8266 hanya perlu melakukan inisiasi serial dan mengirimkan setiap data yang diterima dari sistem ROS. Pada arduino melakukan inisiasi serial pada baud rate yang sama kemudian melakukan penerimaan data.

g. Uji Coba dan Evaluasi Robot keseimbangan

Robot keseimbangan yang telah dibangun, dilakukan uji coba apakah robot mampu mempertahankan keseimbangan dengan baik. Pada robot dilakukan kalibrasi MPU6050, pengaturan nilai konstanta PID, dan uji coba robot untuk bergerak. Hasil yang didapat akan dilakukan evaluasi, untuk menghasilkan hasil yang terbaik.

3.4 Pembuatan Sistem ROS

Pembuatan sistem ROS perlu dilakukan di *operating system* berbasis linux, untuk itu diperlukan instalasi *operating system linux* pada laptop. Untuk dapat menjalankan ROS, dilakukan instalasi ROS, package, tools, dan library.

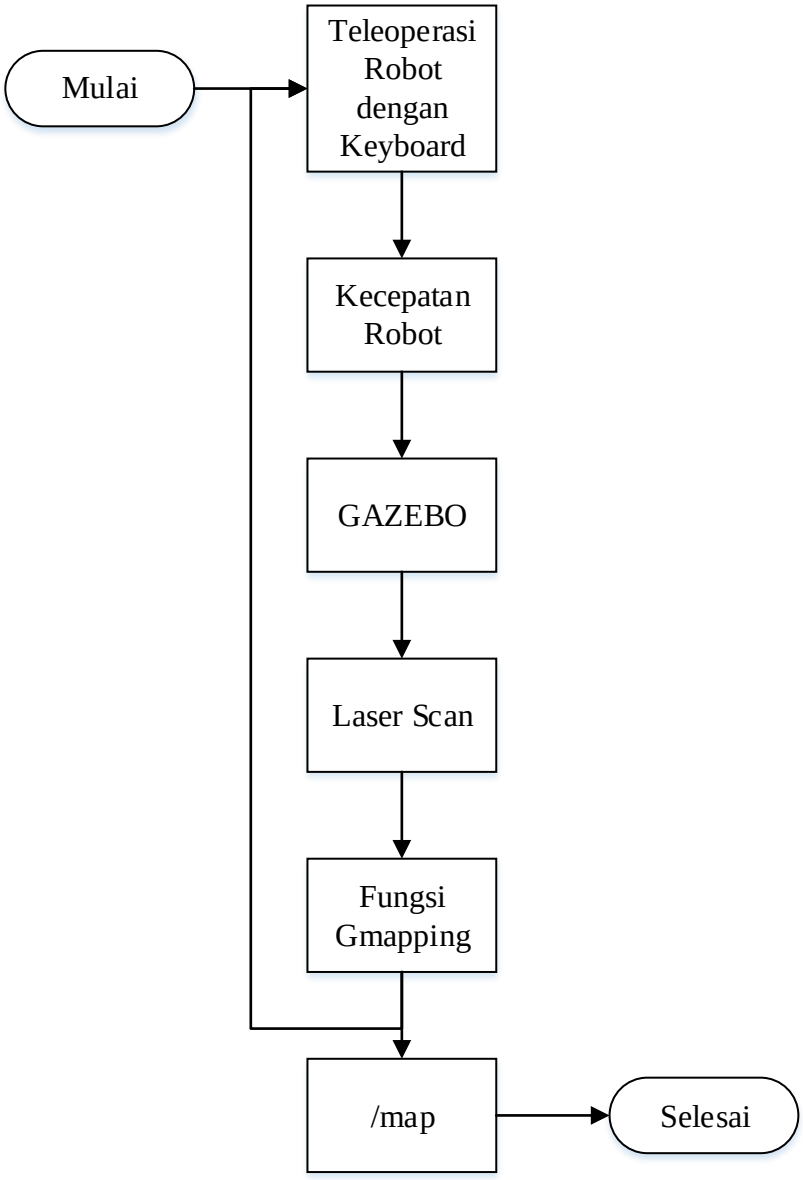
3.4.1 Pembuatan World model dan Robot Model pada Gazebo

Pembuatan *world model* dan *robot model* pada gazebo diperlukan untuk menjankan navigasi pada robot. *World model* dibuat sesuai dengan model ruangan yang ada pada dunia nyata, begitu pula dengan *robot model*, disesuaikan dengan ukuran dan kondisi dari robot keseimbangan.

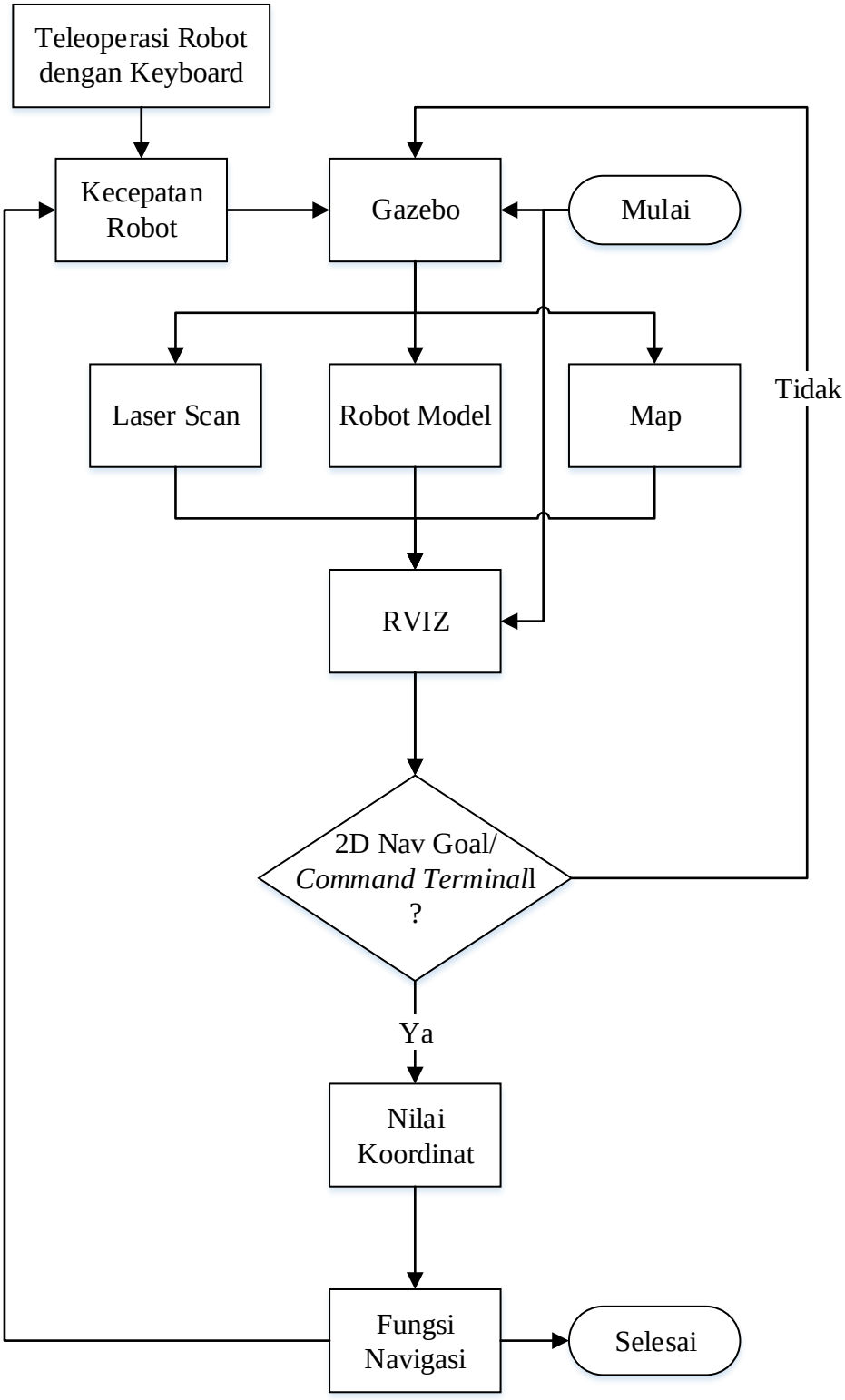
3.4.2 Pembuatan Sistem Mapping dan Navigasi dengan RVIZ

ROS menyediakan *package* yang dapat digunakan untuk melakukan mapping dan navigasi. Untuk melakukan mapping digunakan *gmapping package*, yang menyediakan SLAM (*Simultaneous Localization and Mapping*). Proses pembuatan map dapat dilihat pada gambar 3.4. ROS juga menyediakan *navigation package*, salah satunya adalah amcl, dimana penggunaannya mengimplementasikan pendekatan lokalisasi Monte Carlo adaptif (atau KLD-sampling), yang menggunakan filter partikel untuk melacak pose robot terhadap peta yang diketahui.

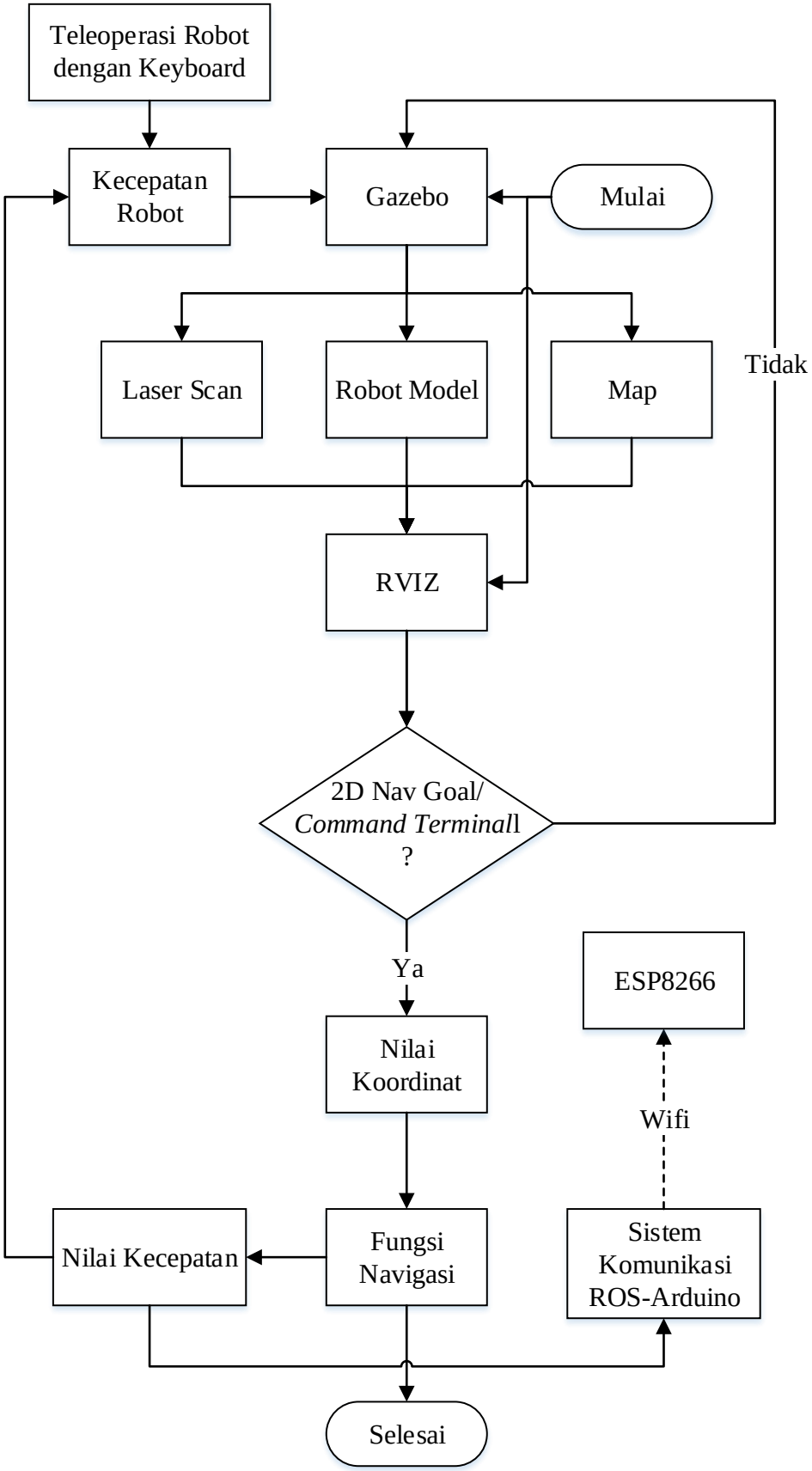
Sistem navigasi yang akan dilakukan terbagi menjadi dua bagian. Pertama navigasi secara simulasi dalam GAZEBO. Data data dari Gazebo berupa model robot, laser scan, dan dunia simulasi dikirim dan divisualisasikan dengan RVIZ. RVIZ menerima data data tersebut dan digunakan untuk melakukan fungsi navigasi, flowchart navigasi simulasi dapat dilihat pada gambar 3.5. Navigasi yang kedua digunakan simulasi GAZEBO dan robot keseimbangan yang telah dibuat di dunia nyata. Data data dari Gazebo berupa model robot, laser scan, dan dunia simulasi dikirim dan divisualisasikan dengan RVIZ. Data data tersebut digunakan untuk melakukan navigasi, hasil dari navigasi berupa data kecepatan dikirimkan ke simulasi GAZEBO dan ke robot keseimbangan melalui sistem komunikasi ROS-Arduino dengan jaringan wifi. Proses navigasi real dapat dilihat pada gambar 3.6.



Gambar 3. 4 Flowchart Pembuatan Map



Gambar 3. 5 Flowchart Navigasi Simulasi



Gambar 3. 6 Flowchart Navigasi Real

3.4.3 Pembuatan Sistem Komunikasi Data ROS-Arduino

Komunikasi data ROS-arduino dilakukan secara *wireless* melalui wifi. ROS menyediakan *package* bernama rosserial_server, dimana *package* tersebut membuat laptop menjadi server atau master. Client hanya perlu melakukan koneksi dengan IP address dari master dan menghubungkan port yang sesuai.

Sistem komunikasi ini juga bertujuan untuk menyelaraskan putaran roda dari robot keseimbangan dengan visualisasi pada Gazebo dan RVIZ. Untuk itu perlu adanya konversi data antara data dari sistem navigasi dan robot keseimbangan.

3.5 Pengujian *Tracking* Simulasi

Pengujian dilakukan dengan mengirimkan perintah koordinat dan orientasi pada sistem navigasi di RVIZ. Pengujian dilakukan pada map yang telah dibuat berdasarkan salah satu ruangan milik penulis. Sistem navigasi RVIZ mendapatkan input dari GAZEBO berupa simulasi real world, laser scan, dan model robot.

3.6 Pengujian *Tracking* Real

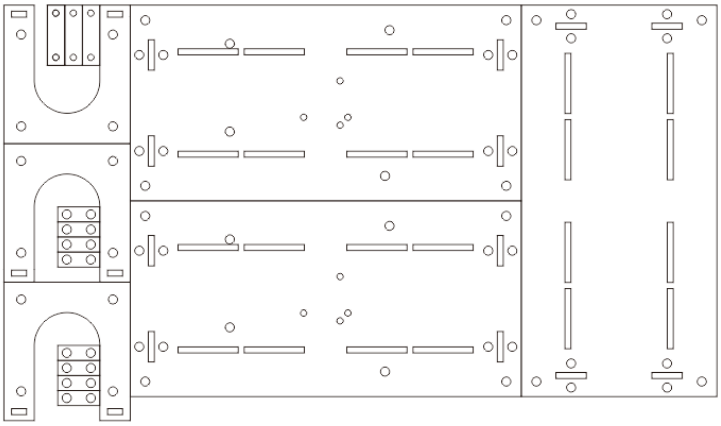
Pengujian *Tracking* real dilakukan sama seperti *Tracking* pada simulasi, namun data kecepatan dikirimkan ke robot keseimbangan di dunia nyata. Robot keseimbangan bergerak berdasarkan navigasi RVIZ, dengan data data dari simulasi real world pada GAZEBO.

BAB IV
HASIL DAN ANALISIS

4.1 Perancangan dan Pembuatan Robot Keseimbangan

a. Desain Casing dan Perancangan Robot

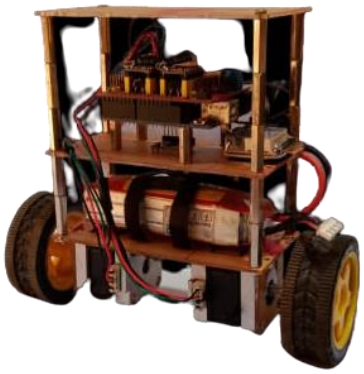
Pada pembuatan robot keseimbangan digunakan casing akrilik dan spacer. Casing tersusun dari 3 layer akrilik berukuran 13cm x 6.5cm, dimana setiap layernya dipisahkan oleh 3 spacer berukuran 2.5cm. Desain casing dibuat menggunakan software corel draw. Gambar 4.1 menunjukkan hasil desain casing robot keseimbangan yang telah dibuat.



Gambar 4. 1 Desain Casing Robot Keseimbangan

Peletakan komponen diatur sedemikian rupa sehingga robot keseimbangan tidak memiliki terlalu banyak selisih berat pada kedua sisi. Komponen arduino, stepper driver, dan MPU6050 diletakan pada bagian tengah layer kedua. Hal ini dilakukan untuk mengurangi terjadinya *drifting* pada saat robot berputar karena tidak tegak lurusnya sensor MPU6050 dengan pusat massa dari robot. Hal ini juga memberikan respon yang pas pada MPU6050 saat melakukan pengukuran sudut, tidak terlalu cepat (dekat dengan pusat putaran roda) atau tidak terlalu lambat (jauh dari pusat putaran roda). Baterai diletakan pada bagian bawah, karena tempat yang tersisa adalah layer ketiga (paling atas) atau layer pertama (paling bawah). Baterai memiliki massa yang cukup besar dibanding dengan komponen lain, sehingga diletakan di layer pertama.

Hal ini dimaksudkan untuk mengurangi beban pada stepper saat menyeimbangkan robot. Hasil dari desain casing dan perancangan robot dapat dilihat pada gambar 4.2.



Gambar 4. 2 Robot Keseimbangan

b. Konfigurasi MPU6050 dengan Arduino Uno

MPU6050 merupakan sensor yang dapat melakukan pengukuran 3-*Axis accelerometer* dan 3-*Axis gyroscope.* Data dari *gyroscope* berupa kecepatan rotasi sumbu x, y, dan z dalam bentuk derajat per detik. Data dari *accelerometer* berupa percepatan gravitasi di sumbu x, y, dan z.

Penggunaan MPU6050 dibutuhkan library wire.h untuk melakukan komunikasi I2C. Diperlukan beberapa deklarasi berupa: I2C address dari MPU6050 dan nilai kalibrasi *accelerometer*.

Pada *setup function* dilakukan pemanggilan beberapa fungsi yang didapat dari library wire.h, digunakan untuk menentukan data mana saja yang akan diambil dari register sensor MPU6050. Pada setup function dilakukan konfigurasi pengambilan data dengan frekuensi 250Hz dan I2C *clock speed* sebesar 400KHz. Dilakukan juga kalibrasi gyro sebanyak 500 data dengan selang waktu 3700 mikrodetik.

Pada *loop function* dilakukan pengambilan data *gyroscope* dan *accelerometer* dengan frekuensi 250Hz sesuai dengan kecepatan yang telah ditetapkan pada setup *function.*

Penggunaan data baik *gyroscope* dan *accelerometer* masing masing memiliki kelemahan dan kelebihan masing masing. Data *gyroscope* sangat mudah mengalami *drifting* dimana data akan berubah seiring dengan perubahan waktu. Sedangkan pada *accelerometer* sangat rentan

dengan adanya getaran, dimana ketika sensor mengalami getaran, keluaran dari *accelerometer* akan mengalami osilasi yang cukup tinggi. Untuk menangani kelemahan disetiap sensor, maka digunakan komplemen filter, dimana dengan komplemen filter, nilai hasil dari *gyroscope* akan dikompensasi dengan nilai dari *accelerometer*.

c. Konfigurasi Driver Stepper DRV8825

Driver stepper DRV8825 mampu bekerja pada motor stepper dengan arus mencapai 2.2A. Spesifikasi tersebut cukup untuk mengontrol motor stepper nema 17 dengan arus 1.7A. Pada driver dilakukan konfigurasi pembatasan arus dengan memutar potensiometer yang ada pada driver. Hal ini dimaksudkan agar tidak terjadi *over current* pada motor stepper yang menyebabkan kerusakan. Driver memiliki fitur *microstepping,* dimana step yang dimiliki stepper nema 17 dapat dibagi lagi sesuai dengan nilai dari *microstepping.* Pada penelitian digunakan *microstepping* ¼, yang membuat step default nema 17(200) menjadi 800 step untuk mencapai satu putaran penuh.

d. Konfigurasi Motor Stepper Nema 17 dengan Arduino Uno

Stepper motor dapat dikontrol dengan menggunakan perubahan pulsa antara high dan low lewat driver. Perubahan pulsa ini terjadi dalam kecepatan yang tinggi. Untuk itu digunakanlah fungsi timer pada Arduino Uno. Pemberian inputan high atau low pada pin *direction* dapat mengubah arah putaran motor stepper.

Pada *setup function* dilakukan pengaturan timer, dimana timer akan berjalan dengan kecepatan 20 mikrosekon.

Pada *loop function* dilakukan konversi hasil perhitungan PID kontroler menjadi nilai *throttle* yang digunakan sebagai inputan pada fungsi timer untuk mengontrol motor stepper.

e. PID Kontroler dalam Arduino

Penggunaan PID membutuhkan beberapa deklarasi awal seperti: setpoint, nilai kp, kd, ki, input, dan output. Penggunaan PID dengan pengaturan nilai konstanta yang tepat, memberikan respon yang cepat untuk robot menyeimbangkan diri.

Pembuatan kode program PID pada arduino cukup sederhana, hal ini menjadikan PID dipilih oleh peneliti. Kode yang dibutuhkan juga sederhana sehingga tidak teralalu membebani kinerja mikrokontroler dan tidak mengganggu proses kode yang lain.

f. ESP8266

ESP8266 bekerja sebagai penghubung antara robot keseimbangan dengan sistem ROS pada laptop. ESP8266 akan tersambung wifi dan melakukan komunikasi data melalui wifi. Data data yang diterima oleh ESP8266 akan disalurkan ke mikrokontroler arduino melalui komunikasi serial. Selanjutnya data data tersebut digunakan arduino untuk mengontrol pergerakan robot.

Dalam komunikasi data antara ESP8266 dengan ROS, digunakan library ros.h, dimana dengan library tersebut, proses pengambilan dan *pertukaran* data akan lebih mudah dengan pemanggilan fungsi publish dan *subscribe*

Dalam komunikasi data antara ESP8266 dan arduino digunakan library arduinojson.h, library tersebut memungkinkan komunikasi serial dengan cepat dan ringan. Komunikasi data serial antara arduino dan ESP8266 hanya dapat dilakukan dengan kecepatan 10Hz. Hal memberikan respon perubahan kecepatan yang cukup lambat bagi mikrokontroler.

g. Uji Coba dan Evaluasi Robot keseimbangan

Hasil uji coba menghasilkan pembacaan sudut oleh sensor MPU6050 dengan stabil. Pembacaan sensor dapat digunakan oleh robot sebagai inputan bagi PID kontroler untuk menentukan nilai *throttle*, yang selanjutnya digunakan oleh fungsi timer untuk menghasilkan output pulsa untuk menggerakan motor stepper. Robot keseimbangan dapat berfungsi dengan baik, menjaga keseimbangan robot pada setpoint 0 derajat.

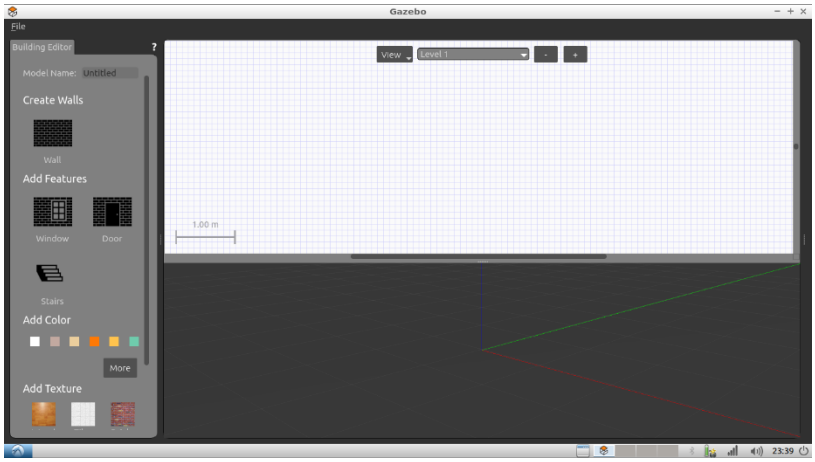
4.2 Pembuatan Sistem ROS

Pembuatan sistem ROS dilakukan dengan menggunakan OS lubuntu 18.04 berbasis linux dan distro melodic. Untuk bisa memulai ROS, dilakukan

instalasi ROS, package, dan tool yang diperlukan. Perlu dilakukan konfigurasi *environment* dan pembuatan *workspace.*

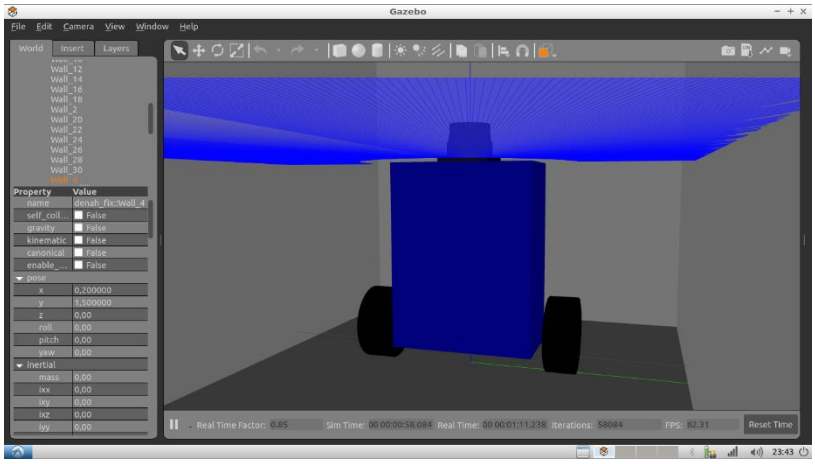
4.2.1 Pembuatan World model dan Robot Model pada Gazebo

Pembuatan *world model* pada gazebo dilakukan dengan menggunakan fitur *building editor* pada gazebo, dapat dilihat pada gambar 4.3. Fitur tersebut memungkinkan untuk membuat *world model* pada gazebo sesuai dengan dunia nyata. *World model* yang digunakan berdasarkan pada denah salah satu ruangan milik penulis.



Gambar 4. 3 *Building Editor Tools* Gazebo

Pembuatan robot model menggunakan *Unified Robotic Description Format (URDF),* yang merupakan XML format yang digunakan di ROS untuk mendeskripsikan semua elemen robot. Secara umum URDF terdiri dari tiga file berformat xacro dan satu file berformat gazebo. Hasil dari pembuatan model robot disajikan pada gambar 4.4.



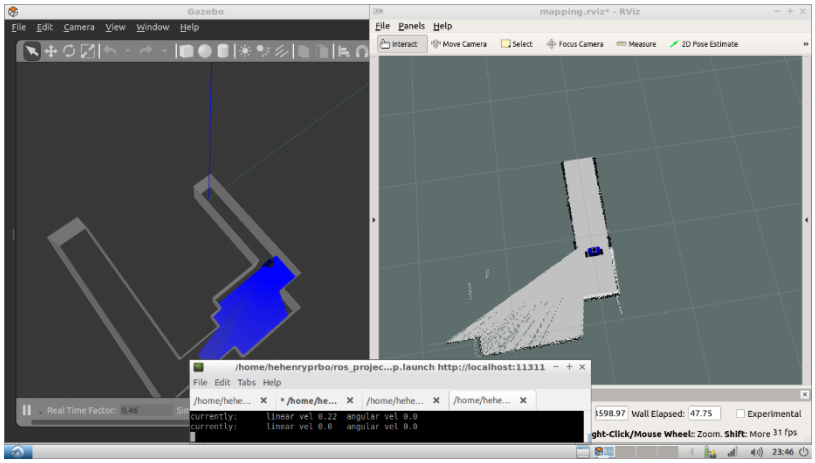
Gambar 4.4 Model Robot Pada Gazebo

4.2.2 Pembuatan Sistem Mapping dan Navigasi dengan RVIZ

Pada dasarnya untuk menjalankan mapping dan navigasi pada ROS, diperlukan tiga tahapan berikut yaitu: pembuatan map, penyimpanan map, dan pemuatan map.

1. Pembuatan Map

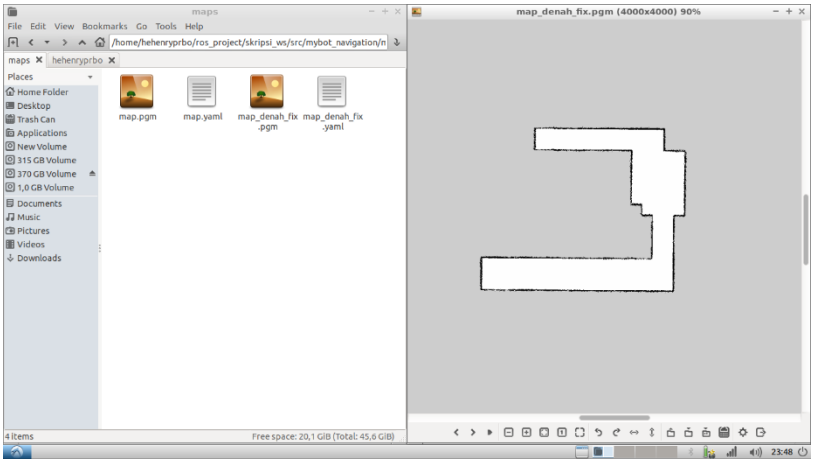
   Pembuatan map dilakukan dengan pemanggilan world model yang telah dibuat di gazebo. Untuk dapat membuat map diperlukan sebuah package yang telah disediakan oleh ROS, yaitu gmapping. Dengan menggunakan package ini, dapat dilakukan pembuatan sebuah *2-D occupancy grid map* dari data laser scan yang telah dibuat pada model robot. Cuplikan proses pembuatan map dapat dilihat pada gambar 4.5.



Gambar 4.5 Pembuatan Map

2. Penyimpanan Map

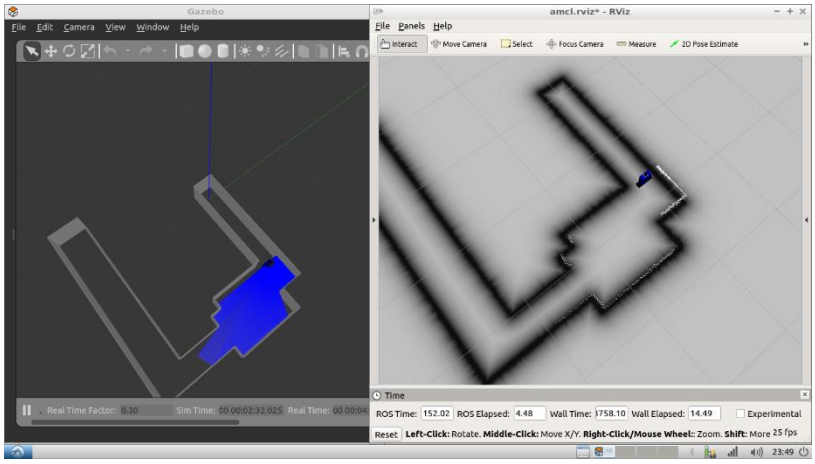
   Penyimpanan Map dilakukan secara sederhana dengan menggunakan *command terminal*. Hasil penyimpanan map dapat dilihat pada gambar 4.6.



Gambar 4.6 Map Hasil Gmapping

3. Pemuatan Map

   Pada tahapan ini, *world model*, *robot model*, map, dan fungsi navigasi dipanggil secara bersamaan. Gambar 4.7 menunjukkan hasil pemuatan map dan fungsi navigasi. Pada tahap ini, sistem navigasi telah dapat digunakan. Dengan menggunakan tools *2D nav goal* pada rviz, dapat dilakukan pemilihan titik tujuan pada map yang telah dibuat, dan robot akan segera menuju ke tujuan.



Gambar 4. 7 Pemuatan Map dan Navigasi

4.2.3 Pembuatan Sistem Komunikasi Data ROS-Arduino

Komunikasi data dilakukan melalui jaringan wifi. Untuk bisa memulai komunikasi, digunakan package rosserial_server. Penggunaan rosserial_server membuat sistem ROS pada laptop sebagai host yang dapat berperan sebagai *publisher*(ketika mengirim data) atau sebagai *subscriber(*ketika menerima data). Setiap data tersimpan pada *topic*

tertentu, sehingga ESP8266 pada arduino tinggal mengambil atau mengirim data sesuai topic.

Dalam peneilitan ini, data yang diambil berupa kecepatan linier x dan kecepatan sudut z. Untuk dapat mengubah kecepatan linier x dan kecepatan sudut z menjadi kecepatan roda kiri dan roda kanan, diperlukan inverse kinematics. Kedua nilai tersebut diubah menjadi nilai kecepatan roda kiri dan roda kanan dengan formula seperti pada gambar 4.8.

$$v_r = \frac{2v_c + w_c L}{2}$$

$$v_l = \frac{2v_c - w_c L}{2}$$

Gambar 4.8 *Inverse Kinematics* Kecepatan Robot

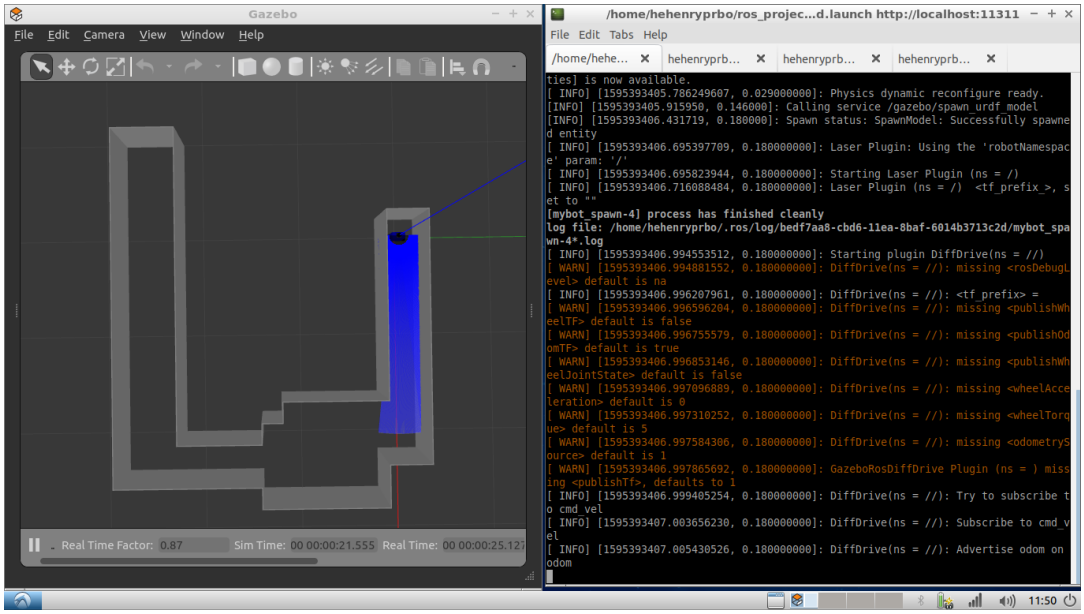
Dimana :

v_c	:	Kecepatan linier Robot
w_c	:	Kcepatan sudut Robot
L	:	Jarak antara roda robot
v_r	:	Kecepatan roda kanan
v_l	:	Kecepatan roda kiri

4.3 Pengujian *Tracking* Simulasi

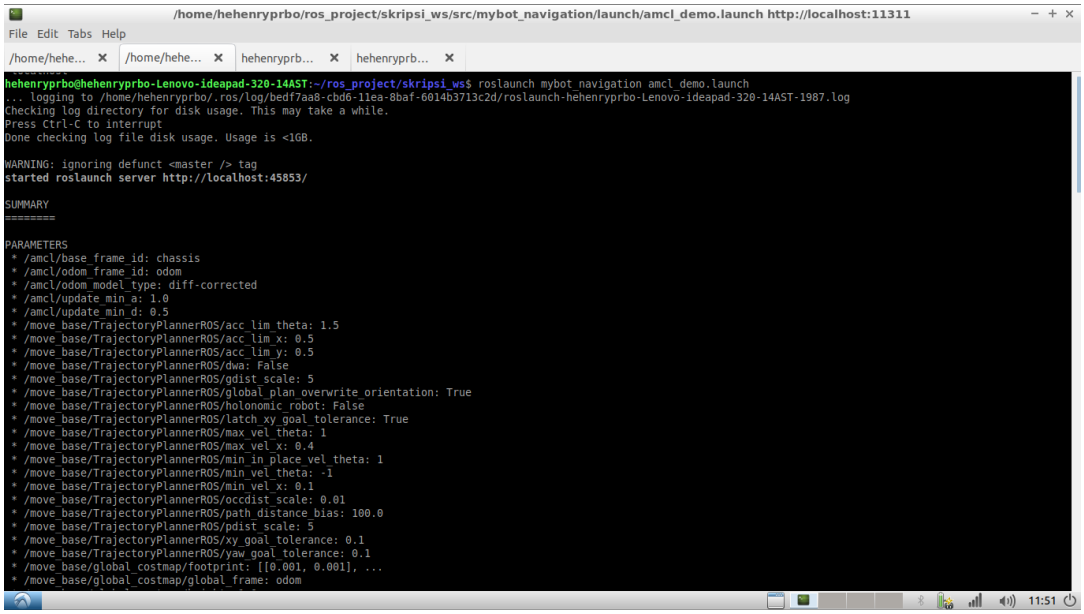
Pengujian *Tracking* simulasi dilakukan pada sistem navigasi di RVIZ. Model robot dan simulasi dunia nyata disajikan pada Gazebo, data data yang dihasilkan GAZEBO ditransfer ke RVIZ digunakan untuk melakukan proses navigasi.

Untuk memulai pengujian, dilakukan pemunculan GAZEBO dengan *command terminal* “roslaunch mybot_gazebo mybot_world.launch”. Hasil pemunculan GAZEBO dapat dilihat pada gambar 4.9.



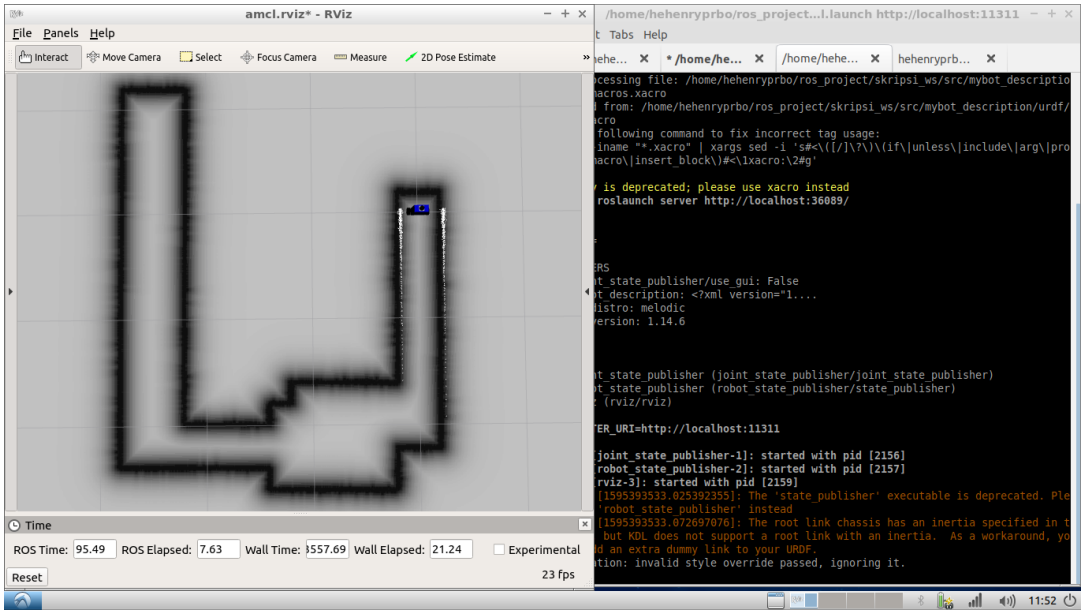
Gambar 4. 9 Pemunculan Gazebo

Selanjutnya dilakukan pemanggilan fungsi navigasi dengan *command terminal* "roslaunch mybot_navigation amcl_demo.launch", seperti yang terlihat pada gambar 4.10.



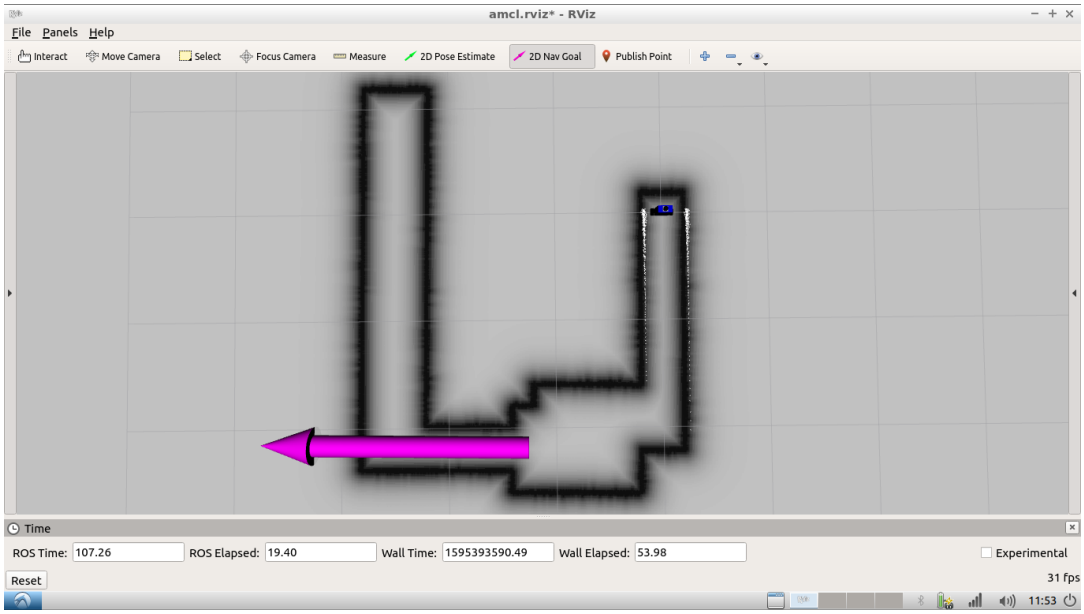
Gambar 4.10 Pemanggilan Fungsi Navigasi

Kemudian dilakukan pemunculan RVIZ dengan *command terminal* "roslaunch mybot_description mybot_rviz_amcl.launch" seperti yang terlihat pada gambar 4.11.



Gambar 4. 11 Pemunculan RVIZ

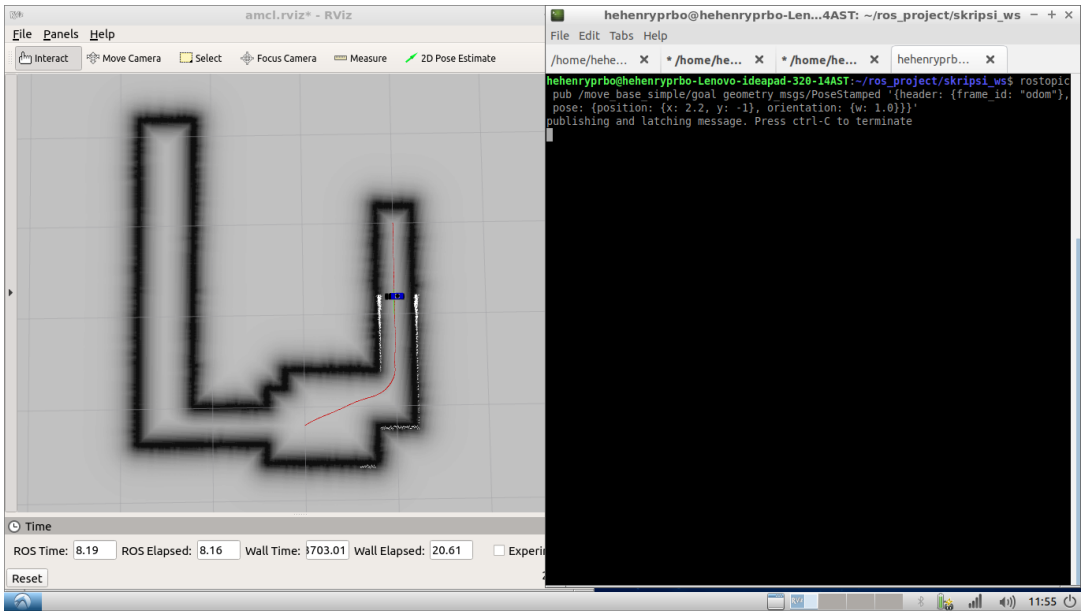
Navigasi dapat dilakukan pada RVIZ, dimana dengan menekan fungsi 2D Nav Goal ke map, maka robot akan segera bergerak menuju ke titik. Contoh penggunaan fungsi 2D Nav Goal dapat dilihat pada gambar 4.12.



Gambar 4.12 Navigasi dengan 2D Nav Goal

Selain itu navigasi dapat dilakukan melalui *command terminal* sehingga robot dapat menuju ke titik koordinat spesifik “rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped '{header: {frame_id:

"odom"}, pose: {position: {x: 2.2, y: -1}, orientation: {w: 1.0}}}'", contoh penggunaan seperti pada gambar 4.13.



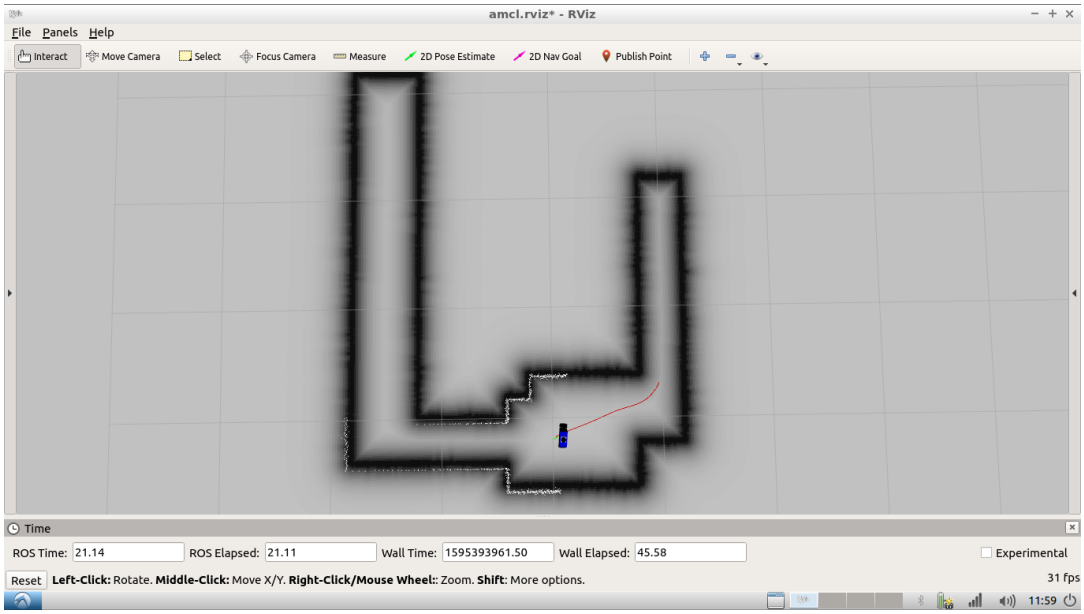
Gambar 4. 13 Navigasi Melalui *command terminal*

Pengujian menghasilkan robot dapat menuju ke titik tujuan baik menggunakan 2D Nav Goal maupun melalui *command terminal*. Dari lima kali pengujian didapatkan data seperti pada gambar 4.14. Gambar 4.15 dan 4.16 menunjukkan robot sampai pada titik tujuan dengan menggunakan fungsi 2D Nav Goal dan melalui *command terminal*.

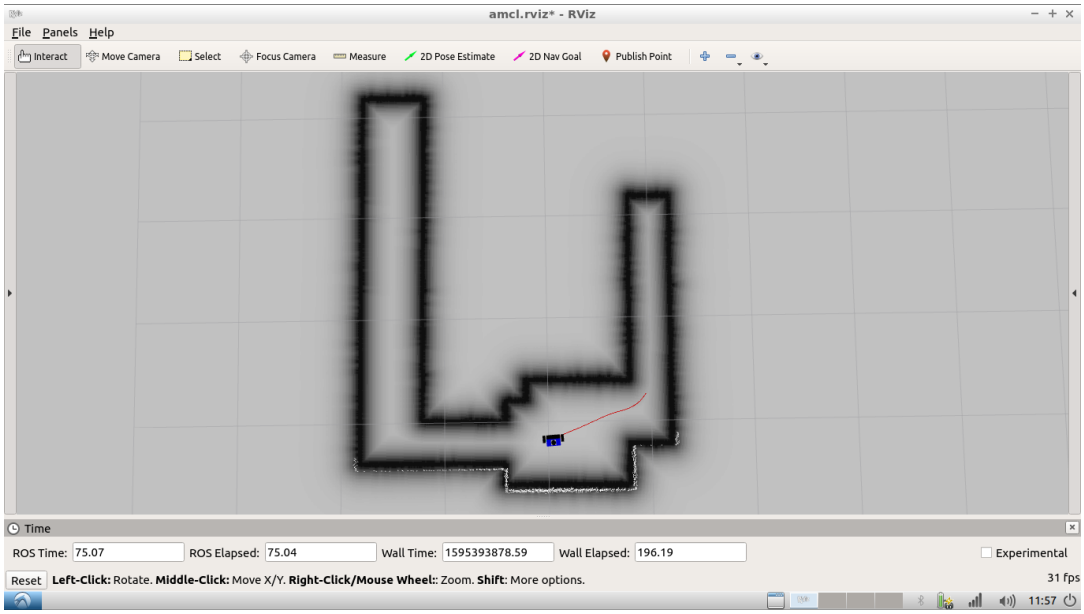
No	Koordinat Tujuan	Hasil Akhir	Error	
			x	y
1	(2.2, -1)	(2.15, -0.95)	-0.05	-0.05
2	(2.2, -1)	(2.15, -0.95)	-0.05	-0.05
3	(2.2, -1)	(2.14, -0.95)	-0.06	-0.05
4	(2.2, -1)	(2.15, -0.95)	-0.05	-0.05
5	(2.2, -1)	(2.15, -0.95)	-0.05	-0.05
Rata-Rata Error			-0.052	-0.05

Gambar 4. 14 Data Hasil Pengujian *Tracking* Simulasi

Hasil pengujian dari *tracking simulasi*, menunjukan error yang hampir konstan di setiap pengujian. Dengan error rata-rata sebesar -0.052 m pada sumbu X dan -0.05 m pada sumbu Y.



Gambar 4. 15 Robot Sampai di 2D Nav Goal



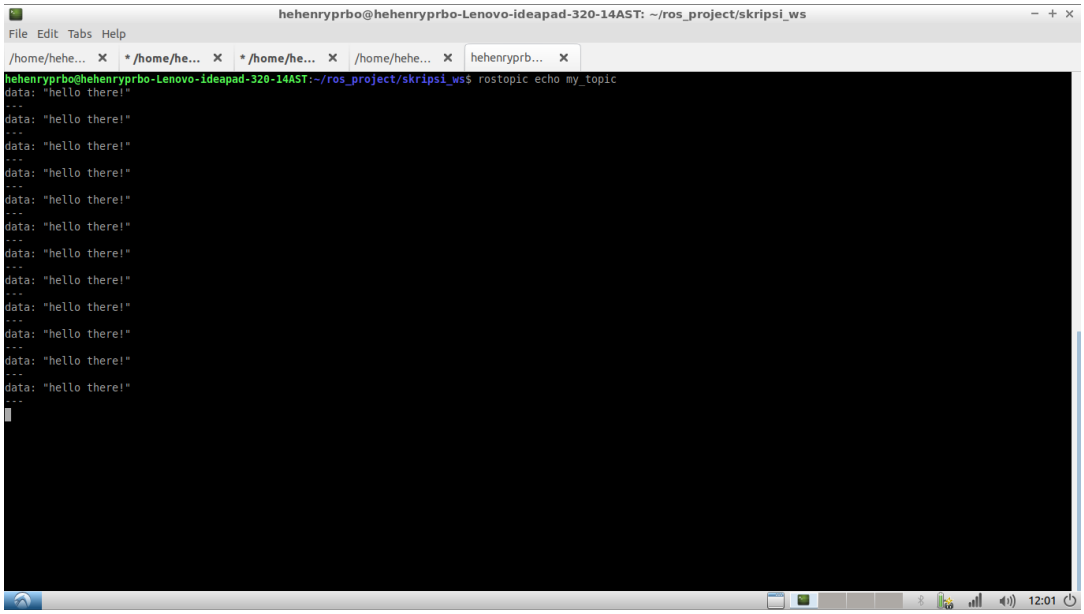
Gambar 4. 16 Robot Sampai ke titik Perintah

4.4 Pengujian *Tracking* Real

Pengujian *Tracking* real melalui proses yang sama seperti pada *Tracking* simulasi, namun ditambah dengan pemanggilan fungsi komunikasi ROS-Arduino dengan *command terminal* "roslaunch rosserial_server socket.launch".

Robot keseimbangan telah diatur untuk bisa melakukan koneksi otomatis dengan sambungan wifi dan ROS. Untuk memastikan bahwa robot

telah terkoneksi dengan sistem ROS, dilakukan pengecakan dengan *command terminal* “rostopic echo my_topic”, maka akan muncul data berupa “hello there” yang dikirim oleh robot keseimbangan ke sistem ROS. Data tersebut membuktikan bahwa komunikasi ROS-Arduino telah terbangun, seperti yang terlihat pada gambar 4.17.



Gambar 4. 17 Terhubungnya Robot denga ROS

Selanjutnya robot keseimbangan dinyalakan dalam keadaan berdiri, dan diletakan di titik koordinat (0, 0), yang telah disesuaikan dengan dunia simulasi pada Gazebo. Kemudian dilakukan pemberian perintah tujuan navigasi. Agar pengujian lebih terukur, perintah tujuan navigasi dilakukan melalui *command terminal*, agar dapat mengarah ke titik koordinat yang spesifik.

Pengujian dilakukan dengan memberikan tujuan ke titik koordinat (2.2, -1). Dari lima kali pengujian didapatkan error rata rata -0.044 m pada sumbu X dan 0.38 m pada sumbu Y, seperti yang terlihat pada gambar 4.18.

No	Koordinat Tujuan	Hasil Akhir	Error	
			x	y
1	(2.2, -1)	(2.21, -1.4)	-0.01	0.40
2	(2.2, -1)	(2.3, -1.38)	-0.10	0.38
3	(2.2, -1)	(2.25, -1.35)	-0.05	0.35
4	(2.2, -1)	(2.22, -1.38)	-0.02	0.38
5	(2.2, -1)	(2.24, -1.37)	-0.04	0.37
Rata-Rata Error			-0.044	0.38

Gambar 4. 18 Data Hasil Pengujian *Tracking* Real

4.5 Analisa Data

4.5.1 Analisa Data *Tracking* Simulasi

Pada pengujian *Tracking* simulasi, robot dapat berjalan menuju titik tujuan yang ditentukan, namun hasil akhir posisi robot mengalami error rata-rata sebesar -0.052 m pada sumbu X dan -0.05 m pada sumbu Y. Meskipun pada sistem navigasi, toleransi error ini dapat ditetapkan, namun apabila nilai toleransi terlalu kecil, menyebabkan robot akan hanya berputar putar di sekitar titik tujuan, sebaliknya apabila penetapan nilai terlalu besar, memberikan pengaruh pada terlalu besarnya error lokasi akhir robot terhadap titik tujuan. Hal ini bisa dikarenakan beberapa faktor:

1. Faktor pengaturan jarak rintangan

   Pengaturan jarak rintangan yang sesuai menimbulkan kebingungan pada robot ketika titik tujuan berada dekat dengan batas atau rintangan, hal ini menimbulkan robot hanya akan berputar putar di sekitar titik tujuan.

2. Faktor sensor *laser scan*

   Sensor berperan penting bagi robot untuk mengerti keadaan di sekitar, pengaturan jarak pandang sensor dan penentuan kondisi lingkungan di sekitar robot akan berpengaruh pada robot ketika ingin mencapai titik tujuan.

3. Faktor Dimensi Robot dan resolusi map

   Dimensi robot dan resolusi map saling berkaitan, dimana ketika pergerakan robot tidak mampu mengimbangi resolusi dari map dan

titik tujuan yang ditetapkan, maka robot akan mengalami kegagalan untuk menuju ke titik tujuan.

4.5.2 Analisa Data *Tracking* Real

Pada pengujian *Tracking* real, robot dapat berjalan sesuai dengan navigasi yang diberikan. Robot mampu menuju titik tujuan yang ditentukan (2.2, -1), namun mengalami error yang cukup tinggi bila dibanding dengan *Tracking* simulasi. Selain faktor faktor yang terdapat pada *Tracking* simulasi, terdapat faktor lain yang mempengaruhi.

1. Faktor manufaktur robot

   Proses manufaktur robot yang kurang presisi, seperti peletakan kedua roda dan kesejajaran antar roda.

2. Faktor transfer data

   Pada pengujian *Tracking* real ini, data disampaikan dari sistem navigasi ROS menuju ke ESP8266 kemudian baru diterima oleh arduino untuk menggerakan robot. Proses transfer data antara ROS ke ESP8266 tidak terlalu ada kendala, frekuensi komunikasi data bisa mencapai 1KHz. Namun komunikasi antara ESP8266 dengan arduino, hanya terbatas di 10Hz. Ketika frekuensi komunikasi dinaikan, akan terjadi error yang mengakibatkan proses komunikasi terhenti dan mengganggu proses penyeimbangan pada robot. Selain itu frekuensi yang terlalu tinggi menyebabkan penambahan delay pada program arduino.

3. Faktor Komputasi PC

   Penggunaan ROS membutuhkan komputasi yang cukup besar, dimana untuk menjalankan proses navigasi dibutuhkan setidaknya tiga package. PC yang memiliki kemampuan komputasi yang kurang memberikan dampak pada kecepatan pemrosesan sistem navigasi. Hal ini berdampak munculnya lag pada simulasi maupun visualisasi RVIZ dan delay pada setiap perhitungan kecepatan robot, sehingga membuat robot harus berulang kali berhenti.

4. Faktor kalibrasi kecepatan

Kalibrasi kecepatan menjadi faktor penting pada penelitian yang dilakukan, karena kecepatan menjadi satu satunya acuan bagi robot untuk bergerak sesuai navigasi. Kecepatan pada *Tracking* simulasi diatur pada range 0.1 – 0.2 m/s. Dengan keliling roda robot keseimbangan sebesar 0.21352, *microstepping* 800, dan timer 0.00002 s. Maka untuk robot dapat berjalan dengan kecepatan 0.1 – 0.2 m/s dibutuhkan pemberian pulsa antara 373.1343284 – 769.2307692 pulsa/s. Dengan perhitungan tersebut, kecepatan dapat disesuaikan dengan data kecepatan dari sistem ROS. Namun karena robot keseimbangan membutuhkan sebuah proses penyeimbangan, maka untuk bisa mencapai kepada kecepatan yang dibutuhkan. Robot perlu melalui proses pemiringan badan robot, sehingga mempengaruhi hasil kalibrasi kecepatan.

5. Faktor permukaan jalan yang dilalui

   Robot keseimbangan sangat mudah terpengaruh oleh keadaan permukaan jalan yang dilalui. Ketika jalan yang dilalui semakin tinggi, maka akan terjadi proses perlambatan robot, sebaliknya ketika jalan yang dilalui semakin rendah, akan terjadi penambahan kecepatan pada robot. Perubahan perubahan ini yang membuat jarak yang dilalui robot tidak sesuai dengan navigasi yang diberikan oleh sistem ROS.

4.5.3 Analisa Perbandingan Hasil *Tracking* Simulasi dan Real

Hasil pengujian antara *tracking* simulasi dan *tracking* real menunjukan perbedaan rata rata error. Pada pengujian *tracking* real terjadi perbedaan error yang cukup tinggi di sumbu Y, serta kurang akuratnya robot menuju titik tujuan bila dibandingkan dengan *tracking* simulasi. Secara keseluruhan hal ini dikarenakan pada *tracking* simulasi, memiliki kondisi yang ideal, dimana semua parameter berjalan sesuai dengan nilai yang telah ditentukan. Pada *tracking* real, kondisi ideal sulit sekali dicapai karena banyak faktor seperti yang dijelaskan pada poin 4.5.2, serta tidak adanya *feedback* dari robot keseimbangan pada dunia nyata untuk mengerti kondisi lingkungan sekitar robot.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

a. Robot keseimbangan beroda dua dapat dibangun menggunakan mikrokontroler arduino dengan penggerak berupa motor stepper.
b. Integrasi robot keseimbangan beroda dua dapat diintegrasikan dengan ROS dengan menambahkan ESP8266 pada robot, yang kemudian dapat melakukan komunikasi dengan sistem ROS melalui jaringan wifi.
c. Sistem navigasi pada robot keseimbangan beroda dua dibuat dengan menggunakan *navigation package* yang tersedia pada ROS.
d. Uji coba *tracking* simulasi menunjukan robot virtual mampu menuju lokasi tujuan dengan error rata-rata -0.052 m di sumbu X dan -0.05 m di sumbu Y. Uji coba *tracking* real menunjukan robot keseimbangan mampu menerima data kecepatan dari sistem ROS, untuk bergerak menuju titik tujuan berdasarkan virtual map dan virtual sensor, dengan error rata-rata -0.044 m pada sumbu X dan 0.38 m pada sumbu Y.
e. Perbedaan hasil uji coba antara *tracking* simulasi dan *tracking* real dipengaruhi beberapa faktor seperti: konfigurasi parameter dalam sistem ROS yang kurang tepat, manufaktur robot, transfer data, kemampuan komputasi PC, kalibrasi kecepatan, dan permukaan jalan yang dilalui.

5.2 Saran

Dari penelitian yang sudah dilakukan, didapatkan beberapa hal yang dapat dijadikan sebagai masukan untuk penelitian lebih lanjut :

a. Perlu adanya penambahan sensor pada robot keseimbangan, sehingga dapat melakukan lokalisasi dan mengenali keadaan dari lingkungan sekitar secara real time.
b. Perlu adanya penggantian penggunaan mikrokontroler dengan spesifikasi lebih tinggi seperti STM32 atau menggunakan *single board computer* seperti raspberry pi.
c. Menggunakan permodelan sistem dalam mengendalikan robot keseimbangan.

d. Penggunaan motor dengan kualitas yang lebih baik atau motor dengan sensor yang memberikan umpan balik.

DAFTAR PUSTAKA

[1] D. Erlangga, E. D, R. H S, S. Sunarto, K. Rahardjo T.S, and F. G, "Sistem Navigasi Mobile Robot Dalam Ruangan Berbasis Autonomous Navigation," *J. Mech. Eng. Mechatronics*, vol. 4, no. 2, p. 78, 2019.

[2] S. Gatesichapakorn, J. Takamatsu, and M. Ruchanurucks, "ROS based Autonomous Mobile Robot Navigation using 2D LiDAR and RGB-D Camera," *2019 1st Int. Symp. Instrumentation, Control. Artif. Intell. Robot. ICA-SYMP 2019*, pp. 151–154, 2019.

[3] Z. Fang, S. Zhao, S. Wen, and Y. Zhang, "A Real-Time 3D Perception and Reconstruction System Based on a 2D Laser Scanner," *J. Sensors*, vol. 2018, pp. 454–459, 2018.

[4] C. Luo, G. E. Jan, J. Zhang, and F. Shen, "Boundary aware navigation and mapping for a mobile automaton," *2016 IEEE Int. Conf. Inf. Autom. IEEE ICIA 2016*, no. August, pp. 561–566, 2017.

[5] Z. Li and X. Mei, "Navigation and Control System of Mobile Robot Based on ROS," *Proc. 2018 IEEE 3rd Adv. Inf. Technol. Electron. Autom. Control Conf. IAEAC 2018*, no. Iaeac, pp. 368–372, 2018.

[6] H. Bin, L. W. Zhen, and L. H. Feng, "The kinematics model of a two-wheeled self-balancing autonomous mobile robot and its simulation," *2010 2nd Int. Conf. Comput. Eng. Appl. ICCEA 2010*, vol. 2, pp. 64–68, 2010.

[7] R. Sadeghian and M. T. Masoule, "An experimental study on the PID and Fuzzy-PID controllers on a designed two-wheeled self-balancing autonomous robot," *2016 4th Int. Conf. Control. Instrumentation, Autom. ICCIA 2016*, no. January, pp. 313–318, 2016.

[8] K. V Ignatiev, M. M. Kopichev, and A. V Putov, "Autonomous Omni-Wheeled Mobile Robots," pp. 0–3, 2016.

[9] M. Köseoğlu, O. M. Çelik, and Ö. Pektaş, "Design of an autonomous mobile robot based on ROS," *IDAP 2017 - Int. Artif. Intell. Data Process. Symp.*, 2017.

[10] G. Zhang and T. Liu, "Bio-inspired autonomous navigation system for logistics mobile robots with inertial AHRS," *Proc. 2017 IEEE 3rd Inf. Technol. Mechatronics Eng. Conf. ITOEC 2017*, vol. 2017-Janua, pp. 971–

975, 2017.

[11] J. S. Saputro, P. H. Rusmin, and A. S. Rochman, “Design and implementation of trajectory tracking motion in mobile robot skid steering using model predictive control,” *ICSET 2018 - 2018 IEEE 8th Int. Conf. Syst. Eng. Technol. Proc.*, no. October, pp. 73–78, 2019.

[12] A. Jalil, “Robot Operating System (Ros) Dan Gazebo Sebagai Media Pembelajaran Robot Interaktif,” *Ilk. J. Ilm.*, vol. 10, no. 3, pp. 284–289, 2018.

[13] R. Kannan Megalingam, C. Ravi Teja, S. Sreekanth, and A. Raj, “ROS based Autonomous Indoor Navigation Simulation Using SLAM Algorithm,” *Int. J. Pure Appl. Math.*, vol. 118, no. 7, pp. 199–205, 2018.

[14] S. Khan and M. K. Ahmmed, “Where am I? Autonomous navigation system of a mobile robot in an unknown environment,” *2016 5th Int. Conf. Informatics, Electron. Vision, ICIEV 2016*, pp. 56–61, 2016.

[15] Abba Suganda Girsang, “ALGORITMA DIJKSTRA,” *mti.binus.ac.id*, 2017. [Online]. Available: https://mti.binus.ac.id/2017/11/28/algoritma-dijkstra/.

[16] T. A. Mai, D. N. Anisimov, T. S. Dang, and E. Fedorova, “Fuzzy-PID Controller for Two Wheels Balancing Robot Based on STM32 Microcontroller,” *Proc. - 2019 Int. Conf. Eng. Technol. Comput. Sci. Innov. Appl. EnT 2019*, pp. 20–24, 2019.

[17] S. Xin, M. Gong, Y. Sun, and Z. Zhang, “Control system design for two-wheel self-balanced robot based on Fuzzy-PD control,” *5th Int. Conf. Intell. Control Inf. Process. ICICIP 2014 - Proc.*, pp. 169–174, 2015.

LAMPIRAN

Program Arduino

```
#include
<Wire.h>                                                    //Include the
Wire.h library so we can communicate with the gyro
#include <ArduinoJson.h>

int gyro_address =
0x68;                                          //MPU-6050 I2C address
(0x68 or 0x69)
int acc_calibration_value = -8368;

//Various settings
float pid_p_gain =
10;                                            //Gain setting for the
P-controller (10)
float pid_i_gain =
1.1;                                           //Gain setting for the
I-controller (1.1)
float pid_d_gain =
10;                                            //Gain setting for the
D-controller (10)
float turning_speed =
30;                                         //Turning speed (20)
float max_target_speed = 150;
float kalibrasiSpeed = 12.5;
int deadband = 10;
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
///
//Declaring global variables
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
///
byte start, received_byte, low_bat;

int left_motor, throttle_left_motor, throttle_counter_left_motor,
throttle_left_motor_memory;
int right_motor, throttle_right_motor,
throttle_counter_right_motor, throttle_right_motor_memory;
int battery_voltage;
int receive_counter;
int gyro_pitch_data_raw, gyro_yaw_data_raw,
accelerometer_data_raw;

long gyro_yaw_calibration_value, gyro_pitch_calibration_value;

unsigned long loop_timer;

float angle_acc, angle, self_balance_pid_setpoint;
float pid_error_temp, pid_i_mem, gyro_input, pid_output,
pid_last_d_error;
float pid_output_left, pid_output_right;
float pid_setpoint = -0.5;
double angle_gyro;

//communication between arduino esp
String command;
float vr, vl;
```

```
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
///
//Setup basic functions
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
///
void setup(){
  Serial.begin(9600);
          //Start the serial port at 9600 kbps
  Wire.begin();
          //Start the I2C bus as master
  TWBR =
12;
 //Set the I2C clock speed to 400kHz

  //To create a variable pulse for controlling the stepper motors
a timer is created that will execute a piece of code (subroutine)
every 20us
  //This subroutine is called TIMER2_COMPA_vect
  TCCR2A =
0;                                                                /
/Make sure that the TCCR2A register is set to zero
  TCCR2B =
0;                                                                /
/Make sure that the TCCR2A register is set to zero
  TIMSK2 |= (1 <<
OCIE2A);                                                  //Set
the interupt enable bit OCIE2A in the TIMSK2 register
  TCCR2B |= (1 <<
CS21);                                                    //Set
the CS21 bit in the TCCRB register to set the prescaler to 8
  OCR2A =
39;
//The compare register is set to 39 => 20us / (1s / (16.000.000MHz
/ 8)) - 1
  TCCR2A |= (1 <<
WGM21);                                                   //Set
counter 2 to CTC (clear timer on compare) mode

  //By default the MPU-6050 sleeps. So we have to wake it up.
  Wire.beginTransmission(gyro_address);
          //Start communication with the address found during
search.
  Wire.write(0x6B);
          //We want to write to the PWR_MGMT_1 register (6B hex)
  Wire.write(0x00);
          //Set the register bits as 00000000 to activate the gyro
  Wire.endTransmission();
          //End the transmission with the gyro.
  //Set the full scale of the gyro to +/- 250 degrees per second
  Wire.beginTransmission(gyro_address);
          //Start communication with the address found during
search.
  Wire.write(0x1B);
          //We want to write to the GYRO_CONFIG register (1B hex)
  Wire.write(0x00);
          //Set the register bits as 00000000 (250dps full scale)
```

```
  Wire.endTransmission();
          //End the transmission with the gyro
  //Set the full scale of the accelerometer to +/- 4g.
  Wire.beginTransmission(gyro_address);
          //Start communication with the address found during
search.
  Wire.write(0x1C);
          //We want to write to the ACCEL_CONFIG register (1A hex)
  Wire.write(0x08);
          //Set the register bits as 00001000 (+/- 4g full scale
range)
  Wire.endTransmission();
          //End the transmission with the gyro
  //Set some filtering to improve the raw data.
  Wire.beginTransmission(gyro_address);
          //Start communication with the address found during
search
  Wire.write(0x1A);
          //We want to write to the CONFIG register (1A hex)
  Wire.write(0x03);
          //Set the register bits as 00000011 (Set Digital Low
Pass Filter to ~43Hz)
  Wire.endTransmission();
          //End the transmission with the gyro

  DDRD = DDRD | B11111100;    // this is safer as it sets pins 2
to 7 as outputs
  DDRB = DDRB | B00111111;    // this is safer as it sets pins 8
to 13 as outputs
  pinMode(13,
OUTPUT);


//  for(receive_counter = 0; receive_counter < 500;
receive_counter++){       //Create 500 loops
//    if(receive_counter % 15 == 0)digitalWrite(13,
!digitalRead(13));        //Change the state of the LED every 15
loops to make the LED blink fast
//    Wire.beginTransmission(gyro_address);
            //Start communication with the gyro
//    Wire.write(0x43);
            //Start reading the Who_am_I register 75h
//    Wire.endTransmission();
            //End the transmission
//    Wire.requestFrom(gyro_address,
4);                                       //Request 2 bytes from
the gyro
//    gyro_yaw_calibration_value +=
Wire.read()<<8|Wire.read();               //Combine the two bytes
to make one integer
//    gyro_pitch_calibration_value +=
Wire.read()<<8|Wire.read();             //Combine the two bytes to
make one integer
//    delayMicroseconds(3700);
            //Wait for 3700 microseconds to simulate the main
program loop time
//  }
```

```
//  gyro_pitch_calibration_value /=
500;                                          //Divide the total value
by 500 to get the avarage gyro offset
//  gyro_yaw_calibration_value /=
500;                                            //Divide the total
value by 500 to get the avarage gyro offset

  gyro_pitch_calibration_value =
247;                                        //Divide the total value
by 500 to get the avarage gyro offset
  gyro_yaw_calibration_value = -38;
//  Serial.print(gyro_pitch_calibration_value);Serial.print("\t");
Serial.print(gyro_yaw_calibration_value);
//  Serial.print("\n\r");


  loop_timer = micros() +
4000;                                                  //Set the
loop_timer variable at the next end loop time

}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
///
//Main program loop
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
///
void loop(){
  ////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
/////
  //Angle calculations
  ////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
/////
  Wire.beginTransmission(gyro_address);
          //Start communication with the gyro
  Wire.write(0x3F);
          //Start reading at register 3F
  Wire.endTransmission();
          //End the transmission
  Wire.requestFrom(gyro_address,
2);                                          //Request 2 bytes from
the gyro
  accelerometer_data_raw =
Wire.read()<<8|Wire.read();                       //Combine the two
bytes to make one integer
  accelerometer_data_raw +=
acc_calibration_value;                           //Add the
accelerometer calibration value
  if(accelerometer_data_raw > 8200)accelerometer_data_raw =
8200;           //Prevent division by zero by limiting the acc
data to +/-8200;
  if(accelerometer_data_raw < -8200)accelerometer_data_raw = -
8200;         //Prevent division by zero by limiting the acc data
to +/-8200;
```

```
  angle_acc = asin((float)accelerometer_data_raw/8200.0)*
57.296;           //Calculate the current angle according to the
accelerometer

  if(start == 0 && angle_acc > -0.5&& angle_acc <
0.5){                     //If the accelerometer angle is almost 0
    angle_gyro =
angle_acc;                                                  //Load
the accelerometer angle in the angle_gyro variable
    start =
1;                                                                //
Set the start variable to start the PID controller
  }

  Wire.beginTransmission(gyro_address);
         //Start communication with the gyro
  Wire.write(0x43);
         //Start reading at register 43
  Wire.endTransmission();
         //End the transmission
  Wire.requestFrom(gyro_address,
4);                                       //Request 4 bytes from
the gyro
  gyro_yaw_data_raw =
Wire.read()<<8|Wire.read();                                 //Combine
the two bytes to make one integer
  gyro_pitch_data_raw =
Wire.read()<<8|Wire.read();                               //Combine the
two bytes to make one integer

  gyro_pitch_data_raw -=
gyro_pitch_calibration_value;                           //Add the gyro
calibration value
  angle_gyro += gyro_pitch_data_raw *
0.000031;                                 //Calculate the traveled
during this loop angle and add this to the angle_gyro variable

  gyro_yaw_data_raw -=
gyro_yaw_calibration_value;                               //Add the
gyro calibration value
  //Uncomment the following line to make the compensation active
//  angle_gyro -= gyro_yaw_data_raw * -
0.0000003;                                //Compensate the gyro offset
when the robot is rotating

  angle_gyro = angle_gyro * 0.9995 + angle_acc * 0.0005;
//  angle_gyro = angle_gyro * 0.9999 + angle_acc * 0.0001;

  ////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
/////
  //PID controller calculations
  ////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
/////
  //The balancing robot is angle driven. First the difference
between the desired angel (setpoint) and actual angle (process
value)
```

```
  //is calculated. The self_balance_pid_setpoint variable is
automatically changed to make sure that the robot stays balanced
all the time.
  //The (pid_setpoint - pid_output * 0.015) part functions as a
brake function.
  pid_error_temp = angle_gyro - self_balance_pid_setpoint -
pid_setpoint;
  if(pid_output > 10 || pid_output < -10)pid_error_temp +=
pid_output * 0.015 ;

  pid_i_mem += pid_i_gain *
pid_error_temp;                                       //Calculate the I-
controller value and add it to the pid_i_mem variable
  if(pid_i_mem > 400)pid_i_mem =
400;                                                  //Limit the I-
controller to the maximum controller output
  else if(pid_i_mem < -400)pid_i_mem = -400;
  //Calculate the PID output value
  pid_output = pid_p_gain * pid_error_temp + pid_i_mem +
pid_d_gain * (pid_error_temp - pid_last_d_error);
  if(pid_output > 400)pid_output =
400;                                                //Limit the PI-controller
to the maximum controller output
  else if(pid_output < -400)pid_output = -400;

  pid_last_d_error =
pid_error_temp;                                                 //Store the
error for the next loop

  if(pid_output < deadband && pid_output > -deadband)pid_output =
0;                          //Create a dead-band to stop the motors
when the robot is balanced

  if(angle_gyro > 20 || angle_gyro < -20 || start == 0){     //If
the robot tips over or the start variable is zero or the battery
is empty
    pid_output =
0;                                                                //Set
the PID controller output to 0 so the motors stop moving
    pid_i_mem =
0;                                                                 //Rese
t the I-controller memory
    start =
0;                                                                      //
Set the start variable to 0
    self_balance_pid_setpoint =
0;                                             //Reset the
self_balance_pid_setpoint variable
  }

  ////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
/////
  //Control calculations
  ////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
/////
  pid_output_left =
pid_output;                                                   //Copy the
```

```
controller output to the pid_output_left variable for the left
motor
  pid_output_right = pid_output;
  takeCommand();
// Take data from esp and then use it to control the spin of each
stepper
  if(vr == vl && vr > 0){
    pid_setpoint = vr * kalibrasiSpeed;
  }
  if(vr == vl && vr < 0){
    pid_setpoint = vr * kalibrasiSpeed;
  }
  if(vr == vl && vr == 0){
    pid_setpoint = 0;
  }
//  pid_output_left = vl * kalibrasiSpeed;
//  pid_output_right = vr * kalibrasiSpeed;
  //The self balancing point is adjusted when there is not forward
or backwards movement from the transmitter. This way the robot
will always find it's balancing point
  if(pid_setpoint ==
0){                                                    //If the
setpoint is zero degrees
    if(pid_output < 0)self_balance_pid_setpoint +=
0.0015;                   //Increase the self_balance_pid_setpoint
if the robot is still moving forewards
    if(pid_output > 0)self_balance_pid_setpoint -=
0.0015;                   //Decrease the self_balance_pid_setpoint
if the robot is still moving backwards
  }

  ////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
/////
  //Motor pulse calculations
  ////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
/////
  //To compensate for the non-linear behaviour of the stepper
motors the folowing calculations are needed to get a linear speed
behaviour.
  if(pid_output_left > 0)pid_output_left = 405 -
(1/(pid_output_left + 9)) * 5500;
  else if(pid_output_left < 0)pid_output_left = -405 -
(1/(pid_output_left - 9)) * 5500;

  if(pid_output_right > 0)pid_output_right = 405 -
(1/(pid_output_right + 9)) * 5500;
  else if(pid_output_right < 0)pid_output_right = -405 -
(1/(pid_output_right - 9)) * 5500;

  //Calculate the needed pulse time for the left and right stepper
motor controllers
  if(pid_output_left > 0)left_motor = 400 - pid_output_left;
  else if(pid_output_left < 0)left_motor = -400 - pid_output_left;
  else left_motor = 0;

  if(pid_output_right > 0)right_motor = 400 - pid_output_right;
```

```
  else if(pid_output_right < 0)right_motor = -400 -
pid_output_right;
  else right_motor = 0;
  throttle_left_motor = left_motor;
  throttle_right_motor = right_motor;

  Serial.print(angle_gyro);//Serial.print("\t");Serial.print(left_
motor);Serial.print("\t");Serial.print(pid_output);
  Serial.print("\n\r");
  while(loop_timer > micros());
  loop_timer += 4000;
}
void takeCommand() {
  // Check if the other Arduino is transmitting
  if (Serial.available())
  {
    // Allocate the JSON document
    // This one must be bigger than for the sender because it must
store the strings
    StaticJsonDocument<300> doc;

    // Read the JSON document from the "link" serial port
    DeserializationError err = deserializeJson(doc, Serial);

    if (err == DeserializationError::Ok)
    {
      vr = doc["vr"].as<float>();
      vl = doc["vl"].as<float>();
//      Serial.println(vr);
    }
  }
}
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
///
//Interrupt routine  TIMER2_COMPA_vect
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
///
ISR(TIMER2_COMPA_vect){
  //Left motor pulse calculations
  throttle_counter_left_motor
++;                                              //Increase the
throttle_counter_left_motor variable by 1 every time this routine
is executed
  if(throttle_counter_left_motor >
throttle_left_motor_memory){              //If the number of loops
is larger then the throttle_left_motor_memory variable
    throttle_counter_left_motor =
0;                                          //Reset the
throttle_counter_left_motor variable
    throttle_left_motor_memory =
throttle_left_motor;                         //Load the next
throttle_left_motor variable
    if(throttle_left_motor_memory <
0){                                       //If the
throttle_left_motor_memory is negative
```

```
      PORTD = PORTD &
0b11011111;                                          //Set output
5 low to reverse the direction of the stepper controller
      throttle_left_motor_memory *= -
1;                                     //Invert the
throttle_left_motor_memory variable
    }
    else PORTD = PORTD |
0b00100000;                                        //Set output 5
high for a forward direction of the stepper motor
  }
  else if(throttle_counter_left_motor == 1)PORTD |=
0b00000100;             //Set output 2 high to create a pulse for
the stepper controller
  else if(throttle_counter_left_motor == 2)PORTD &=
0b11111011;             //Set output 2 low because the pulse only
has to last for 20us

  //right motor pulse calculations
  throttle_counter_right_motor
++;                                          //Increase the
throttle_counter_right_motor variable by 1 every time the routine
is executed
  if(throttle_counter_right_motor >
throttle_right_motor_memory){           //If the number of loops
is larger then the throttle_right_motor_memory variable
    throttle_counter_right_motor =
0;                                      //Reset the
throttle_counter_right_motor variable
    throttle_right_motor_memory =
throttle_right_motor;                    //Load the next
throttle_right_motor variable
    if(throttle_right_motor_memory <
0){                                    //If the
throttle_right_motor_memory is negative
       PORTD = PORTD &
0b10111111;                                          //Set output
6 low to reverse the direction of the stepper controller
      throttle_right_motor_memory *= -
1;                                    //Invert the
throttle_right_motor_memory variable
    }
    else PORTD = PORTD |
0b01000000;                                        //Set output 6
high for a forward direction of the stepper motor
  }
  else if(throttle_counter_right_motor == 1)PORTD |=
0b00001000;            //Set output 3 high to create a pulse for
the stepper controller
  else if(throttle_counter_right_motor == 2)PORTD &=
0b11110111;            //Set output 3 low because the pulse only
has to last for 20us
}
```

Program ESP8266

```
// library declaration
#include <ESP8266WiFi.h>
#include <ros.h>
#include <std_msgs/Float64.h>
#include <std_msgs/String.h>
#include "geometry_msgs/Twist.h"
#include <ArduinoJson.h>

//  create some wifi setup and  declaration
const char* ssid     = "balancing";
const char* password = "tamvanama";
// connect to same ip with ROS system
IPAddress server(192,168,43,93);
// connect to port of ROS system
const uint16_t serverPort = 11411;

// declare some variable to make esp send data to Arduino only if
data different with the previous data
float x, z, lastX, lastZ;
float vr, vl;
float L = 0.15;
char hello[13] = "hello there!";

// ESP as subscriber take data from specific ROS topic
void messageCb1( const geometry_msgs::Twist& vel){
  x = vel.linear.x;
  z = vel.angular.z;
}

// some function to initiate data communication between esp and
ros
std_msgs::String str_msg;
ros::NodeHandle nh;
ros::Subscriber<geometry_msgs::Twist> s1("cmd_vel", &messageCb1);
ros::Publisher p("my_topic", &str_msg);

// function that only run once every system turn on
void setup()
{
  // Use ESP8266 serial to monitor the process
  Serial.begin(9600);
          //Start the serial port at 9600 kbps
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  // Connect the ESP8266 the the wifi AP
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // Another way to get IP
  Serial.print("server : ");
  Serial.println(server);
  Serial.print("serverPort : ");
```

```
  Serial.println(serverPort);
  Serial.print("My IP = ");
  Serial.println(WiFi.localIP());

  // Set the connection to rosserial socket server
  nh.getHardware()->setConnection(server, serverPort);
  nh.initNode();
  nh.advertise(p);
  nh.subscribe(s1);
}

void loop()
{
// check whether esp connect to the ros system, send some word to
ros system, indicated that esp connected to ros
  if (nh.connected()) {
    str_msg.data = hello;
    p.publish( &str_msg );
  } else {
    Serial.println("Not Connected");
  }
// code for translating data from ros to velocity value for
stepper on arduino
  vr = ((2 * x) + (z * L)) / 2;
  vl = ((2 * x) - (z * L)) / 2;
  StaticJsonDocument<200> doc;
  doc["vr"] = vr;
  doc["vl"] = vl;
  serializeJson(doc, Serial);
  nh.spinOnce();
// make sure that data send only within 100 miliseconds
  delay(100);
}
```

Dokumentasi Kegiatan

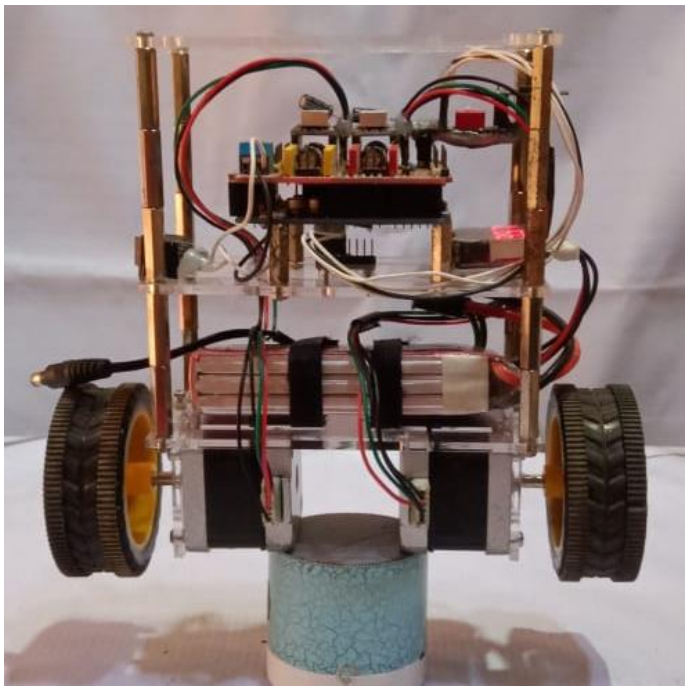


Foto Robot Keseimbangan Tampak Depan

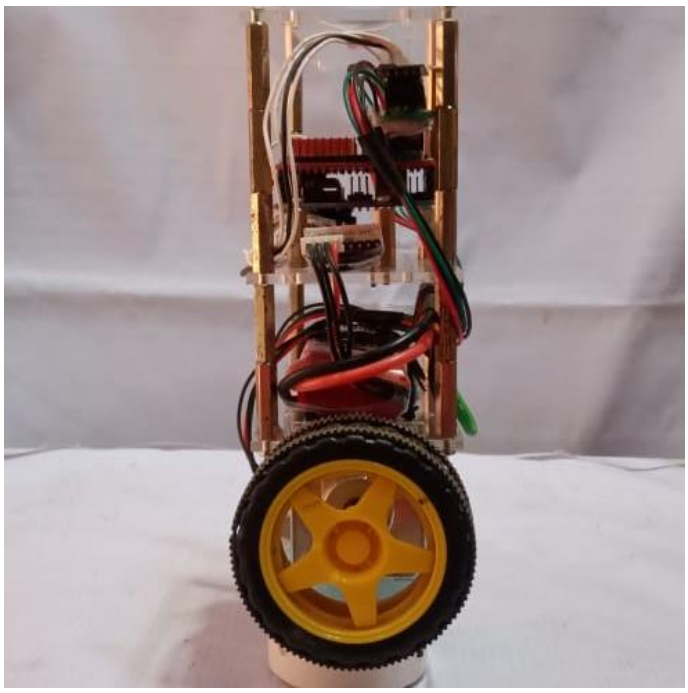


Foto Robot Keseimbangan Tampak Samping

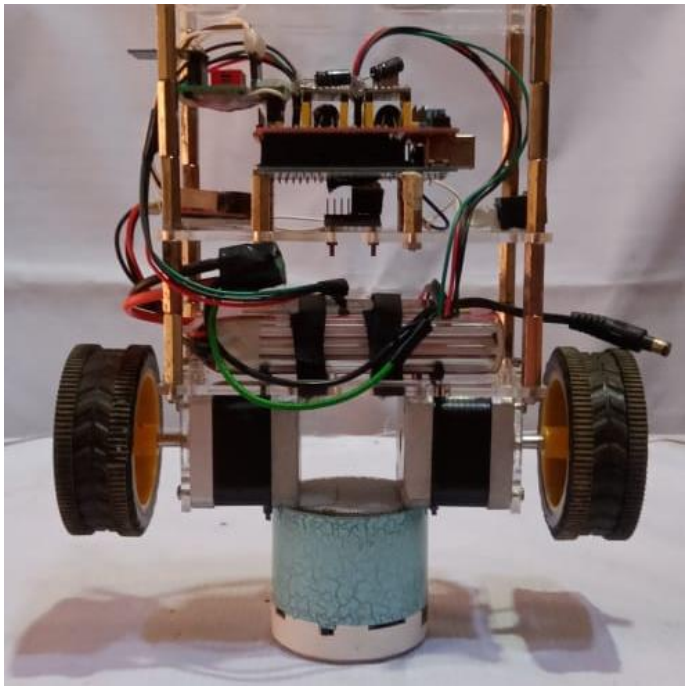


Foto Robot Keseimbangan Tampak Belakang

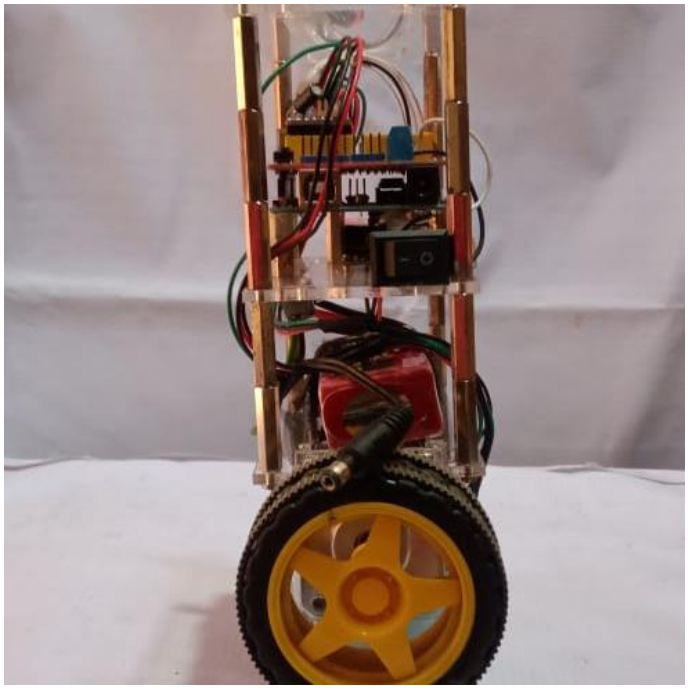


Foto Robot Keseimbangan Tampak Samping

Foto Robot Keseimbangan Saat Mencapai Titik Tujuan (2.2, -1)

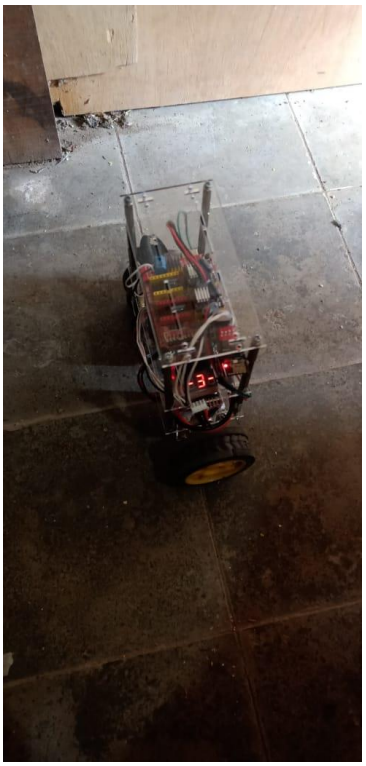


Foto Robot Keseimbangan di Titik Awal (0, 0)