

## **SKRIPSI**

### **RANCANG BANGUN SISTEM NAVIGASI ROBOT OTONOM *WAITER-BOT BERBASIS ROBOT OPERATING SYSTEM***

**Disusun dan diajukan oleh:**

**ABD. SALAM**

**D041191099**



**PROGRAM STUDI SARJANA TEKNIK ELEKTRO  
FAKULTAS TEKNIK  
UNIVERSITAS HASANUDDIN  
GOWA  
2023**

## **LEMBAR PENGESAHAN SKRIPSI**

### **RANCANG BANGUN SISTEM NAVIGASI ROBOT OTONOM *WAITER-BOT* BERBASIS *ROBOT OPERATING SYSTEM***

Disusun dan diajukan oleh

**ABD. SALAM**

**D041191099**

Telah dipertahankan di hadapan Panitia Ujian yang dibentuk dalam rangka

Penyelesaian Studi Program Sarjana Program Studi Teknik Elektro

Fakultas Teknik Universitas Hasanuddin

Pada tanggal

Dan dinyatakan telah memenuhi syarat kelulusan

Menyetujui,

Pembimbing Utama,

Pembimbing Pendamping,

Muh Anshar, ST., M.Sc (Research), Ph.D.  
NIP. 197708172005011003

Prof. Dr. Ir. Andani Achmad, MT.  
NIP. 196012311987031022

Ketua Program Studi,

Dr.Eng. Ir. Dewiani, MT.  
NIP. 196910261994122001

## **LEMBAR PENGESAHAN SKRIPSI**

### **RANCANG BANGUN SISTEM NAVIGASI ROBOT OTONOM *WAITER-BOT* BERBASIS *ROBOT OPERATING SYSTEM***

Disusun dan diajukan oleh

**ABD. SALAM**

**D041191099**

Telah memenuhi syarat untuk melakukan Seminar Hasil

Pada tanggal 22 September 2023

Menyetuui,

Pembimbing Utama,

Pembimbing Pendamping,

Muh Anshar, ST., M.Sc (Research), Ph.D.  
NIP. 197708172005011003

Prof. Dr. Ir. Andani Achmad, MT.  
NIP. 196012311987031022

Ketua Program Studi,

Dr.Eng. Ir. Dewiani, MT.  
NIP. 196910261994122001

## **PERNYATAAN KEASLIAN**

Yang bertanda tangan dibawah ini:

Nama : Abd. Salam

NIM : D041191099

Program Studi : Teknik Elektro

Jenjang : S1

Menyatakan dengan ini bahwa karya tulisan saya berjudul

### **RANCANG BANGUN SISTEM NAVIGASI ROBOT OTONOM *WAITER-BOT* BERBASIS *ROBOT OPERATING SYSTEM***

Adalah karya tulisan saya sendiri dan bukan merupakan pengambilan alihan tulisan orang lain dan bahwa skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri.

Semua informasi yang ditulis dalam skripsi yang berasal dari penulis lain telah diberi penghargaan, yakni dengan mengutip sumber dan tahun penerbitannya. Oleh karena itu semua tulisan dalam skripsi ini sepenuhnya menjadi tanggung jawab penulis. Apabila ada pihak manapun yang merasa ada kesamaan judul dan atau hasil temuan dalam skripsi ini, maka penulis siap untuk diklarifikasi dan mempertanggungjawabkan segala resiko.

Segala data dan informasi yang diperoleh selama proses pembuatan skripsi, yang akan dipublikasi oleh Penulis di masa depan harus mendapat persetujuan dari Dosen Pembimbing.

Apabila dikemudian hari terbukti atau dapat dibuktikan bahwa sebagian atau keseluruhan isi skripsi ini hasil karya orang lain, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Gowa, 22 September 2023

Yang Menyatakan

Abd. Salam

## **ABSTRAK**

**ABD. SALAM.** *Rancang Bangun Sistem Navigasi Robot Otonom Waiter-Bot Berbasis Robot Operating System (ROS)* (dibimbing oleh Muh Anshar dan Andani Ahmad)

Perkembangan teknologi robotika semakin masif diterapkan di industri manufaktur seiring memasuki era revolusi industri 5.0 dalam membantu meringankan beban manusia serta meningkatkan efisiensi waktu dan tenaga. Salah satu bentuk penerapannya pada penelitian robot otonom *Waiter-Bot* sebagai robot cerdas berbasis *Robot Operating System (ROS)* dengan misi pengantaran logistik secara *autonomus*. Penelitian ini menggunakan metode kuantitatif dengan pengujian robot melakukan perencanaan jalur navigasi dalam mencapai misi. Metode perencanaan jalur navigasi terdiri atas perencanaan jalur lokal (*local planner*) dan jalur global (*global planner*) untuk memperoleh jalur trayektori lintasan paling terdekat dan teraman. Pengujian *global planner* dilakukan dengan menggunakan dua skenario. Skenario 1, robot melakukan navigasi dengan kemampuan mengenali batas jalur lintasan. Adapun skenario 2, robot diberikan perlakuan penambahan objek diluar hasil pemetaan untuk mengenali dan menghindari *obstacle* tersebut. Pengujian *local planner* menggunakan algoritma *Dynamic Window Approach (DWA)* untuk mengetahui kemampuan robot merencanakan opsi jalur baru dalam kondisi situasional agar tetap bergerak otonom untuk mencapai target misi. Hasil yang diperoleh dari penelitian ini yakni pada pengujian *global planner*, waktu tempuh rata-rata saat robot melewati misi A,B,C dengan jarak tempuh 12,02 meter pada skenario 1 yakni 3,81 menit serta skenario 2 dengan waktu tempuh rata-rata 4 menit 12 detik. Sedangkan pengujian *local planner*, diperoleh waktu transisi pivot dengan rata-rata 73 detik. Dalam sistem navigasi robot berbasis ROS pada penelitian ini diketahui jumlah *node* yang aktif yakni 14 *node* dalam 8 topik. Dengan demikian navigasi *Waiter-Bot* dapat menjalankan misi dengan efektif.

Kata Kunci: robot, navigasi, otonom, ROS, *local planner*, *global planner*

## ABSTRACT

**ABD. SALAM.** *Design of Navigation System for Waiter-Bot Autonomous Robot Based on Robot Operating System (ROS) (supervised by Muh Anshar and Andani Ahmad)*

The development of robotics technology is increasingly massively applied in the manufacturing industry as it enters the era of the Industrial Revolution 5.0 in helping to ease the burden on humans and increase time and energy efficiency. One form of application is in the research of the *Waiter-Bot* autonomous robot as a Robot Operating System (ROS)-based intelligent robot with an autonomous logistics delivery mission. This research uses a quantitative method by testing the robot to plan the navigation path in achieving the mission. The navigation path planning method consists of local path planning (local planner) and global path (global planner) to obtain the closest and safest trajectory path. Testing the global planner is done using two scenarios. Scenario 1, The robot navigates with the ability to recognize the boundaries of the trajectory path. As for scenario 2, the robot is given the treatment of adding objects outside the mapping results to recognize and avoid these obstacles. Local planner testing uses the Dynamic Window Approach (DWA) algorithm to determine the robot's ability to plan new path options in situational conditions to keep moving autonomously to achieve mission targets. The results obtained from this research are in global planner testing, the average travel time when the robot passes missions A, B, C with a distance of 12.02 metres in scenario 1 is 3.81 minutes and scenario 2 with an average travel time of 4 minutes 12 seconds. While testing the local planner, the pivot transition time is obtained with an average of 73 seconds. This study's ROS-based robot navigation system shows that the number of active nodes is 12 nodes in 6 topics. Thus the *Waiter-Bot* navigation can carry out the mission effectively.

Keywords: robot, navigation, autonomous, ROS, local planner, global planner

## DAFTAR ISI

LEMBAR PENGESAHAN SKRIPSI.....	2
LEMBAR PENGESAHAN SKRIPSI.....	3
PERNYATAAN KEASLIAN.....	4
ABSTRAK.....	5
ABSTRACT.....	6
DAFTAR ISI.....	i
DAFTAR GAMBAR .....	iv
DAFTAR TABEL.....	vi
DAFTAR LAMPIRAN .....	vii
KATA PENGANTAR .....	viii
BAB I PENDAHULUAN .....	1
1.1 Latar Belakang .....	1
1.2 Rumusan Masalah .....	2
1.3 Tujuan Penelitian.....	2
1.4 Manfaat Penelitian.....	3
1.5 Batasan Masalah.....	3
1.6 Metode Penelitian.....	3
1.7 Sistematika Penulisan .....	4
BAB II TINJAUAN PUSTAKA.....	5
2.1 Penelitian Terkait .....	5
2.1.1 Implementasi Sistem Robot Otonom dengan Sensor Kinect menggunakan <i>Algoritme Gmapping dan Timed Elastic Band</i> .....	5
2.1.2 Sistem Navigasi <i>Waypoint</i> Pada <i>Autonomous Mobile Robot</i> .....	5
2.1.3 Navigasi Robot Otonom Berbasis ROS Menggunakan Sensor Lidar 2D dan Kamera RGB-D.....	6
2. 2 Sistem Kendali .....	6
2.2.1 Sistem <i>Open Loop</i> .....	7
2.2.2 Sistem <i>Close Loop</i> .....	7
2.3 Probabilistik Dalam Robotika.....	7
2.4 Robot Otonom .....	9

2.5 <i>Robot Operating System (ROS)</i> .....	9
2.5.1 <i>ROS Package</i> .....	10
2.5.2 <i>ROS Node</i> .....	10
2.5.3 <i>ROS Topic</i> .....	11
2.6 Navigasi Robot Berbasis ROS.....	12
2.6 Perencanaan Jalur Navigasi .....	14
2.6.1 <i>Navigation Stack</i> .....	14
2.6.2 <i>Local Planner</i> .....	15
2.6.3 <i>Global Planner</i> .....	17
2.6.4 <i>Local Costmap</i> .....	18
2.6.5 <i>Global Costmap</i> .....	18
2.7 <i>Motion Control</i> .....	19
2.8 <i>Differential Drive Mobile Robot</i> .....	20
2.9 Aktuator.....	22
2.10 Kinematika dan <i>Trajectory Tracking</i> .....	23
BAB III METODE PENELITIAN .....	24
3.1 Waktu dan Lokasi Penelitian .....	24
3.2 Rancangan Umum Penelitian .....	24
3.3 Rancangan Perangkat Keras .....	27
3.3.1 Mekanisasi Robot.....	27
3.3.3 Perancangan Subsistem Proses .....	32
3.4 Rancangan Perangkat Lunak .....	38
3.4.1 Instalasi ROS.....	38
3.4.2 <i>Autonomous Movement</i> .....	39
3.4.3 <i>Navigation Stack</i> .....	40
3.5 Rancangan Pengujian .....	45
3.5.1 Perencanaan <i>Local Planner</i> .....	45
3.5.2 Perencanaan <i>Global Planner</i> .....	48

3.5.3 Skenario Pengujian Navigasi Robot .....	50
BAB IV ANALISIS DAN PEMBAHASAN .....	53
4.2 Pengujian Respon Kecepatan Robot Terhadap Objek Penghalang .....	55
4.3 Pengujian Skenario Navigasi Robot Pada Lingkungan Statis dan Dinamis ....	57
4.3.1 Kemampuan <i>Waiter-Bot</i> melakukan Navigasi Pada Lingkungan Statis.....	58
4.3.2 Kemampuan <i>Waiter-Bot</i> melakukan Navigasi Pada Lingkungan Dinamis.....	58
4.4 Sistem Komunikasi <i>Node</i> Pada <i>Navigation Stack</i> Dalam ROS.....	61
4.4.1 Data Status Komunikasi <i>Node</i> Robot Saat Tidak Bergerak.....	61
4.4.2 Data Status Komunikasi <i>Node</i> Robot Saat Berjalan .....	61
BAB V KESIMPULAN DAN SARAN .....	63
5.1 Kesimpulan.....	63
5.2 Saran.....	64

## DAFTAR GAMBAR

Gambar 1 Sistem open loop .....	7
Gambar 2 Sistem close loop.....	7
Gambar 3 Contoh robot otonom; robot pelayan .....	9
Gambar 4 Blok diagram sistem komunikasi ROS .....	11
Gambar 5 Hirarki navigasi robot otonom .....	12
Gambar 6 Visualisasi ide dasar algoritma DWA .....	16
Gambar 7 Sistem navigasi ROS menggunakan algoritma DWA.....	17
Gambar 8 Diagram alir rancangan penelitian .....	25
Gambar 9 Sistem navigasi Waiter-Bot berbasis ROS.....	25
Gambar 10 Diagram blok sistem.....	26
Gambar 11 Diagram blok masukan.....	26
Gambar 12 Diagram blok proses.....	26
Gambar 13 Tampilan eksisting Waiter-Bot .....	27
Gambar 14 Blok umum diagram komponen elektronika.....	29
Gambar 15 Blok diagram subsistem komponen masukan.....	29
Gambar 16 Kinect memberikan tiga output: IR, RGB, projector .....	30
Gambar 17 Invers Kinect pada fungsi kedalaman asli.....	30
Gambar 18 Blok diagram modul sensor IMU MPU-60X0 .....	31
Gambar 19 Sistem kerja encoder .....	32
Gambar 20 Board Arduino Mega 2560.....	33
Gambar 21 Intel Core i9 Gen 12 .....	34
Gambar 22 Motor DC Power Window 12 V .....	36
Gambar 23 Blok diagram aktuator robot .....	36
Gambar 24 Instalasi keseluruhan perangkat keras robot.....	37
Gambar 25 Hirarki Sistem Navigasi Waiter-Bot .....	40
Gambar 26 Blok Diagram <i>Autonomous Movement</i> .....	40
Gambar 27 Flow chart perencanaan local planner.....	46
Gambar 28 Visual Dynamic Window Approach .....	47
Gambar 29 Flow chart pengujian global planner.....	50
Gambar 30 Desain denah pengujian navigasi robot pada lingkungan statis .....	51

Gambar 31 Desain denah pengujian navigasi robot pada lingkungan dinamis ...	52
Gambar 32 Peta pengujian navigasi robot .....	57
Gambar 33 Waiter-Bot membuat jalur trajektori navigasi dalam ROS .....	57
Gambar 34 Navigasi robot pada skenario 1 .....	58
Gambar 35 Pengujian navigasi robot pada skenario 2 lingkungan statis.....	53
Gambar 36 Peta pengujian navigasi robot .....	57
Gambar 37 <i>Waiter-Bot</i> membuat jalur trayektori navigasi dalam ROS .....	57
Gambar 38 Navigasi robot pada skenario lingkungan statis.....	58
Gambar 39 Uji sistem navigasi Waiter-Bot pada lingkungan dinamis .....	59
Gambar 40 Pengujian navigasi robot pada skenario 2 lingkungan dinamis .....	55
Gambar 41 sistem komunikasi node pada skenario 2 lingkungan dinamis .....	55
Gambar 42 Grafik kecepatan linear uji skenario 2 lingkungan dinamis .....	55
Gambar 43 Grafik kecepatan angular uji skenario 2 lingkungan dinamis .....	56
Gambar 44 Komunikasi node saat robot tidak bergerak .....	61
Gambar 45 Komunikasi node saat robot bergerak.....	61

## **DAFTAR TABEL**

Tabel 1 Komponen utama rancang bangun robot otonom <i>Waiter-Bot</i> .....	28
Tabel 2 Spesifikasi Arduino Mega 2560.....	33
Tabel 3 Spesifikasi Intel Core i9 Generasi 12.....	35
Tabel 4 Komponen elektronika kendalian low level.....	38
Tabel 5 Pengujian skenario lingkungan statis navigasi <i>Waiter-Bot</i> .....	58
Tabel 6 Pengujian skenario 1 navigasi <i>Waiter-Bot</i> misi A .....	58
Tabel 7 Pengujian skenario 1 navigasi <i>Waiter-Bot</i> misi B .....	58
Tabel 8 Pengujian skenario 1 navigasi <i>Waiter-Bot</i> misi C .....	59
Tabel 9 Pengujian skenario 2 navigasi <i>Waiter-Bot</i> misi A .....	60
Tabel 10 Pengujian skenario 2 navigasi <i>Waiter-Bot</i> misi B .....	60
Tabel 11 Pengujian skenario 2 navigasi <i>Waiter-Bot</i> misi C .....	60
Tabel 12 Data status komunikasi node saat robot berjalan .....	62

## **DAFTAR LAMPIRAN**

Lampiran 1 Data sampling perhitungan kecepatan robot .....	67
Lampiran 2 Dokumentasi pelaksanaan penelitian.....	70
Lampiran 2 Kode program <i>Waiter-Bot</i> .....	76

## KATA PENGANTAR

Puji syukur terpanjatkan kehadirat Allah *subhanahu wata'ala* atas limpahan rahmat dan karunia-Nya sehingga penyusunan skripsi tugas akhir ini dapat terselesaikan. Shalawat serta salam tak lupa tercurahkan kepada baginda Rasulullah *sallallahu 'alaihi wasallam*. Penyelesaian skripsi ini merupakan upaya penulis dalam memenuhi salah satu syarat guna memperoleh gelar Sarjana Teknik di Departemen Teknik Elektro Fakultas Teknik Universitas Hasanuddin.

Skripsi ini penulis persembahkan kepada seluruh pembaca dan pengembang riset selanjutnya sebagai literatur studi Pustaka. Kepada yang terkhusus untuk kedua orang tua dan kakak penulis yang telah mendidik dan senantiasa selalu mendoakan dan mendukung penulis selama ini. Semoga ini menjadi salah satu bentuk *wasilah birrul walidayn* kepada orang tua dan seuruh keluarga penulis.

Skripsi ini berjudul Rancang Bangun Sistem Navigasi Robot Otonom *Waiter-Bot* Berbasis *Robot Operating System*. Pelaksanaan rangkaian penelitian dan penyusunan skripsi ini juga tak terlepas dari seluruh bantuan, bimbingan, nasehat dan doa dari berbagai pihak. Sehubungan dengan hal tersebut, maka penulis hendak mengucapkan terima kasih kepada:

1. Bapak Dr. Amil Ahmad Ilham, S.T., M.IT. selaku Wakil Dekan Bidang Akademik dan Kemahasiswaan Fakultas Teknik yang berkenan memberikan arahan dan secara personal banyak memberikan perhatian dan dukungan kepada penulis.
2. Ibu Dr. Eng. Ir. Dewiani,MT. selaku ketua Departemen Teknik Elektro atas izin dan dukungan moril dan materil dalam pelaksanaan penelitian penulis.
3. Bapak Muh Anshar, ST., M.Sc(Research), Ph.D. selaku Pembimbing I atas bimbingan dan segala bentuk perhatian dan dorongan kepada penulis, serta Prof. Dr. Ir. Andani Achmad, S.T., M.T., selaku Pembimbing II dengan kerendahan hati mempercayakan penulis sebagai mahasiswa bimbingan penyelesaian tugas akhir ini.
4. Para dosen penguji, Ibu Dr. Andi Ejah Umraeni Salam, S.T., M.T. dan Ibu Ida Rachmaniar Sahali, ST., M.T. atas segala saran dan masukan kepada penulis dalam menyelesaikan tugas akhir ini.

5. Seluruh tenaga pendidik dan sivitas akademika Departemen Teknik Elektro dan Fakultas Teknik Unhas atas segala bentuk ketulusan pelayanan selama penulis menempuh perkuliahan.
6. Penasehat akademik penulis, Prof. Baharuddin Hamzah, Prof. Wihardi Tjaronge, Bapak Muh. Ramli, Bapak Amijoyo Muchtar, Bapak Aries Subagiyo dan Abah Nusran atas segala nasehat, kasih sayang dan pelajaran hidup kepada penulis.
7. Penasehat karir dan pengembangan diri, Bapak Mukti Ali, Bapak Muhammad Rusman, Bapak Yusran, Pak Faisal Mahmuddin, Ibu Intan Sari Areni dan Ibu Nadzirah Ikasari atas segala bentuk keikhlasan, dedikasi dan berkenan membimbing penulis dalam berproses dan bertumbuh selama ini.
8. *The honorable mention* sahabat penulis, Fariz, Hasbih, Arya, Iqrimah, Ocan, Dennis dan ikhwah MADZ 2019 serta grup riset *Cognitive Social Robotics and Advanced Artificial Intelligence Centre* (CSR-2AIR) atas segala bentuk solidaritas dan kekeluargaan selama perkuliahan dan pekerjaan proyek.
9. Keluarga Yayasan Al Fityah, *Super Team COV.id*, Ikhwah Al Muhandis, MADZ DE FT-UH dan jamaah Masjid Al Ilmi IKATEK Unhas atas kepercayaan kepada penulis serta menjadi wadah berproses dan bertumbuh.
10. Teman seperjuangan TR19GER, Teknik Elektro Angkatan 2019 yang telah membersamai penulis selama menempuh perkuliahan. Serta para senior dan junior yang tak dapat penulis sebutkan satu-persatu. *Barakallahu fiikum.*

Penulis menyadari bahwa masih terdapat banyak kekurangan dalam skripsi ini, oleh karenya segala bentuk saran dan masukan yang membangun dari semua pihak. Akhir kata penulis berharap semoga skripsi dapat diterima sebagai aktualisasi tri dharma penulis yang dapat memberikan manfaat bagi penulis dan pembacanya.

Gowa, 13 September 2023

Penulis,

Abd. Salam

## BAB I

### PENDAHULUAN

#### 1.1 Latar Belakang

Perkembangan teknologi seiring dengan tuntutan kebutuhan hidup akan pelayanan berubah, kebutuhan serta kesibukan juga meningkat. Pekerjaan manusia dituntut lebih efisien dalam menggunakan waktu untuk memenuhi kebutuhan hidupnya sehingga dapat lebih produktif untuk bekerja. Implementasi robot di industri komersial dan banyak perusahaan lain telah bahwa pengembangan robotika sosial di masa depan sangat potensial. Hal ini membuat banyak usaha jasa yang menyediakan pelayanan dalam pemenuhan kebutuhan manusia bermunculan contohnya usaha restoran. Di sebuah restoran terdapat pelayan yang akan memberikan pelayanan kepada pelanggan. Kinerja pelayan manusia dipengaruhi oleh banyak faktor, antara lain manusia yang masih memiliki rasa lelah dan letih sehingga tidak dapat bekerja secara terus-menerus. Manusia dapat jatuh sakit sehingga tidak dapat melaksanakan tugasnya. Begitupun dalam pelayan rumah sakit telah dikembangkan robot yang terdiri dari robot *Telemedicine (TEL)*, *robot Disinfection (DIS)*, dan *robot Assistance (ASL)*, robot dengan jenis *Assistance (ASL)* akibat kebutuhan akan efektifitas pelayanan yang optimal.)

Dalam sektor industri tercatat sekitar 422.000 unit robot di seluruh dunia dan diperkirakan akan terus bertambah sebanyak 12% per tahun hingga tahun 2022 (IFR, 2018). *Autonomous mobile robot* merupakan salah satu jenis robot yang berkembang saat ini. Teknologi yang dapat diterapkan untuk membantu dan meringankan pekerjaan pelayanan restoran adalah dengan memanfaatkan kecanggihan robot cerdas. Robot ini harus memiliki kemampuan untuk membawa logistik, seperti mengantar makanan, barang, baju, dan/atau obat-obatan (Sierra Marín dkk., 2021). Permasalahan yang dihadapi sebuah *autonomous mobile robot* adalah masalah navigasi karena robot otonom harus mengenali lingkungannya. Oleh karena itu, penelitian ini mengangkat studi rancang bangun *Smart Waiter-Bot* sebagai robot cerdas yang dirancang untuk dapat melakukan pengantaran makanan secara otonom pada titik tujuan dalam lingkungan dalam ruangan. Sistem robot yang dibangun terdiri dari beberapa komponen diantaranya kamera kinect, sensor *encoder*, MPU-90, dan HCSR yang dijalankan berbasis *middleware Robot*

*Operating System* (ROS). ROS digunakan karena setiap grup riset robotika umumnya berfokus pada topik-topik riset robot tertentu sehingga ROS hadir dalam memudahkan pengembangan suatu topik riset tertentu tanpa harus memikirkan topik riset lainnya. ROS mencakup *tools*, *libraries* dan *convention* yang bertujuan untuk menciptakan serta mengembangkan robot yang kompleks. Dengan begitu grup riset yang mengembangkan sistem navigasi tidak perlu harus memikirkan pula sistem biomekanik dari robot tersebut melainkan hanya mengembangkan salah satu bagian dari sistem navigasi tersebut tanpa harus membuat sistem navigasi dari dasar. Dalam pengembangan sistem navigasi *Waiter-Bot* secara otonom digunakan metode perencanaan jalur salah satunya dengan metode algoritma *Dynamic Window Approach* (DWA). Selain itu, diperlukan pula sebuah peta yang presisi agar dapat melakukan navigasi, sehingga robot perlu membangun peta dan lokalisasi lingkungannya terlebih dahulu.

## 1.2 Rumusan Masalah

Adapun rumusan masalah dalam penelitian ini sebagai berikut:

- a. Bagaimana deviasi dan respon kecepatan robot otonom dalam mengeksekusi perintah nilai kecepatan pada *ROS Navigation Stack* untuk melakukan navigasi?
- b. Bagaimana kemampuan navigasi robot otonom *Waiter-Bot* dalam skenario pengujian lingkungan statis dan dinamis?
- c. Bagaimana sistem komunikasi antar node dalam sisem navigasi robot otonom *Waiter-Bot* berbasis *Robot Operating System* (ROS)?

## 1.3 Tujuan Penelitian

Adapun tujuan penelitian ini sebagai berikut:

- a. Mengetahui deviasi dan respon kecepatan robot otonom dalam mengeksekusi perintah nilai kecepatan pada *ROS Navigation Stack* untuk melakukan navigasi.
- b. Mengetahui kemampuan navigasi robot otonom *Waiter-Bot* dalam skenario pengujian lingkungan statis dan dinamis.
- c. Mengetahui sistem komunikasi antar node dalam sisem navigasi robot otonom *Waiter-Bot* berbasis *Robot Operating System* (ROS).

## 1.4 Manfaat Penelitian

Dari penelitian ini, diarapkan dapat memperoleh manfaat sebagai berikut:

- a. Menjadi referensi penelitian pengembangan robot otonom berbasis ROS.
- b. Memberikan wawasan baru bagi para pengembang robot dalam memanfaatkan *framework* ROS untuk topik riset kendali dan robotika modern.
- c. Merepresentasi perkembangan teknologi robotika dalam implementasi di berbagai sektor otomasi industri.

## 1.5 Batasan Masalah

Agar penelitian ini lebih terarah dan terukur, maka difokuskan dalam ruang lingkup sebagai berikut:

- a. Penelitian ini merancang mekanik robot otonom *Waiter-Bot* bagian *low level* dan *high level*.
- b. Pengujian sistem navigasi robot otonom *Waiter-Bot* dilakukan pada lingkungan *indoor*.
- c. Penelitian ini fokus pada pengujian sistem navigasi robot otonom *Waiter-Bot* bagian *navigation stack* berbasis ROS.
- d. Pengujian kemampuan sistem navigasi *Waiter-Bot* terbatas dalam pemberian perintah target melalui PC pada RViz.

## 1.6 Metode Penelitian

Metode penelitian yang dilakukan pada tugas akhir ini adalah:

### 1. Studi literatur

Pada tahapan ini yang dilakukan adalah mencari sumber-sumber referensi dan beberapa materi pendukung yang dijadikan sebagai landasan dalam penelitian ini, sebelum melakukan penerapan dan pengujian sistem secara langsung.

### 2. Pengujian

Pada tahap pengujian ini yang dilakukan adalah penerapan dan pengujian langsung berbasis simulasi menggunakan beberapa skenario dengan tujuan untuk mendapatkan data yang tepat dari hasil pengamatan.

### 3. Analisis Data

Analisis data dilakukan dengan tujuan untuk melihat data yang diperoleh dari pengujian sistem navigasi robot otonom dengan menggunakan beberapa skenario pengujian.

4. Diskusi dan Konsultasi

Diskusi dan konsultasi dilakukan secara daring maupun luring kepada dosen pembimbing maupun pihak-pihak yang berkompeten dibidangnya untuk mendapatkan pengetahuan mengenai penelitian yang dilaksanakan.

5. Simpulan

Simpulan merupakan tahap akhir dari penelitian ini yang diperoleh setelah dilakukan pengambilan dan analisa data mengenai semua permasalahan yang telah dibahas.

### **1.7 Sistematika Penulisan**

Adapun sistematika penulisan laporan Tugas Akhir ini adalah sebagai berikut:

**BAB I PENDAHULUAN**

Bab ini berisi tentang latar belakang, rumusan masalah, tujuan penelitian, batasan masalah, manfaat penelitian, metode penelitian, dan sistematika penulisan laporan.

**BAB II TINJAUAN PUSTAKA**

Bab ini berisi teori-teori penunjang terkait penelitian yang dilakukan yang diambil dari berbagai sumber ilmiah yang digunakan dalam penulisan laporan tugas akhir ini.

**BAB III METODOLOGI PENELITIAN**

Bab ini membahas mengenai rancangan penelitian, waktu dan tempat penelitian, diagram alir perencanaan, dan alat dan bahan yang digunakan pada penelitian tugas akhir ini.

**BAB IV HASIL DAN PEMBAHASAN**

Bab ini membahas mengenai pengujian dan performa navigasi *Waiter-Bot* secara otonom dengan pendekatan jalur lokal dan jalur global.

**BAB V PENUTUP**

Bab ini berisi kesimpulan terhadap hasil pengujian dan saran untuk penelitian selanjutnya.

## BAB II

### TINJAUAN PUSTAKA

#### 2.1 Penelitian Terkait

##### 2.1.1 Implementasi Sistem Robot Otonom dengan Sensor Kinect menggunakan *Algoritme Gmapping dan Timed Elastic Band*

Penelitian ini mengangkat studi kasus pelayan rumah sakit menggunakan teknologi robot. Atas hal ini diperlukan sistem robot bergerak otonom pada rumah sakit untuk membantu tenaga kerja, terutama dalam hal logistik. Sistem terdiri dari beberapa komponen diantarnya sensor kinect, *encoder*, IMU, dan dengan middleware *Robot Operating System* (ROS). Keunggulan pada penelitian ini adalah bentuk dari purwarupa robot yang memiliki ukuran yang besar. Untuk menciptakan navigasi secara otonom diperlukan metode perencanaan jalur, maka dari itu digunakan algoritme *Timed Elastic Band (TEB)*. Sebelum melakukan navigasi, diperlukan sebuah peta yang presisi, maka dari itu dibutuhkan metode pembuatan peta menggunakan algoritme GMapping. Hasil uji pada navigasi menunjukkan akurasi sebesar 50% dari 10 percobaan yang telah berhasil 5 kali. Navigasi dilakukan dengan waktu komputasi algoritme TEB 2,77 detik dan rata-rata waktu pergerakan robot yang berhasil menuju koordinat tujuan adalah 129,98 detik. Untuk pemetaan didapatkan akurasi sebesar 50% dengan akurasi ukuran peta sebesar 98,84% (Hamdan, dkk., 2022).

##### 2.1.2 Sistem Navigasi *Waypoint* Pada *Autonomous Mobile Robot*

Dalam penelitian Ahmad Sulkhan Taufik (2013) dirancang sistem navigasi luar ruang berbasis posisi dengan metode *waypoint*. Sistem navigasi diterapkan pada *autonomous mobile robot* yang bergerak di darat. Sistem navigasi dirancang agar *autonomous mobile robot* mampu mengenali posisi dan arah berdasarkan sistem koordinat bumi, mampu melakukan koreksi arah gerak (*bearing correction*) dan memperkirakan jarak yang telah ditempuh (odometer) untuk meningkatkan akurasi dalam mencapai posisi tujuan, dengan rute yang telah ditentukan oleh operator. Modul GPS *receiver* digunakan sebagai penentu posisi, sedangkan modul *magnetic compass* digunakan sebagai penentu arah dalam sistem navigasi. Hasil pengujian menunjukkan bahwa sistem navigasi *waypoint* mampu mengatur gerak

*autonomous mobile robot* dalam mencapai posisi tujuan dengan akurasi sebesar 11 meter, serta mampu melakukan odometer dengan akurasi sebesar 0,5 meter.

### **2.1.3 Navigasi Robot Otonom Berbasis ROS Menggunakan Sensor Lidar 2D dan Kamera RGB-D**

Penelitian ini berisi tentang implementasi robot bergerak otonom dengan *Robot Operating System (ROS)*. Sistem ini menggunakan kamera 2D LiDAR dan RGB-D dengan ROS 2D *navigation stack*, dengan konsumsi daya yang rendah dan komputer onboard yang murah. Untuk perangkat lunak, digunakan paket ROS resmi dengan perubahan parameter *default* yang minimal. Untuk perangkat keras, keterbatasan peralatan dan pengaturan sistem adalah salah satu tantangan. Sistem yang dibangun bertujuan agar robot dapat melakukan navigasi dengan kemampuan penghindaran rintangan yang dinamis. Fokus pengujian dalam penelitian ini yakni dua pengaturan sistem dari tumpukan navigasi ROS yang diusulkan. Sistem pertama diimplementasikan pada *Raspberry Pi 3* dengan menggunakan LiDAR 2D saja. Sistem kedua diimplementasikan pada Intel NUC menggunakan LiDAR 2D dan kamera RGB-D. Untuk mengevaluasi kinerja, pengujian fungsional dilakukan dalam beberapa percobaan. Hasil percobaan tersebut menunjukkan bahwa robot dapat menghindari objek yang menghalangi atau berhenti jika tidak dapat dihindari (Sukkpranhachai, dkk., 2019).

## **2. 2 Sistem Kendali**

Sistem Kendali atau *control system* terdiri dari dua kata yaitu *system* dan *control*. Sistem berasal dari bahasa latin (*systema*) dan bahasa Yunani (*sustema*) adalah suatu kesatuan yang terdiri dari komponen atau elemen yang dihubungkan bersama untuk mencapai suatu tujuan tertentu. *Control* itu memiliki arti mengatur, mengarah dan mengendalikan. Jadi *system control* adalah hubungan antara komponen-komponen fisik yang membentuk suatu konfigurasi sistem sehingga memberikan hasil yang diharapkan yang dapat dipraktekkan secara otomatis. Adapun jenis-jenis sistem kendali terdiri atas dua macam yakni sistem kendali kalang terbuka (*open loop*) dan kalang tertutup (*close loop*).

### 2.2.1 Sistem *Open Loop*

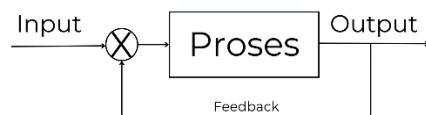
Sistem kendali *open loop* adalah sebuah sistem yang tidak memiliki umpan balik (*feedback*), sehingga bila terdapat gangguan dari dalam maupun dari luar maka sistem tidak dapat melaksanakan tugas seperti yang diharapkan. Suatu sinyal masukan diberikan ke dalam sistem kendali dimana keluarannya bertindak sebagai sinyal penggerak dimana sinyal penggerak ini yang kemudian menghasilkan proses yang akan dikendalikan untuk menghasilkan *output* yang diinginkan. Proses *open loop* dapat dilihat pada Gambar 1.



Gambar 1 Sistem *open loop*

### 2.2.2 Sistem *Close Loop*

Sistem kendali *close loop* adalah sistem kontrol yang memiliki *feedback*, berbeda dengan sistem *open loop*. Pada bagian *output* dari sistem kontrol ini akan dikirim kembali untuk dibandingkan dengan input yang diberikan. Bila masih terdapat selisih antara *output* dan input, maka sistem masih memiliki eror. Eror merupakan selisih antara input dan *output* atau sistem belum mencapai hasil yang diinginkan. Proses kerja dari sistem *close loop* dapat dilihat pada Gambar 2.



Gambar 2 Sistem *close loop*

## 2.3 Probabilistik Dalam Robotika

Dalam *Probabilistic Robotics* berdasarkan peneliti Inggris, Thrun, *et.al.* (2005) memberikan penjelasan mengenai ketidakpastian (*uncertainty*) dalam dunia robotika meliputi:

### 1. Lingkungan

Pada proses eksplorasi lingkungan yang dilakukan oleh robot. lingkungan dianggap bersifat dinamik, sangat luas dan tidak dapat diprediksi. Komposisi alam misalnya tumbuhan, angin, air, manusia menjadi satu kondisi yang memberikan permasalahan dinamik bagi robot untuk memahami lingkungan tersebut.

## 2. Sensor.

Sensor atau indra bagi robot juga bersifat dinamik. Kita tidak dapat menganggap bahwa sensor untuk robot adalah ideal (tidak ada eror). Keterbatasan sensor meliputi jangkauan (*range*) dan resolusi (ketepatan / kepresisian) untuk menyatakan satuan dari pengukuran. Sebagai contoh sensor jarak yang mempunyai beam (lebar pancaran) yang bermacam-macam, tentunya galat pengukuran yang kita dapatkan juga beragam. Contoh selanjutnya kamera yang sangat dipengaruhi oleh faktor pencahayaan, resolusi, kedalaman serta *noise-noise* yang berasal dari lingkungan juga mempengaruhi hasil pengukuran.

## 3. Aktuator

Akurasi dari sebuah aktuator juga menjadi penyebab dalam ketidakpastian dalam dunia robotika untuk menyelesaikan suatu permasalahan yang bersifat ideal. Sebagai contoh pada arm robot, motor penggerak dengan derajat gear memberikan pengaruh error letak / posisi dari EoE (*End of Effector*).

## 4. Komputasi.

Robot diharapkan menjadi sistem yang realtime dalam mengolah suatu proses. Tapi dalam kenyataannya fungsi waktu dibutuhkan sehingga mempengaruhi proses yang terjadi. Jika suatu robot mempunyai komputasi yang tinggi maka suatu proses dapat diselesaikan dalam waktu yang cepat. Misalnya robot dengan kemampuan 100 MIPS (*Million Instruction Per Second*) pasti lebih cepat dalam mengolah proses dibandingkan robot dengan kemampuan 10 MIPS (*Million Instruction Per Second*).

Dalam *probabilistics robotics* sejumlah variabel seperti pengukuran sensor, kontrol, dan posisi robot diasumsikan sebagai variabel yang tidak pasti artinya meskipun dalam pengukuran kita mendapatkan nilai, tapi nilai tersebut mempunyai *range* (jangkauan) yang terkadang sewaktu-waktu juga dapat berubah. Meskipun tidak pasti atau tidak dapat diprediksi tetapi nilai tersebut tetap kita butuhkan untuk input sebuah proses. Pada pembahasan ini akan dijelaskan contoh permasalahan yang dihadapi robot pada lingkungan yang bersifat *undeterministic* dan pembahasan tentang simbol dan notasi yang dipakai dalam *probabilistic robotics*.

## 2.4 Robot Otonom

Robot Otonom (*Autonomous Robot*) adalah robot yang dapat melakukan tugas-tugas yang diinginkan dalam lingkungan yang tidak terstruktur tanpa bimbingan manusia terus menerus berdasarkan logika-logika yang diberikan manusia kepada robot. Banyak jenis robot memiliki beberapa tingkat otonomi. Tingkatan otonomi sangat diinginkan dalam bidang-bidang seperti eksplorasi ruang angkasa, membersihkan lantai, memotong rumput, dan pengolahan air limbah (Anggoro, 2013). Salah satu contoh *autonomous robot* adalah *Waiter-Bot* dapat dilihat pada Gambar 3.



Gambar 3 Contoh robot otonom; robot pelayan

*Service robot* merupakan robot yang digunakan untuk melayani kebutuhan manusia sehari-hari. Robot ini digunakan untuk membantu pekerjaan yang kotor, berbahaya, berulang-ulang dan termasuk pekerjaan rumah tangga termasuk melakukan pengantaran paket dan logistik.

## 2.5 Robot Operating System (ROS)

*Robot Operating System* atau biasa disingkat ROS adalah sebuah *middleware* sistem robotika yang berisi berbagai koleksi *software framework* untuk pengembangan perangkat lunak robotik. ROS bukanlah sebuah sistem operasi seperti *Windows*, *MacOS*, *Android* atau sistem operasi lain yang kita kenal. Namun sebuah perangkat lunak yang menyediakan berbagai *service* yang didesain untuk *heterogenous computer cluster* seperti: abstraksi perangkat keras, pengaturan divais level rendah, implementasi fungsionalitas yang sering digunakan dalam robot, penyampaian pesan antar proses dan menejemen *package*. *Software* pada ROS dapat dibagi menjadi tiga grup: tools untuk mendistribusikan perangkat lunak

berbasis ROS, implementasi ROS *client library* seperti *roscpp*, *rospy*, *roslisp*, dan yang terakhir *package* yang berisi kode pemnghubung dengan aplikasi fungsional robot. *Client libraries* ROS utamanya diperuntukkan untuk sistem UNIX. Untuk hal ini Ubuntu Linux terdaftar sebagai “*supported*” sedangkan varian lain seperti *Fedora Linux*, *macOS*, serta *Windows* terdaftar sebagai “*experimental*”. Dalam ROS terdapat beberapa istilah yakni *package*, *node* dan topik (Ilham, 2019).

Di dalam ROS juga memuat *tools* dan *library* yang bisa digunakan untuk mengembangkan berbagai macam program untuk sistem robot (Rahman, 2020). Tujuan penggunaan ROS juga untuk memudahkan para pengembang robot dalam membuat sistem robot yang di inginkan tanpa harus membuat pengkodean dari awal serta dapat mengembangkan kode sumber bersama – sama (Jalil, 2018). ROS dapat dibagi menjadi 3 bagian yaitu level *file systems*, level grafik komputasi dan level komunitas. Di dalam level *file systems* ROS memuat *packages*, *metapackages*, *packages manifests*, *metapackage manifests*, *message* dan *service*. Di dalam level grafik komputasi memuat level *nodes*, *master*, *parameter server*, *message*, *topic*, *services*, dan *bags*. Dan di dalam level komunitas ini juga bisa memisahkan *resources* ROS agar dapat saling bertukar pengetahuan dan perangkat lunak. *Resources* ini terdiri dari distribusi *ROS*, *repository*, *ROS wiki*, *bug ticket system*, *maling list*, *ROS answer and blog* (Jalil, 2019).

### **2.5.1 ROS Package**

Perangkat lunak pada ROS terorganisir di dalam *packages*. Didalam *packages* tersebut dapat berisi ROS *Node*, *library* independen, dataset, file konfigurasi dan hal lain yang mendukung modul tersebut. Tujuan dari *packages* adalah untuk menyediakan fungsionalitas yang mudah digunakan dengan tujuan menjadikan perangkat lunak yang dapat dengan mudah digunakan kembali (*reuse*) oleh komunitas pengguna ROS.

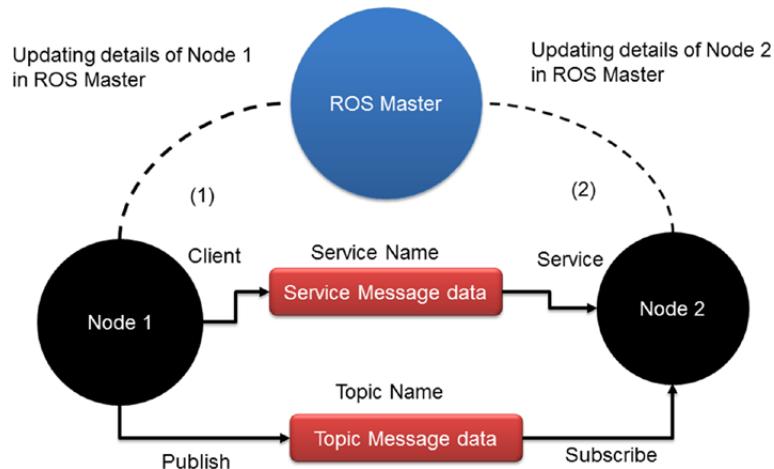
### **2.5.2 ROS Node**

Sebuah *node* dalam ROS adalah sebuah proses yang melakukan suatu komputasi. Berbagai *node* ini terkombinasi bersama menjadi suatu *graph* dan berkomunikasi dari satu *node* ke *node* lain menggunakan aliran *topics*, *services*, dan parameter server. *Node* ini diperuntukkan untuk beroperasi pada level yang sangat sempit. Sebagai contoh sebuah sistem kendali robot biasanya terdiri dari banyak *node*: satu *node* mengendalikan laser *range finder*, satu *node* mengendalikan motor,

satu *node* lokalisasi, satu *node* untuk perencanaan jalur (*path planning*), satu *node* untuk *monitoring* dan seterusnya. Penggunaan *node* ini memberi manfaat antara lain: toleransi kegagalan yang terisolasi pada satu fungsionalitas tertentu pada individu *node*, kompleksitas dari kode berkurang dibanding sebuah sistem *monolithic*, *reusability* serta *Maintainability* tinggi dengan penyederhanaan berdasarkan fungsionalitas-fungsionalitas tersebut. Selain itu sebuah *node* dapat dikembangkan menjadi sebuah *nodelet*, yaitu sebuah *node* yang dapat dipanggil dari dalam kode program lainnya menggunakan modul *nodelet manager*.

### 2.5.3 ROS Topic

*Topic* adalah komponen yang berfungsi sebagai media pertukaran pesan, dalam ROS dikenal dengan istilah *message* yang dinama topik sebagai media antar *node* yang dinamai sesuai isi dari pesan tersebut. Sistemnya menggunakan *anonymous publish/subscribe semantics*. *Node* yang membutuhkan informasi dari suatu *topic* akan melakukan *subscribe* dan *node* yang menyediakan informasi akan melakukan *publish*. Jumlah *node* yang melakukan *subscribe* ataupun *publish* pada suatu *topic* tidak terbatas jumlahnya. Setiap *topic* memiliki tipe datanya masing-masing sesuai tipe ROS *message* yang di bawanya. Kita dapat membuat tipe *message* sendiri sesuai kebutuhan kita, atau menggunakan tipe *message* yang disediakan oleh ROS *client library*. ROS *topic* juga mendukung komunikasi via TCP/IP atau UDP.



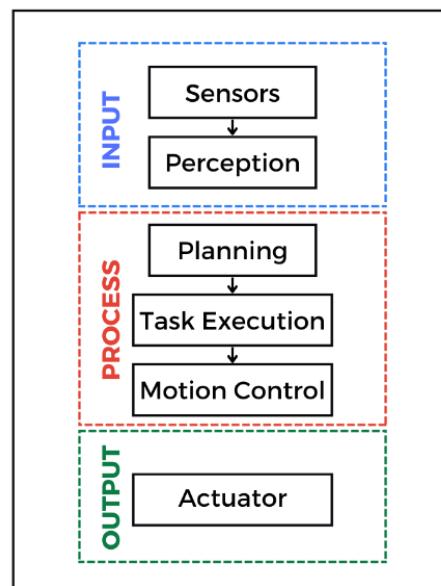
Gambar 4 Blok diagram sistem komunikasi ROS

Pada Gambar 4 menunjukkan dua program yang ditandai sebagai *node* 1 dan *node* 2. Ketika salah satu program dimulai, sebuah *node* berkomunikasi dengan

program ROS yang disebut master ROS. *Node* mengirimkan semua informasinya ke master ROS, termasuk jenis data yang dikirim atau diterima. *Node* yang mengirim data disebut *node publisher* dan *node* yang menerima data disebut *node subscriber*. ROS Master memiliki semua informasi *publisher* dan *subscriber* yang berjalan di komputer. Jika *node* 1 mengirimkan data tertentu yang disebut "A" dan data yang sama diperlukan oleh *node* 2, maka master ROS mengirimkan informasi ke *node* sehingga mereka dapat berkomunikasi satu sama lain.

*Node* pada ROS dapat mengirim berbagai jenis data satu sama lain, yang mencakup tipe data seperti *integer*, *float*, *string*, dan sebagainya. Berbagai tipe data yang dikirim disebut *message* ROS. Dengan pesan ROS, kita dapat mengirim data dengan satu tipe data atau beberapa data dengan tipe data yang berbeda. Pesan-pesan ini dikirim melalui bus pesan atau jalur yang disebut topik ROS. Setiap topik memiliki nama; misalnya, topik bernama "*chatter*" mengirimkan pesan *string*. Ketika sebuah *node* ROS menerbitkan sebuah topik, *node* tersebut mengirimkan topik ROS dengan pesan ROS, dan memiliki data dengan tipe pesan. Pada Gambar 4-12, topik ROS menerbitkan dan berlangganan *node* 1 dan *node* 2. Proses ini dimulai ketika master ROS saling bertukar detail *node* (Joseph, 2018).

## 2.6 Navigasi Robot Berbasis ROS



Gambar 5 Hirarki navigasi robot otonom

Dalam sistem navigasi robot otonom diperlukan proses terlebih dahulu yakni pemetaan dan lokalisasi. Setelah robot berhasil melokalisasi, robot dapat diberikan koordinat tujuan dan menggunakan perencana global untuk merencanakan jalur ke koordinat tersebut. Lingkungan direpresentasikan secara internal sebagai gambar lossless dua dimensi. Lingkungan berisi lima jenis entitas berupa ruang yang belum dijelajahi, ruang kosong yang diketahui, rintangan, radius inflasi, dan robot itu sendiri. Radius inflasi adalah penyangga yang memancar keluar dari semua rintangan. Robot memperlakukan radius inflasi sebagai rintangan dan tidak dapat memetakan jalur yang melaluinya. Radius inflasi sama dengan radius robot itu sendiri dan memiliki efek untuk memastikan bahwa robot selalu menghasilkan jalur dengan ruang yang cukup untuk membersihkan rintangan. Hal ini diimplementasikan sebagai penyangga di sekitar rintangan alih-alih diterapkan pada robot itu sendiri karena alasan kesederhanaan, dan mencapai efek yang sama.

Setelah rencana global dibuat, perencana lokal menerjemahkan jalur ini ke dalam perintah kecepatan untuk motor robot. Perencana lokal melakukan hal ini dengan membuat fungsi nilai di sekitar robot, mengambil sampel dan mensimulasikan lintasan di dalam ruang ini, memberi nilai pada setiap lintasan yang disimulasikan berdasarkan hasil yang diharapkan, mengirimkan lintasan dengan nilai tertinggi sebagai perintah kecepatan ke robot dan mengulanginya hingga tujuan tercapai. Alasan mengapa perencana lokal bekerja dengan cara ini adalah karena perencana lokal ditulis dengan sangat umum untuk digunakan oleh robot yang mungkin memiliki jejak kaki yang tidak beraturan, geometri kemudi *ackerman*, pelengkap, dan konfigurasi lain yang tidak akan bekerja menggunakan algoritma yang dirancang khusus untuk robot dengan bentuk yang sederhana, sistem penggerak diferensial dan tanpa pelengkap. Karena selalu ada sejumlah kecil kesalahan dalam gerakan fisik robot, sangat sulit bagi robot untuk mencapai koordinat tujuannya secara tepat. Hal ini dapat menyebabkan robot berosilasi di sekitar tujuan karena terus berusaha mencapai lokasi yang tepat. Perilaku yang tidak diinginkan ini dapat dielakkan dengan mengatur fungsi *latch* yang menyebabkan robot berhenti ketika berada dalam jarak tertentu dari tujuan (Derect, 2012).

## 2.6 Perencanaan Jalur Navigasi

Dalam konteks robot otonom, perencanaan jalur mengacu pada penentuan jalur yang layak dan efisien untuk diikuti oleh robot saat bergerak dari satu lokasi ke lokasi lain. Hal ini biasanya melibatkan pertimbangan beberapa faktor, seperti lokasi dan orientasi robot saat ini, tata letak lingkungan (termasuk rintangan apa pun yang harus dihindari robot), dan batasan apa pun pada gerakan robot (seperti kecepatan maksimum atau radius belok). Pendekatan spesifik yang digunakan akan bergantung pada karakteristik robot dan lingkungan, serta persyaratan aplikasi. Secara umum, tujuan perencanaan jalur adalah untuk menemukan jalur yang memungkinkan robot mencapai tujuannya secara efisien dan aman sambil menghindari rintangan dan mematuhi batasan apa pun pada gerakannya. Jalur yang dihasilkan oleh perencana jalur adalah urutan titik-titik jalan diskrit yang dicoba untuk diikuti oleh robot. Setelah jalur direncanakan, robot dapat menggunakan informasi tersebut untuk memandu gerakannya dan menavigasi lingkungan.

Hal ini melibatkan penggunaan algoritma pengambilan keputusan tingkat tinggi untuk beradaptasi dengan kondisi yang berubah dan merespons kejadian tak terduga seperti baterai lemah atau informasi baru tentang lingkungan. Untuk merencanakan jalur, digunakan perencana yang diperluas. Perencana ini menggunakan peta kisi sel yang dibuat oleh peta, peta local *costmap*, dan global *costmap*. Kemudian pada setiap langkah, perencana ini menghitung cost yang dibutuhkan untuk memperluas jalur, lihat Gambar 5. Secara khusus, A\* memilih jalur yang meminimalkan,

$$f(n) = g(n) + h(n) \quad (1)$$

di mana, n adalah simpul berikutnya dalam jalur,  $g(n)$  adalah *costmap* jalur dari simpul awal ke n, dan  $h(n)$  adalah fungsi heuristik yang mengestimasi *costmap* jalur termurah dari n ke tujuan. A \* dijamin untuk mengembalikan jalur dengan *costmap* paling rendah dari awal ke tujuan.

### 2.6.1 Navigation Stack

*Navigation Stack* cukup sederhana pada tingkat konseptual meskipun *navigation stack* dirancang untuk tujuan umum, ada tiga persyaratan perangkat keras utama yang membatasi penggunaannya:

- a. Ini dimaksudkan untuk penggerak diferensial dan robot beroda holonomik saja. Ini mengasumsikan bahwa pangkalan bergerak dikendalikan dengan mengirimkan perintah kecepatan yang diinginkan untuk dicapai dalam bentuk: kecepatan x, kecepatan y, kecepatan *theta*.
- b. Membutuhkan *laser planner* yang dipasang di suatu tempat di pangkalan bergerak. Laser ini digunakan untuk pembuatan peta dan pelokalan.
- c. *Navigation Stack* dikembangkan pada robot otonom, sehingga kinerjanya akan menjadi lebih optimal pada robot yang bergerak secara kompleks. Ini bekerja pada robot dengan bentuk dan ukuran yang berubah-ubah, tetapi mungkin mengalami kesulitan dengan robot persegi panjang besar di ruang sempit seperti pintu.

Istilah *tf* adalah paket yang memungkinkan pengguna melacak beberapa *frame* koordinat dari waktu ke waktu. Istilah *tf* mempertahankan hubungan antara *frame* koordinat dalam struktur pohon yang disangga dalam waktu dan memungkinkan pengguna mentransformasikan titik, vektor, dll. antara dua frame koordinat pada titik waktu yang diinginkan. Sistem robotik biasanya memiliki banyak bingkai koordinat 3D yang berubah seiring waktu, seperti *frame* lingkungan, *frame* dasar dan sebagainya. *tf* melacak semua area lingkungan ini dari waktu ke waktu dan memungkinkan pengembang untuk mengajukan pertanyaan seperti:

- a. Di mana posisi robot terhadap lingkungannya, 5 detik yang lalu?
- b. Bagaimana pose objek dalam *gripper* relatif terhadap posisi saya?
- c. Bagaimana pose saat ini dari bingkai dasar dalam bingkai peta?

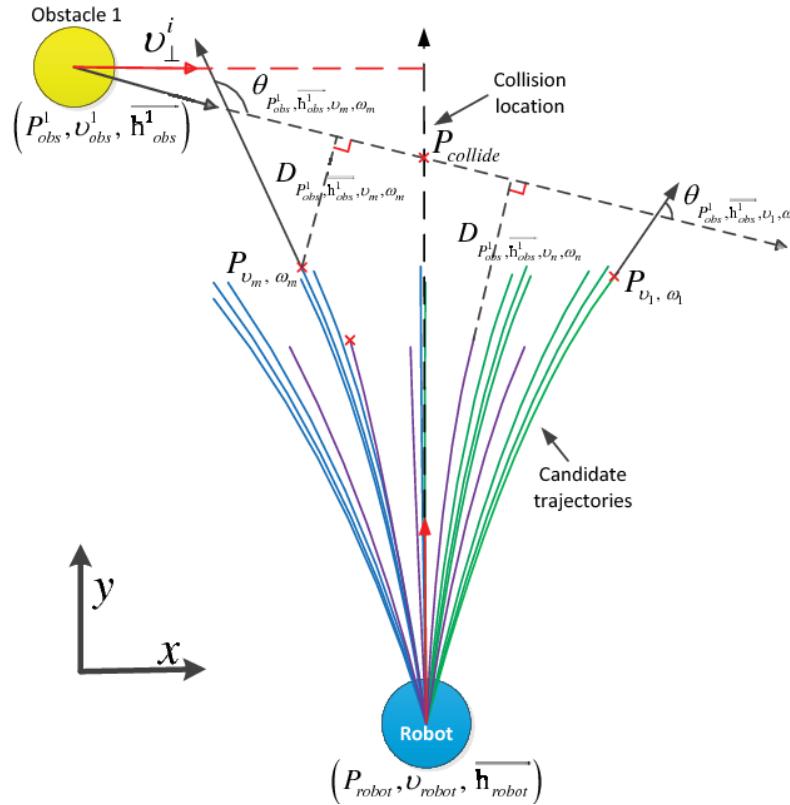
*tf* dapat beroperasi dalam sistem terdistribusi. Ini berarti semua informasi tentang frame koordinat robot tersedia untuk semua komponen ROS di komputer manapun dalam sistem ([ros.org](http://ros.org), 2018).

### 2.6.2 Local Planner

*Local Planner* merupakan perencana jalur untuk mengimbangi resolusi peta global yang rendah, perencana lokal dikonfigurasikan untuk menggunakan peta dengan resolusi yang lebih tinggi, yaitu 5 cm per piksel. Karena perencana lokal hanya perlu memproses data peta di area sekitar robot, maka data yang harus diproses secara signifikan lebih sedikit dan kerapatan piksel yang lebih tinggi hanya

memiliki sedikit efek pada kinerja. Dalam melakukan perencanaan jalur terpendek dalam cakupan lokal terdapat beberapa metode yang telah ada di *library ROS* misalnya DWA (*Dynamic Window Approach*) yang membuat lintasan berdasarkan peta yang telah dibuat sebelumnya (Dieter, *et.al.* 1997).

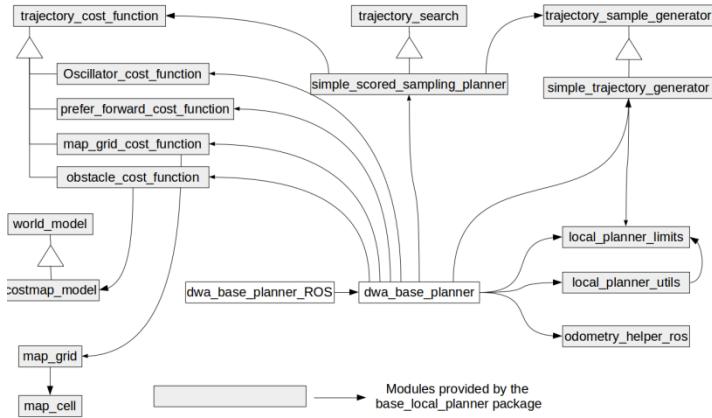
Algoritma DWA akan membuat beberapa macam lintasan dan membuat suatu fungsi nilai untuk menghitung jarak terdekat serta menghindari adanya penghalang baik itu penghalang yang statis maupun yang dinamis. Nilai dari jarak yang dibuat berdasarkan grid yang dibentuk oleh DWA. Fungsi nilai ini berupa  $dx$ ,  $dy$  dan  $d\theta$  yang akan secara simultan dikirimkan ke robot. Ide dasar dari algoritma DWA dapat dilihat pada Gambar 6.



Gambar 6 Visualisasi ide dasar algoritma DWA

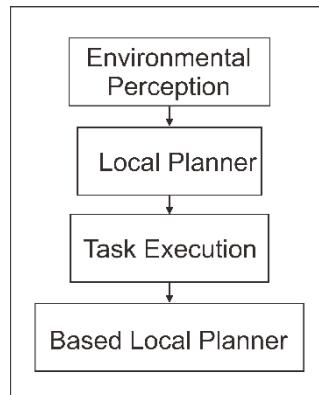
Selain menggunakan DWA sebagai penghitung nilai yang akan dilalui, pada base *local planner* juga melakukan simulasi untuk memprediksi serta mencari nilai-nilai terbaik dalam melakukan pencarian jarak terdekat. Terdapat pula beberapa karakteristik yang perlu diperhatikan seperti kedekatan dengan rintangan, kedekatan dengan tujuan kecepatan dan karakteristik lainnya. Setelah dilakukan

simulasi maka nilai terbaik yang nantinya akan dipilih oleh robot dalam menentukan lintasannya (Fadil dan Cakra, 2021).



Gambar 7 Sistem navigasi ROS menggunakan algoritma DWA

Kutub *encoder* mengukur jarak yang ditempuh oleh robot. Dengan membedakannya, kecepatan roda saat ini dapat diperkirakan. Kemudian, perbedaan antara kecepatan yang diinginkan dan kecepatan saat ini dihitung. Perbedaan ini diberikan pada input pengontrol P sederhana yang bertugas mengubah laju kecepatan yang diinginkan menjadi akselerasi yang diinginkan. Akselerasi ini kemudian dibatasi dan dimasukkan ke dalam pengontrol PID. Pada keluarannya adalah daya motor yang diperlukan untuk mencapai kecepatan tujuan kita. Kontrol PID tambahan ini meningkatkan optimasi pergerakan robot.

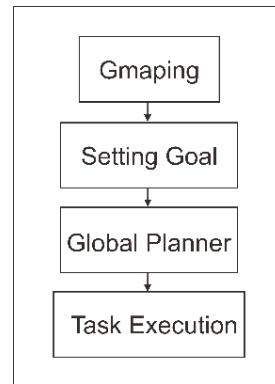


Gambar 8 Hirarki sistem *local planning*

### 2.6.3 Global Planner

Dalam menghasilkan jalur yang layak untuk bergerak dari posisi saat ini ke tujuan yang diinginkan, kendala kinematik robot harus diperhitungkan. Sebagian besar perencana global saat ini menggunakan representasi diskrit dari keadaan robot yang mengurangi kompleksitas komputasi dengan mengabaikan kelengkapan.

Namun, representasi diskrit ini menyulitkan untuk memenuhi batasan kemampuan manuver kebanyakan robot.



Gambar 9 Hirarki sistem *global planning*

Keadaan robot diberikan kode dengan posisi ( $x,y$ ) dan *heading* ( $\theta$ ) untuk memastikan lintasan yang dihasilkan mulus (tanpa perubahan mendadak pada *heading* robot). Namun, kecepatan dan akselerasi dari parameter ini tidak dipertimbangkan. Beberapa algoritma telah dikembangkan untuk mengatasi masalah ini.

#### 2.6.4 Local Costmap

Peta *local costmap* digunakan bersama dengan peta *costmap* global dan merepresentasikan pandangan yang lebih lokal dari lingkungan robot. Peta *costmap* lokal diimplementasikan sebagai *plugin* di *Navigation Stack* dan digunakan oleh beberapa *node* lain yang dibahas di sini. Secara bersamaan, mereka menciptakan *potential field* di sekitar semua rintangan di peta dan terlihat oleh kinect. Hal tersebut sudah cukup karena kecepatannya motor yang rendah, tetapi jika robot diharapkan melaju lebih cepat atau rintangan mendekat dengan cepat, *costmap* yang lebih besar bisa diperlukan. Mirip dengan peta *costmap* global, ukuran bidang potensial di sekitar rintangan dapat ditentukan untuk menjaga jarak yang aman. Masalah yang kerap terjadi adalah secara optimal robot harus dapat membedakan antara rintangan untuk menjaga jarak yang lebih jauh dari rintangan yang bergerak seperti orang yang berjalan di area robot.

#### 2.6.5 Global Costmap

Peta *costmap* global adalah peta grid yang mewakili pandangan global lingkungan robot dan dibuat sebagai lapisan di atas peta. Sebuah pengaturan digunakan di mana sel dalam *costmap* ditafsirkan hanya sebagai tiga nilai, bebas,

ditempati, dan tidak diketahui. Ksaran nilai adalah nilai piksel yang umum (0-255) di mana 255 adalah penghalang, dan 0 adalah bebas. Jika digunakan dalam mode dengan lebih dari tiga nilai, tempat di mana robot harus mencoba menghindari atau memperlambat, dapat ditambahkan ke peta dengan memiliki nilai tengah. Kemudian, menghitung *costmap* untuk melintasi tempat itu dan menggunakannya dalam perencanaan jalur dan mengikuti perilaku dapat disesuaikan untuk bernegosiasi di sekitar area tersebut. Agar tidak menabrak objek dan dinding, batasan kinematik dan bentuk robot dipertimbangkan dalam peta *costmap*.

## 2.7 Motion Control

Kontrol gerak adalah fase terakhir dalam loop kontrol robot, di mana rencana tingkat tinggi yang dihasilkan pada fase sebelumnya diterjemahkan ke dalam gerakan robot. Oleh karena itu, level ini memproses perintah gerak abstrak dan menghasilkan perintah tingkat rendah untuk mengendalikan kecepatan motor.

Penghindaran rintangan merupakan hal yang menarik, dan dapat diklasifikasikan di bawah kontrol gerak. Ini adalah salah satu masalah utama untuk aplikasi robot bergerak yang sukses, karena memastikan keamanan robot dan entitas di sekitarnya. Strategi penghindaran rintangan berkisar dari algoritme primitif yang hanya menghentikan robot saat rintangan terdeteksi; hingga algoritme kompleks yang memungkinkan robot menghindari rintangan.

Borenstein memperkenalkan algoritma *vector field histogram* (vfh) untuk tugas penghindaran rintangan berdasarkan medan potensial lokal. Dalam pendekatan ini, pertama-tama, data jangkauan diambil sampelnya secara terus menerus, dan kisi-kisi lokal dua dimensi dibuat untuk merepresentasikan lingkungan. Pada tahap berikutnya, histogram polar satu dimensi diekstraksi dari grid lokal dalam bentuk sektor-sektor sudut dengan lebar tertentu. Terakhir, histogram satu dimensi ini diberi ambang batas dan sektor sudut dengan kerapatan tertinggi dipilih sebagai arah. Kecepatan robot juga disesuaikan dalam korelasi dengan jarak dari rintangan. Dalam (Ulrich, et. al., 1998) algoritma ini ditingkatkan dengan memasukkan kinematika robot karena algoritma asli mengasumsikan bahwa robot dapat mengubah arahnya secara instan (dinamai vfh+). Algoritma ini kemudian diperbaiki dan dinamakan sebagai vfh\* dalam (Ulrich, et. al., 2000). Berbeda dengan vfh dan vfh+, yang merupakan algoritme lokal murni berdasarkan

pembacaan sensor saat ini, vfh\* memasukkan algoritme pencarian grafik A\* untuk mempertimbangkan lebih dari sekadar lingkungan sekitar robot.

Dalam (Fox *et. al.*, 1997) pendekatan jendela dinamis diperkenalkan sebagai metode penghindaran rintangan. Kendala kinematik dari robot penggerak *Synchro* diperhitungkan dengan mencari ruang kecepatan robot secara langsung. Ruang pencarian selanjutnya direduksi menjadi jendela dinamis, yang berisi kecepatan yang dapat dicapai oleh robot, dengan kecepatan dan akselerasinya. Akhirnya, jendela ini dicari kecepatan yang sesuai dengan arah target robot. Dalam (Brock *et. al.*, 1999) metode ini diadaptasi ke robot *holonomic*, yang memungkinkan penghindaran rintangan berkecepatan tinggi dengan kemampuan manuver yang tinggi.

Terakhir, diagram kedekatan diperkenalkan dalam (Minguez *et. al.*, 2000) yang didasarkan pada aturan heuristik yang disimpulkan dari kemungkinan situasi keamanan tinggi dan rendah yang dapat diakhiri oleh robot. Berdasarkan lima aturan (dua situasi keamanan rendah dan tiga situasi keamanan tinggi), lima perilaku didefinisikan, di mana robot membandingkan situasinya saat ini dengan situasi yang telah ditentukan dan menjalankan perilaku yang sesuai. Hal ini ditunjukkan bahwa pendekatan reaktif ini dapat bekerja dengan baik di lingkungan yang berantakan dengan lorong-lorong yang sempit, dibandingkan dengan pendekatan sebelumnya.

## **2.8 Differential Drive Mobile Robot**

Robot dengan penggerak diferensial (*differential drive*) adalah robot yang menggunakan 2 penggerak yang dipasang pada kedua rodanya dan bergerak secara independen. Tipe robot ini dilengkapi dengan 1 atau 2 roda pasif (*caster wheels*) yang bertujuan untuk mempertahankan keseimbangan robot. Setiap roda dapat diatur untuk bergerak dengan kecepatan dan arah putar yang berbeda, sehingga memungkinkan robot untuk dapat bergerak lurus maupun melingkar. Jika robot pergerakan melingkar, robot harus berputar mengilingi suatu titik yang berada sejajar dengan sumbu kedua rodanya, yang disebut sebagai *Instantaneous Center of Curvature* (ICC). Perubahan kecepatan setiap roda akan merubah lintasan robot (Dudek, *et.al.* 2010). Kecepatan sudut robot (terhadap ICC) harus sama untuk kedua roda, sehingga dapat berlaku beberapa persamaan (2) dan (3) sebagai berikut.

$$w \left( R + \frac{1}{2} \right) = Vr \quad (2)$$

$$w \left( R + \frac{1}{2} \right) = Vt \quad (3)$$

Dimana,

$i$  : Jarak antara kedua roda

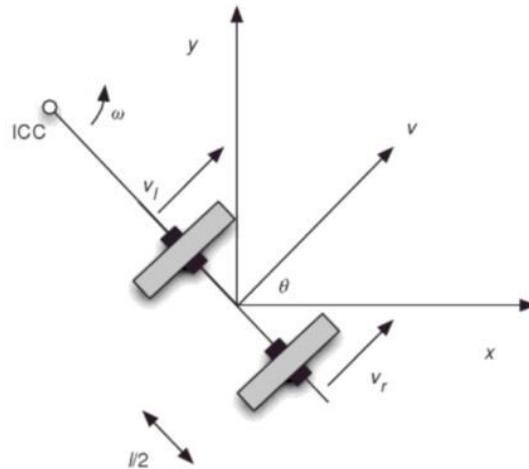
$v_r$  : Kecepatan roda kanan terhadap tanah

$v_t$  : Kecepatan roda kiri terhadap tanah

$R$  : Jarak titik tengah robot dengan titik pusat rotasi

Nilai kecepatan angular robot terhadap titik pusat rotasi ( $\omega$ ) dapat dicari dengan mengurangkan persamaan (4) dan (5) sehingga didapat persamaan (3) sebagai berikut.

$$w = \frac{Vr - Vl}{l} \quad (4)$$



Gambar 9 Sistem *differential drive robot*

Nilai jarak pusat robot dengan sumbu rotasi ( $R$ ) dapat dicari dengan menjumlahkan persamaan (1) dan (2) sehingga didapat persamaan (4) sebagai berikut.

$$R = \frac{Vr - Vl}{Vr - vl} \quad (5)$$

Persamaan (4) dan (5) menunjukkan beberapa jenis pergerakan yang dapat dilakukan oleh robot yaitu:

Jika  $v_r = vl$ , robot bergerak lurus karena nilai  $R$  tak hingga dan  $\omega = 0$  atau tidak ada perputaran.

Jika  $vl = -vr$ , robot berputar di tempat atau berputar dengan titik pusat perputaran berada pada titik pusat robot ( $R = 0$ ).

Jika  $vl = 0$  dan  $vr \neq 0$ , robot akan berputar berlawanan arah jarum jam dengan pusat rotasi berada pada roda kiri ( $R = l_2$ ) dan berlaku sebaliknya.

## 2.9 Aktuator

Mekanisme perhitungan kecepatan putar dilakukan dengan menghitung jumlah pulsa *rotary encoder* secara periodik. Perputaran motor menghasilkan pulsa *encoder* yang selanjutnya dideteksi oleh kontroler melalui mekanisme *hardware interrupt*. Hasil percobaan menunjukkan *rotary encoder* yang digunakan, seperti yang terlihat pada gambar 10, mempunyai jumlah 339 pulsa setiap putaran.



Gambar 10 Contoh komponen aktuator

Sehingga kecepatan roda dapat dihitung dengan persamaan (5) sebagai berikut.

$$rpm = \frac{(Jumlah Pulsa / 339)}{(Time sampling \times 60)} \quad (6)$$

Selanjutnya akan dilakukan pembacaan dan kalibrasi sensor kecepatan pada roda kanan dan roda kiri. Pengamatan dan pencatatan dilakukan dengan menggunakan serial monitor yang dikirimkan oleh kontroler. Kalibrasi kecepatan dilakukan dengan membandingkan hasil pembacaan sensor dengan rpm meter. Pengukuran kecepatan dilakukan pada tegangan baterai 28,8 volt (tegangan maksimum 29,4 volt) sebelum masuk ke *buck converter* dan motor berputar pada arah yang menyebabkan robot bergerak maju.

## 2.10 Kinematika dan *Trajectory Tracking*

Perancangan sistem kontrol kinematik dan *trajectory tracking* yang diimplementasikan pada robot menggunakan 2 algoritma kinematik, yaitu *inverse velocity* kinematik untuk kontrol kinematik gerak robot dan *forward velocity* kinematik yang digunakan untuk sistem *trajectory tracking* (Kim dan Yi, 2014). Pada dasarnya, persamaan kinematik dapat dilihat pada (1). Di mana  $x$  adalah vektor pose yang berisi koordinat  $x$  koordinat  $x$ , koordinat  $y$ , dan arah hadap ( $\theta$ ) dari robot pada lintasan di mana robot bergerak (Sorour, dkk. 2017).

$$x = f(q) \quad (7)$$

Persamaan (7) merupakan persamaan dalam dimensi posisi posisi, dimana pengendalian robot dengan kontrol posisi sulit untuk dilakukan (Morales, et.al., 2021). Sehingga persamaan (7) dapat diturunkan terhadap waktu untuk mendapatkan persamaan dalam dimensi kecepatan dalam dimensi kecepatan seperti yang tertulis di bawah ini.

$$\frac{dx}{dt} = \frac{df(q)}{dt} \quad (8)$$

$$\dot{x} = \frac{df(q)}{dq} \times \frac{dq}{dt} \quad (9)$$

$$\dot{x} = J \cdot \dot{q} \quad (10)$$

Dimana  $J$  adalah matriks Jacobian, dimana matriks tersebut berisi model matematis yang berkorelasi dengan spesifikasi mekanik robot dengan desain kontrol yang dibuat.  $\dot{q}$  adalah vektor yang berisi kecepatan aktuator yang dimiliki oleh robot. Dimana  $r$  adalah jari-jari roda robot,  $L$  adalah jarak antara titik tengah roda kiri dan roda kanan robot, dan sudut  $\theta$  adalah sudut dari arah robot ke lintasan.

## BAB III

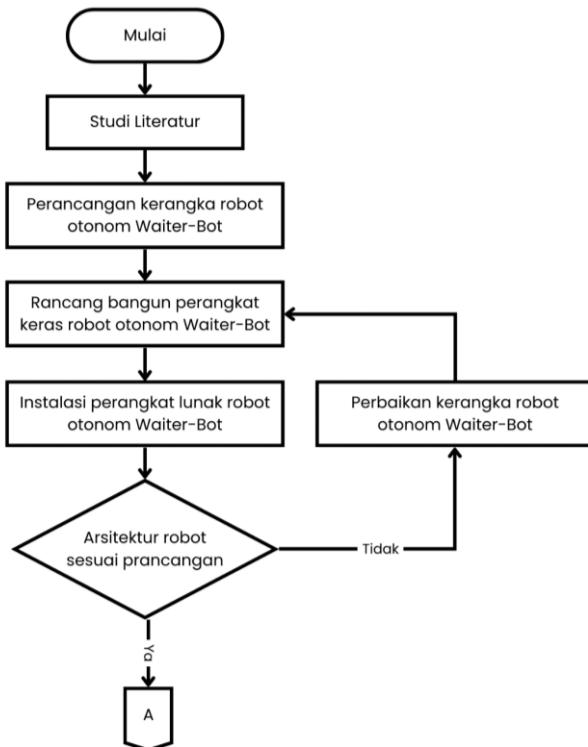
### METODE PENELITIAN

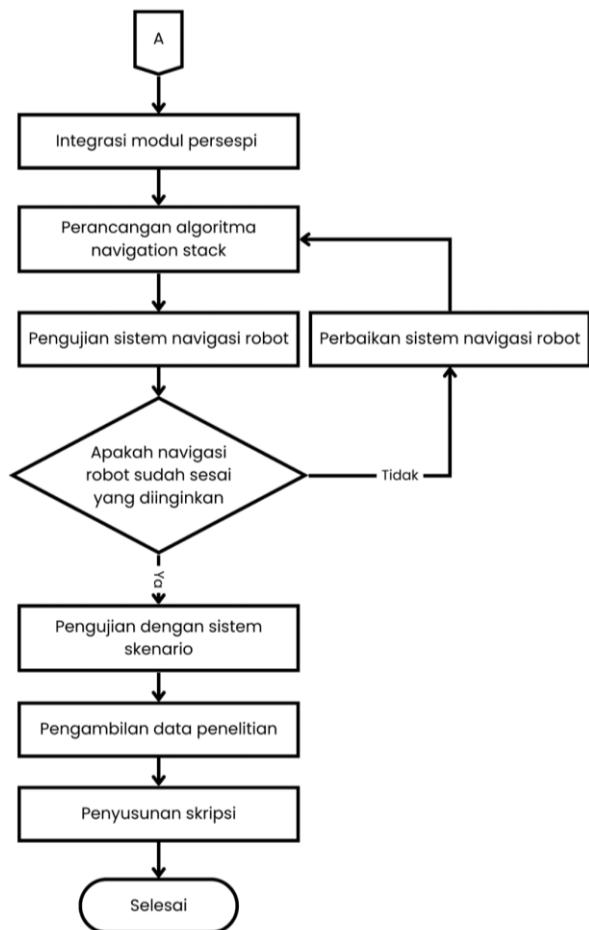
#### **3.1 Waktu dan Lokasi Penelitian**

Pelaksanaan penelitian tugas akhir ini terhitung sembilan bulan tepatnya pada bulan Januari-September 2023. Adapun lokasi pelaksanaan penelitian ini bertempat di Laboratorium Sistem Kendali dan Instrumentasi Departemen Teknik Elektro Fakultas Teknik Universitas Hasanuddin.

#### **3.2 Rancangan Umum Penelitian**

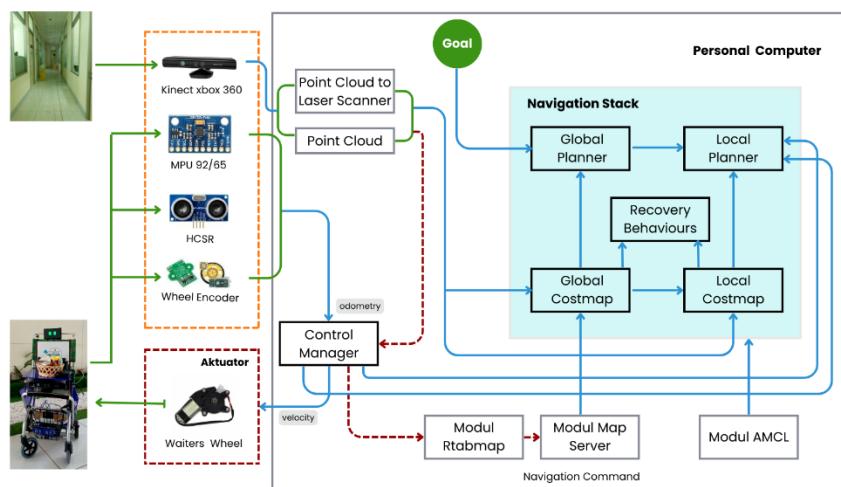
Pada tahapan perancangan umum implementasi sistem navigasi robot otonom *Waiter-Bot* menggunakan modul *framework Robot Operating System* (ROS) sebagai robot *middleware*. Secara keseluruhan alur pelaksanaan penelitian ini diawali dengan penguatan studi literatur dan konsultasi dengan dosen pembimbing. Dilanjutkan dengan melakukan tahap perancangan perangkat keras maupun perangkat lunak robot. Lalu pada tahap pengujian dilakukan di dalam ruangan (*indoor*) dengan pembagian beberapa skenario pengujian. Blok diagram perancangan penelitian dapat dilihat pada Gambar 10.





Gambar 11 Diagram alir rancangan penelitian

Tahapan perancangan sistem navigasi robot otonom *Waiter-Bot* meliputi aspek perangkat keras dan perangkat lunak di mana blok diagram secara keseluruhan dapat dilihat pada Gambar 12.



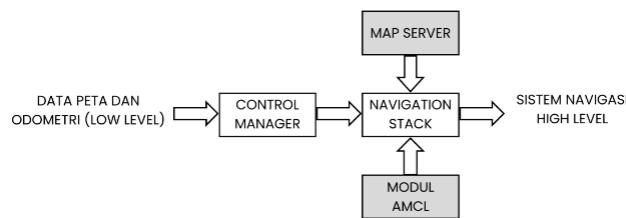
Gambar 12 Sistem navigasi *Waiter-Bot* berbasis ROS

Berdasarkan gambar tersebut, perancangan penelitian ini berorientasi pada tiga aspek besar yaitu masukan, proses dan keluaran.



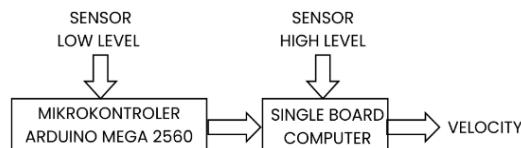
Gambar 8 Diagram blok sistem

Diagram blok secara keseluruhan menerima informasi eksisting dari tahap persepsi robot yakni hasil pemetaan dan lokalisasi kemudian diolah dalam mendukung proses navigasi robot otonom.



Gambar 9 Diagram blok masukan

Pada blok masukan meliputi data pemetaan dan lokalisasi yang akan dikontrol oleh control manager dan akan menjadi input *navigation stack*. Begitu pula dengan data dari modul *map server* dan AMCL merupakan data lokalisasi yang akan dibutuhkan oleh robot saat melakukan navigasi.



Gambar 10 Diagram blok proses

Pada tahap pemrosesan sistem dilakukan oleh mikrokontroler Arduino mega 2560 dan komputer berbasis ROS yang dijalankan dalam PC. Segala bentuk komunikasi dan proses yang terjadi pada sensor *low level* dilakukan oleh mikrokontroler. Adapun sensor untuk sistem *high level* diproses oleh PC. Sistem *low level* meliputi *voltmeter*, sensor IMU, ultrasonik HCSR, *encoder* dan motor DC. Pada sistem *high level* terdiri atas sensor kinect dan *framework* ROS. Secara keseluruhan, perancangan.

### 3.3 Rancangan Perangkat Keras

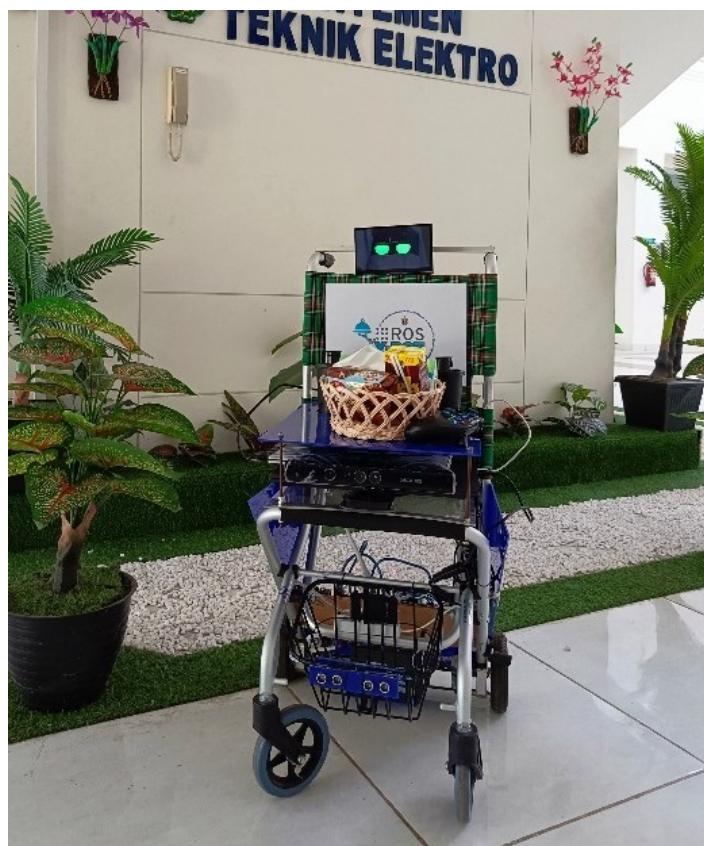
Dalam perancangan perangkat keras, terbagi atas beberapa tahapan perancangan sebagai berikut.

#### 3.3.1 Mekanisasi Robot

Pada tahap rancang bangun mekanisasi robot otonom *Waiter-Bot* terdiri atas desain produk dan perancangan elektronika.

##### a. Desain Robot

Dalam mekanisasi robot dimulai dengan merancangan chasis atau badan robot agar komponen sensor dan daya dapat diletakaan bagian *control board*. Chasis dirancang dengan memodifikasi kursi roda *traveling* menjadi robot pengantar makanan. Terdapat talangan untuk meletakkan barang yang akan diangkut oleh robot, roda sebagai komponen aktuator dan layar LCD yang dapat menampilkan ekspresi robot pada Gambar 16.



Gambar 11 Tampilan eksisting *Waiter-Bot*

Keterangan informasi terkait komponen yang digunakan dalam mekanisasi robot otonom *Waiter-Bot* disajikan dalam tabel 1.

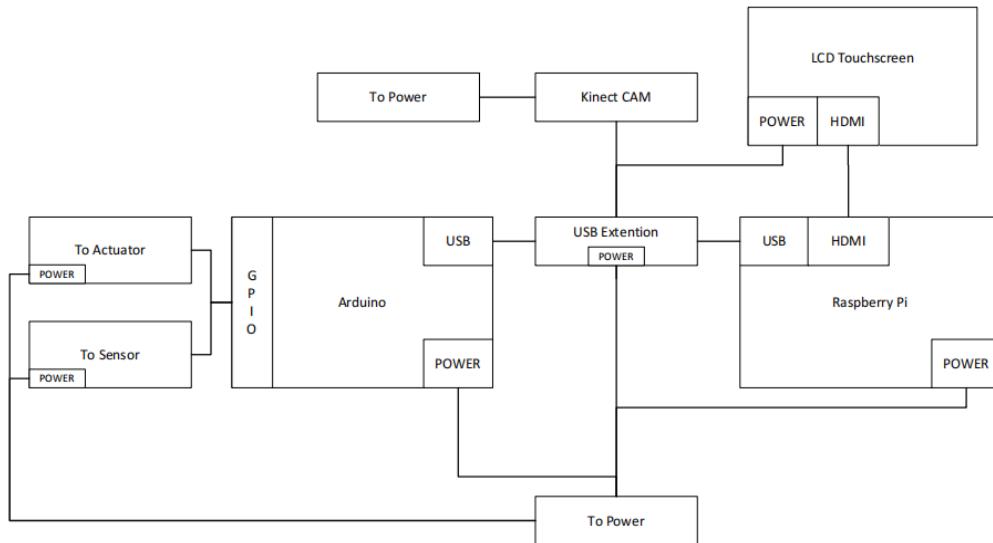
Tabel 1 Komponen utama rancang bangun robot otonom *Waiter-Bot*

Jenis	Komponen	Kegunaan
Sensor	Kinect Xbox 360	Sebagai sensor kamera dan pengganti <i>laser scanner</i>
	Encoder	Menghitung kecepatan dan arah pergerakan dari aktuator
	MPU 6050	Memberikan informasi <i>real-time</i> tentang pergerakan dan orientasi robot untuk bennavigasi
Aktuator	Ultrasonic HC-SR04	Mendeteksi jarak objek penghalang robot pada <i>low level</i>
	Motor Power Window	Sebagai actuator robot dalam melakukan navigasi
Daya	Aki Panasonic 12V	Catu daya dalam menyuplai tegangan pada sistem elektronika robot
	Regulator 12 V	Mengatur dan menyesuaikan tegangan inputan 12 V
Prosesor	Motor Driver DC HB-25	Mengatur kecepatan dan arah pergerakan dari aktuator
	Arduino Mega 2560	Pengolah data yang diperoleh dari sensor bagian <i>low level</i>
	Arduino Uno	Khusus mengolah data dari <i>motor driver</i>
PC		Melakukan komputasi ROS dan mengontrol sistem bagian <i>high level</i>

### b. Desain Elektronika

Desain kelistrikan rancang bangun robot otonom *Waiter-Bot* melibatkan penentuan kebutuhan daya robot sehingga pemilihan komponen yang sesuai dan dapat memenuhi persyaratan perlu diperhatikan. Hal ini termasuk menghitung daya

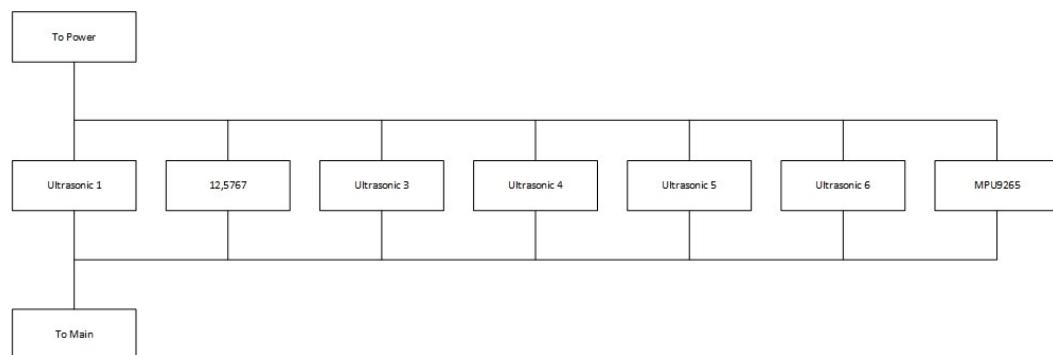
yang dibutuhkan untuk menjalankan semua sensor, aktuator, atau komponen elektronika robot. Sumber tegangan berasal dari dua aki masing-masing 12V. Oleh karena itu, tegangan ada yang diubah menjadi 5V untuk mengoperasikan mikrokontroler Arduino dan komponen yang terhubung dengannya, Gambar 17.



Gambar 12 Blok umum diagram komponen elektronika

#### d. Perancangan Susbsistem Masukan

Sistem masukan pada robot otonom *Waiter-Bot* yakni visualisasi odometri robot yang digunakan dalam tahap pemetaan dan lokalisasi. Dapat dilihat blok diagram subsistem masukan rancang bangun robot otonom *Waiter-Bot* pada Gambar 18.



Gambar 13 Blok diagram subsistem komponen masukan

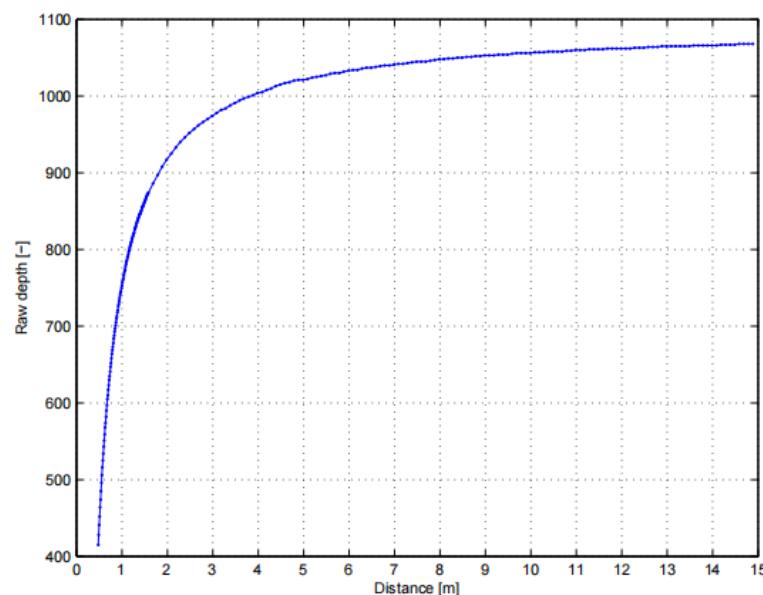
### e. Sensor Kinect Xbox 360

Sensor Kinect merupakan perangkat yang mampu mendeteksi gerak objek dan memanipulasi gambar berdasarkan perintah yang diberikan, sehingga sensor Kinect juga disebut sebagai perangkat *motion-sensing*. Kinect adalah perangkat komposit yang terdiri dari proyektor IR dengan pola dan kamera IR, yang digunakan untuk melakukan triangulasi titik-titik di ruang angkasa. Kamera ini berfungsi sebagai *deep camera*, dan kamera warna (RGB), yang dapat digunakan untuk mengenali konten gambar dan tekstur titik 3D, Gambar 19.



Gambar 14 Kinect memberikan tiga output: IR, RGB, *projector*

Kamera IR ( $1280 \times 1024$  piksel untuk FOV  $57 \times 45$  derajat, panjang fokus 6,1 mm, ukuran piksel  $5,2 \mu\text{m}$ ) digunakan untuk mengamati dan menerjemahkan pola proyeksi IR untuk melakukan triangulasi pemandangan 3D.

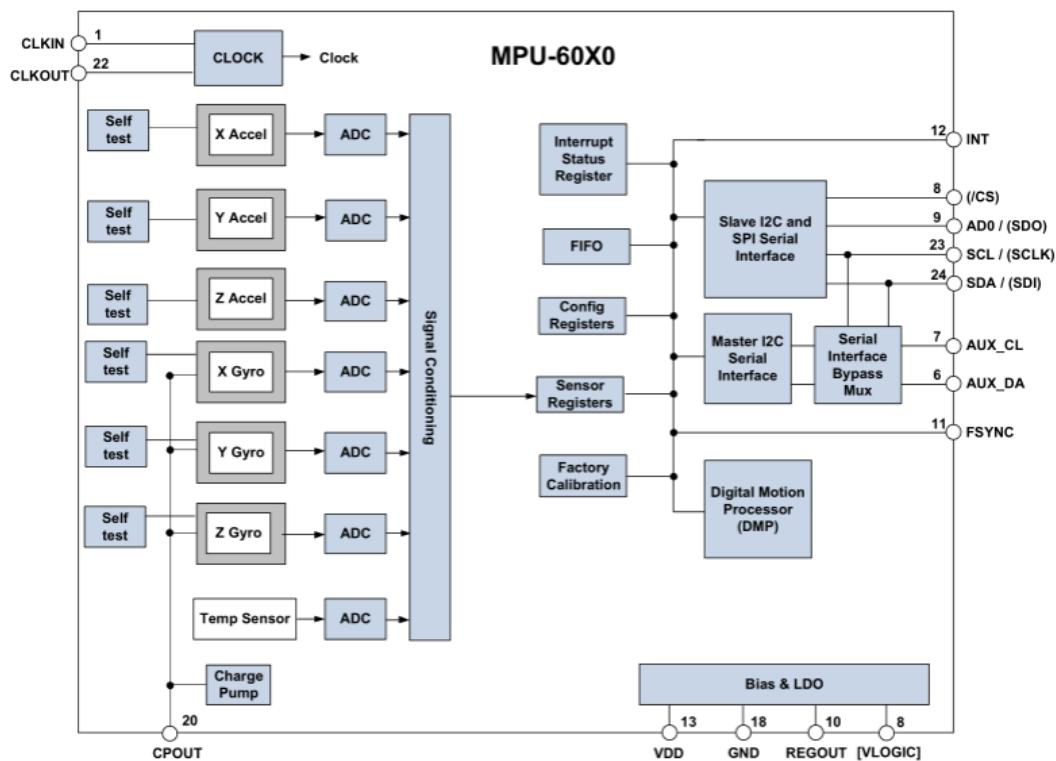


Gambar 15 Invers Kinect pada fungsi kedalaman asli

Dalam penelitian oleh Jan Sisek, *et.al* (2013) menunjukkan resolusi kedalaman sebagai fungsi jarak pada sensor kinect. Resolusi kedalaman diukur dengan menggerakkan Kinect menjauh (0,5 m-15 m) dari target planar yang cukup halus untuk merekam semua nilai yang dikembalikan dalam bidang tampilan sekitar 5° di sekitar pusat gambar.

#### f. Sensor IMU

Pada perancangan sistem sensor *Inertia Measuremen Unit* (IMU) ini dibagi menjadi dua bagian utama, yaitu sistem sensor IMU sebagai kompas dengan sensor CMPS11 dan sistem sensor IMU sebagai *feedback* posisi derajat kemiringan yaw robot dengan sensor MPU6050. Pada pengkabelan modul sensor MPU6050 dan Arduino ini dilakukan dengan menggunakan komunikasi I2C, yaitu dengan menghubungkan pin SDA dan SCL pada modul dengan pin A4 (SDA) dan A5 (SCL) pada Arduino, dan pin INT pada modul dengan pin 2 pada Arduino. Untuk tegangan supply, modul ini bekerja pada tegangan 2.375V ~ 3.46V.



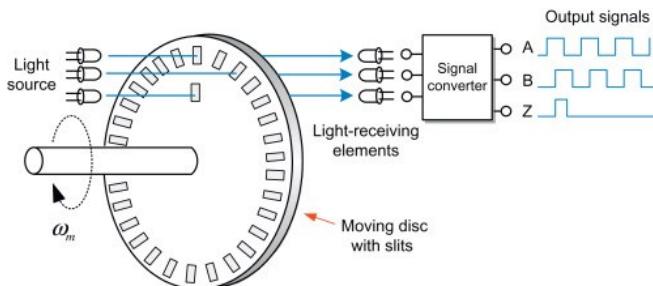
Gambar 16 Blok diagram modul sensor IMU MPU-60X0

Sensor IMU giroskop pada modul MPU-6050 ini terdiri dari tiga giroskop MEMS vibrasi, yang mendekripsi rotasi pada sumbu x, y, dan z. Ketika gyroskop

diputar pada salah satu sumbu, Efek Coriolis menyebabkan getaran yang dideteksi oleh pickoff kapasitif yang menghasilkan sinyal tegangan. Sinyal yang ini dihasilkan diperkuat, didemodulasi, dan difilter untuk menghasilkan tegangan yang sebanding dengan tingkat sudut. Tegangan ini didigitalkan menggunakan *on-chip* 16-bit *Analog-to-Digital Converters* (ADC) untuk sampel setiap sumbu. Rentang skala penuh sensor *gyro* dapat diprogram secara digital hingga  $\pm 250, \pm 500, \pm 1000$ , atau  $\pm 2000$  derajat per detik (dps) (Bagoes Prawira, 2018).

#### **g. Sensor Encoder**

*Rotary Encoder* merupakan suatu komponen elektro mekanis yang memiliki fungsi untuk memonitoring posisi anguler pada suatu poros yang berputar. Dari perputaran benda tersebut data yang termonitoring diubah oleh *y ke dalam bentuk data digital berupa lebar pulsa kemudian dikirim ke kontroler Arduino. Berdasarkan data yang di dapat berupa posisi anguler (sudut) kemudian dapat diolah oleh kontroler sehingga mendapatkan data berupa kecepatan, arah, dan posisi dari perputaran porosnya.*



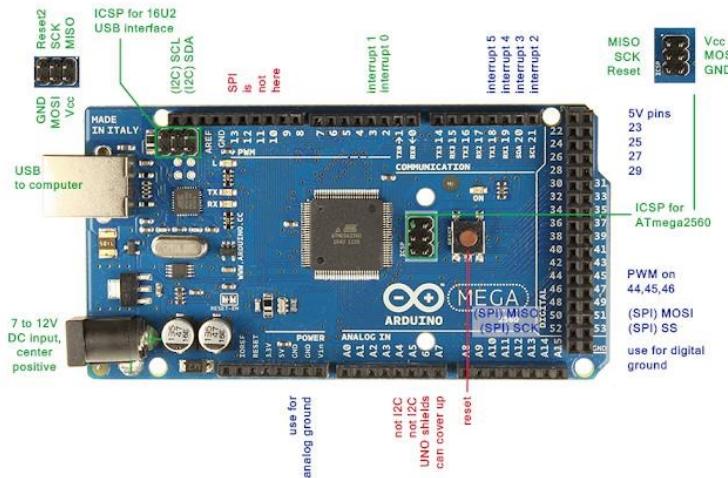
Gambar 17 Sistem kerja *encoder*

#### **3.3.3 Perancangan Subsistem Proses**

Pemrosesan dilakukan oleh mikrokontroler yang melakukan pembacaan masukan (sensor), mengontrol keluaran (pergerakan motor) dan mencari jalur di lingkungan labirin. Beberapa hal yang harus diperhatikan dalam pemilihan mikrokontroler yaitu jumlah memori dan port I/O yang dimiliki. Penelitian ini direncanakan menggunakan mikrokontroler Arduino Mega 2560 sebagai prosesor untuk sistem sensor level bawah (*low level*) serta PC *Intel core i9 Victus* sebagai prosesor untuk sistem (*high level*).

### a. Arduino Mega 2560

Arduino Mega 2560 merupakan papan mikrokontroller yang berbasis ATmega 2560 dimana memiliki 54 pin digital input / *output* (15 pin diantaranya adalah PWM), 16 pin analog input, 4 pin UART (serial port hardware). Arduino Mega 2560 juga dilengkapi oscillator 16 Mhz, sebuah port USB, power jack DC, ICSP header, dan tombol reset. Itu semua dibutuhkan untuk mendukung mikrikontroler, untuk mulai mengaktifkan cukup dengan menghubungkan power dari USB ke computer atau dengan adaptor AC – DC ke jack DC. Arduino Mega 2560 juga kompatibel dengan sebagian besar *shield* yang dirancang untuk *Arduino Deumilanove* atau *Diecimila* (Utami, 2015). Arduino Mega 2560 ditunjukkan pada gambar 23 dibawah ini.



Gambar 18 Board Arduino Mega 2560

Spesifikasi dari Arduino Mega 2560 dapat dilihat di tabel 3 dibawah ini.

Tabel 2 Spesifikasi Arduino Mega 2560

Mikrokontroler	ATmega2560
Tegangan Operasi	5V
Tegangan Input (disarankan)	7V-12V
Tegangan Input (limit)	6V-20V
Digital pin I/O	54 buah (15 diantaranya menyediakan PWM <i>output</i> )
Analog pin I/O	16 buah
Arus DC per pin I/O	20 mA

Arus DC pin 3.3V	50 mA
Memori <i>Flash</i>	256 KB, 8 KB digunakan untuk bootloader
SRAM	8 KB
EEPROM	4 KB
Waktu Kecepatan	16 Mhz
LED_BUILTIN	13
Panjang	101.52 mm
Lebar	53.3 mm
Berat	37 g

Arduino Mega 2560 mempunyai 16 pin analog input, masing-masing pin analog input menyediakan resolusi 10 bit (memiliki 1024 nilai yang berbeda). Secara default pin-pin ini diukur dari Ground sampai dengan 5 Volt, namun dapat mengubah titik jangkau menggunakan pin AREF dan fungsi *Analog Reference* (Akhmadi dkk, 2014).

#### b. Prosesor *Intel Core i9*

Prosesor adalah komponen komputer yang berfungsi sebagai pemroses dan pengontrol kinerja dalam sebuah sistem yang berfungsi sebagai pemroses dan pengontrol kinerja dalam sebuah sistem komputer. Intel *Core i9* merupakan jenis prosesor baru yang memiliki kemampuan jauh diatas Intel *Core i7*. Prosesor ini diberikan peningkatan yang sangat pesat mulai dari katan yang sangat pesat mulai dari jumlah inti dan transistor. Untuk yang terbaru, Spesifikasi *Intel Core i9* tidak hanya dilengkapi dengan 18 *core* saja melainkan dengan fitur *Hyper Threading*.



Gambar 19 Intel Core i9 Gen 12

Secara teoritis, semakin banyak core yang dimiliki suatu prosesor, maka semakin banyak tugas yang dapat dilakukan oleh laptop atau PC maka semakin banyak tugas yang dapat dilakukan oleh laptop atau PC desktop secara bersamaan. Dengan jumlah tersebut, jelas bahwa prosesor *Intel Core i9* secara bersamaan.

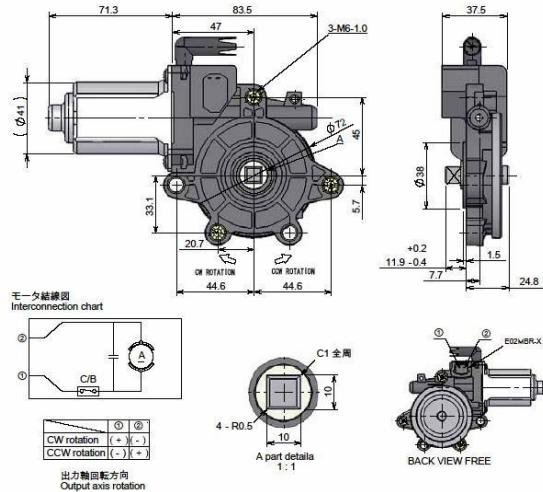
Tabel 3 Spesifikasi Intel Core i9 Generasi 12

Koleksi Produk	Prosesor Intel® Core™ i9 Generasi ke-11
Nomor Prosesor	i9-11900H
Frekuensi Turbo Maks	4.90 GHz
Cache	24 MB Intel® Smart Cache
Kecepatan Bus	8 GT/s
Frekuensi Konfigurasi TDP Lebih Tinggi	2.50 GHz
Konfigurasi TDP Lebih Tinggi	45 W
Frekuensi Konfigurasi TDP Lebih Rendah	2.10 GHz
Konfigurasi Tdp Lebih Rendah	2.10 GHz
Ukuran Memori Maks	128 GB
Jenis Memori	Up to 3200 MT/s
Bandwith Memori Maks	51.2 GB/s
Grafis Porsesor	Grafis Intel® UHD untuk Prosesor Intel® Core™ Generasi ke-11
Frekuensi Dasar Grafik	350 MHz
Frekuensi Dinamis Maks Grafis	1.45 GHz
<i>Output</i> Grafis	eDP 1.4b   DP 1.4   HDMI 2.0b

Sumber: Intel, 2021

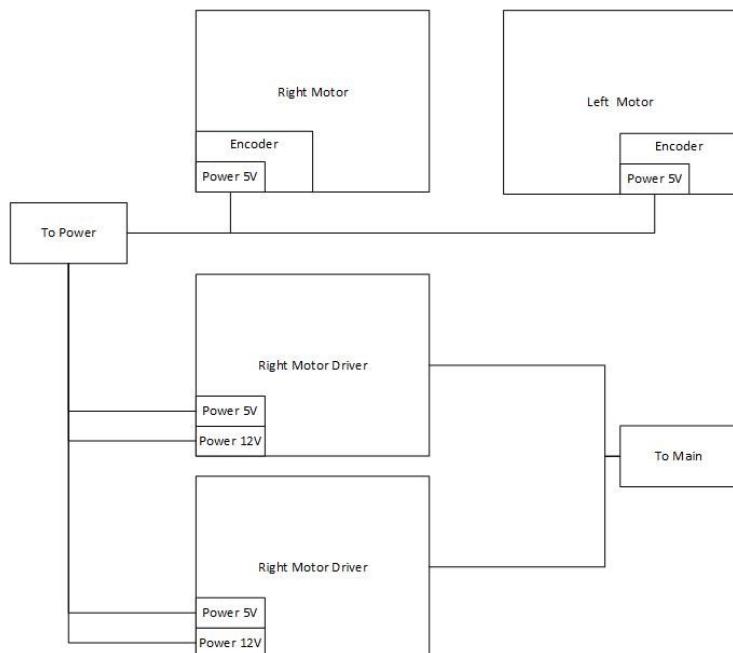
### c. Perancangan Subsistem Keluaran

Keluaran dari sistem merupakan pergerakan roda robot secara otonom dengan menggunakan komponen motor DC *Power Window*. Motor DC *Power Window* adalah suatu motor yang mengubah energi listrik searah menjadi mekanis yang berupa tenaga penggerak torsi (Christanto dkk., 2014).



Gambar 20 Motor DC Power Window 12 V

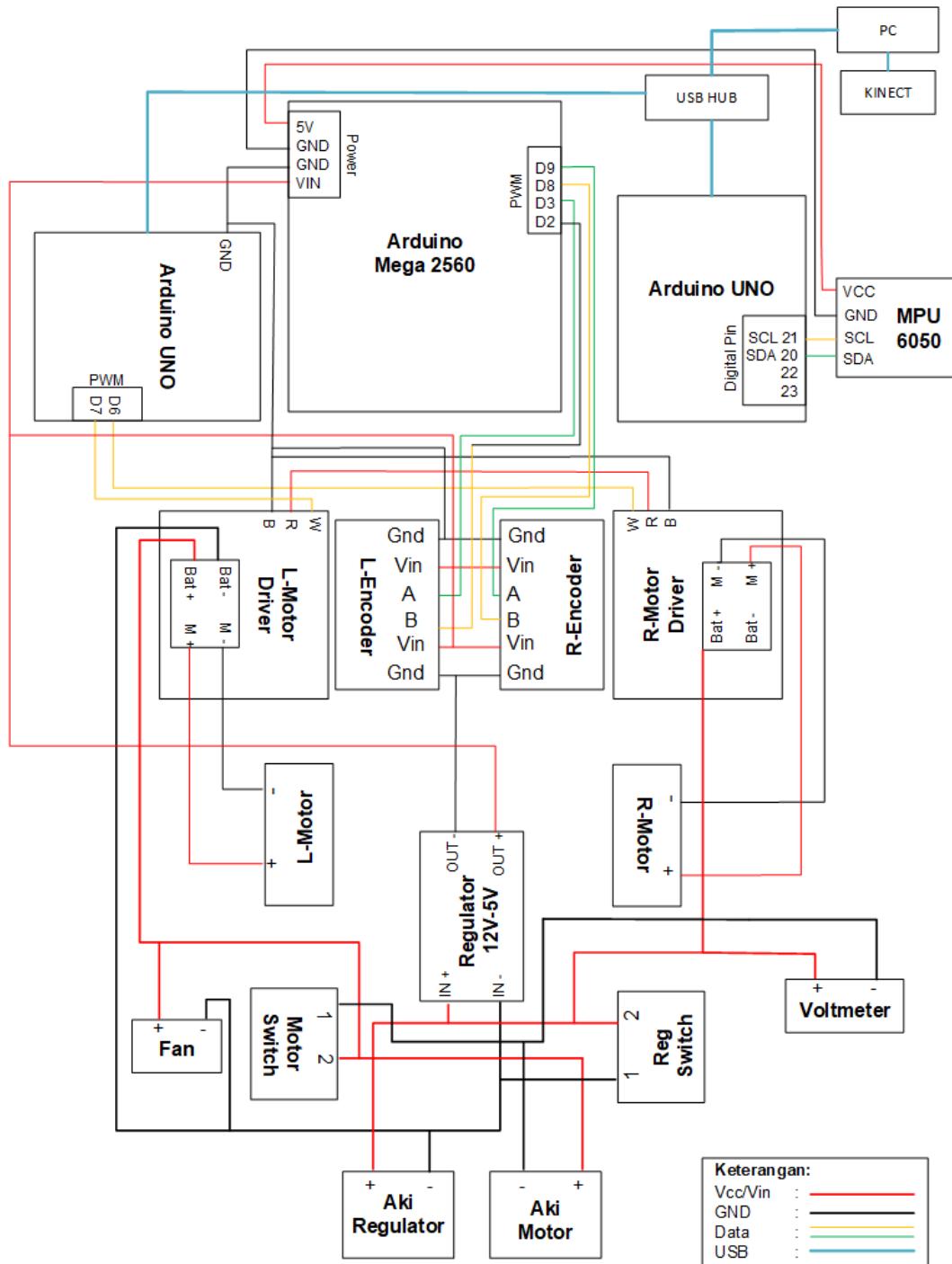
Motor *power window* dipilih sebagai motor penggerak karena torsi tinggi dengan rating tegangan input yang rendah yaitu 12 VDC, dan dimensi motor yang relatif simpel dilengkapi dengan internal *gearbox* sehingga memudahkan untuk instalasi mekanik. Berikut rangkaian skematis motor DC *power window* dengan *driver* motor dan Arduino Mega 2560.



Gambar 21 Blok diagram aktuator robot

Dalam mengendalikan motor DC *Power Window* dibutuhkan sebuah komponen *driver motor*. Dalam penelitian ini digunakan motor driver HB-25 dalam mengendalikan kecepatan putaran motor. Untuk menghubungkan motor DC dan

driver motor ke Arduino Mega 2560, terlebih dahulu perlu menghubungkan kabel dan komponen seperti Gambar 27. Ketika Arduino sebagai prosesor bagian *low level* menerima perintah, maka *driver motor* akan mengubah daya yang diberikan ke motor DC sehingga motor berputar sesuai dengan instruksi.



Gambar 22 Instalasi keseluruhan perangkat keras robot

Berikut komponen elektronika yang digunakan dalam prosesor *low level*:

Tabel 4 Komponen elektronika kendalian *low level*

Komponen	Tegangan (V)	Arus (A)	Daya (P)	Jumlah
<i>Quadrature Encoder</i>	5	5	25	2
MPU 6050	5	3.9	19.5	1
<i>Ultrasonic HC-SR04</i>	5	0.015	0.075	6
<i>Driver DC HB-25</i>	6	0.5	3	2

Arduino Mega 2560 memiliki 8 KB SRAM (*Static Random Access Memory*) yang dapat digunakan untuk menyimpan variabel dan data sementara selama program berjalan. Dalam perhitungan penggunaan memori dalam menjalankan program dari sistem *low level* pada Arduino Mega 2560 sekitar 6 KB kebutuhan penyimpanan Arduino.

### 3.4 Rancangan Perangkat Lunak

Perangkat lunak dibuat dibagi menjadi beberapa bagian sesuai dengan tahapan proses dari awal akuisisi data dari *low level* kemudian diproses pada sistem *high level* untuk memperoleh sistem navigasi robot otonom. Proses secara keseluruhan dari sistem perangkat lunak yang harus dibuat secara umum adalah:

1. Integrasi sensor *low level* pada komputasi Arduino Mega 2560
2. Sistem komunikasi komponen *low level* dengan sistem *high level*
3. Proses pemetaan dan lokalisasi
4. Pembuatan topik dan *node* pada ROS
5. Perencanaan jalur navigasi robot otonom *Navigation Stack*

#### 3.4.1 Instalasi ROS

Untuk cara penginstalan *ROS Noetic* cukup mengikuti langkah - langkah yang sudah diberikan pada website ROS. Untuk langkah – langkahnya sebagai berikut.

1. Open terminal pada Ubuntu, kemudian ikuti langkah – langkah yang terdapat pada website ROS.
2. *Setup Source.list*

Mengizinkan komputer untuk menerima software dari *package.ros.org*.

```
sudo sh -c 'echo "deb http://packages.ROS.org/ROS/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ ROS-latest.list'
```

3. *Setup keys*

```
sudo apt install curl # if you haven't already install ed
curl curl -s https://raw.githubusercontent.com/ROS/ROSDistro/master/ROS.asc | sudo apt-key add -
```

4. Instalasi

Sebelum melakukan penginstalan pastikan Ubuntu telah diperbarui ke pembaruan terbaru

```
sudo apt update
```

5. Setelah Ubuntu ter-*update*, selanjutnya bisa dilakukan penginstalan *ROS Noetic*. Untuk penginstalan memiliki 3 opsi

- a. Opsi Pertama ini sangat disarankan karena seluruh package 2D/3D seperti Gazebo dan RViz sudah bisa langsung digunakan

```
sudo apt install ROS-noetic-desktop-full
```

- b. Opsi Kedua hanya melakukan penginstalan ROS-base dan sudah terdapat tools grafis seperti rqt dan RViz.

```
sudo apt install ROS-noetic-desktop
```

- c. Opsi Ketiga hanya menginstal dasar ROS Noetic saja. Jadi perlu melakukan penginstallan RViz jika ingin menggunakannya.

```
sudo apt install ROS-noetic-ROS-base
```

6. Untuk mengakses ROS yang telah terinstall harus mengkonfigurasikan terlebih dahulu pada saat membuka terminal.

```
source /opt/ROS/noetic/setup.bash
```

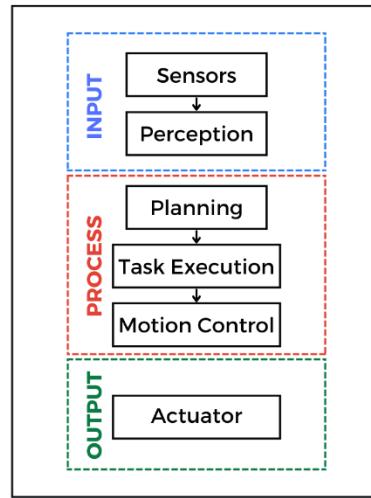
Dapat juga mengakses ROS tanpa harus mengkonfigurasi terlebih dahulu menggunakan perintah

```
Echo "source /opt/ROS/noetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

7. *ROS Noetic* pun selesai diinstal

### **3.4.2 Autonomous Movement**

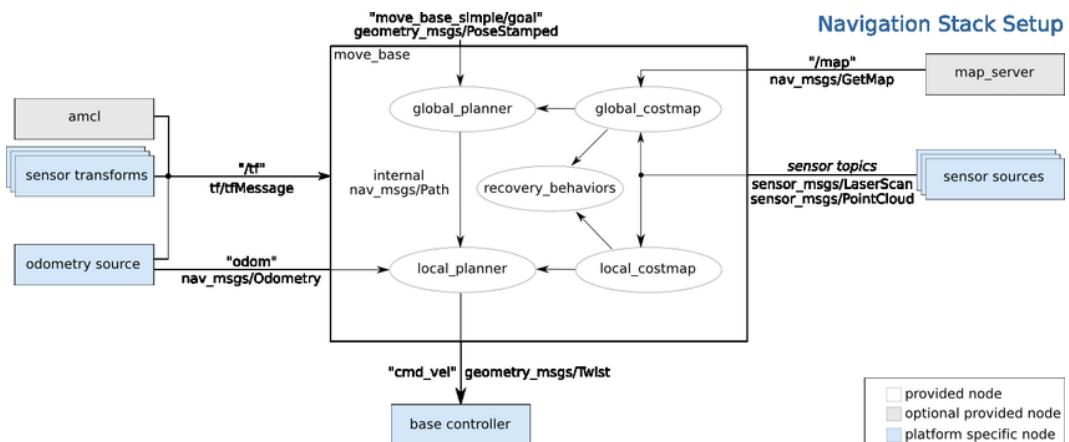
*Autonomous Movement* yang dibuat pada penelitian ini dihasilkan dari pembacaan sensor pada tahap persepsi. Data pembacaan dari sensor tersebut berupa *visual odometry* yang akan di-*publish* oleh sebuah *node publisher* dan kemudian akan di *subscribe* oleh *node subscriber* yaitu *node* yang mengeksekusi program *autonomous movement*. Berikut adalah gambar blok diagram dari proses dari *Autonomous Movement* Gambar 28.



Gambar 23 Hirarki sistem navigasi *Waiter-Bot*

### 3.4.3 Navigation Stack

Sistem kendali yang digunakan merupakan kendali *close loop* dengan mempertimbangkan pengaruh rintangan terhadap kecepatan roda robot dalam kemampuan merencanakan jalur terdekat dan teraman dalam sistem navigasinya. Secara hirarki navigasi robot otonom *Waiter-Bot* dapat dilihat pada Gambar 24.



Gambar 24 Blok Diagram *Autonomous Movement*

Terdapat beberapa hal yang perlu diperhatikan dalam proses navigasi mulai dari perencanaan sensor, *gmaping*, perencanaan jalur lokal dan perencanaan jalur global. Adapun *navigation stack* pada *Waiter-Bot* seperti Gambar 24 *navigation stack setup*. Dari Gambar 2 terlihat bahwa AMCL, sensor transform serta odometry source digunakan sebagai lokalisasi dari posisi robot terhadap lingkungan dan informasi dari odometry digunakan saat melakukan pencarian jalur lokal.

*Global\_planner* didapat dari informasi *global\_costmap* yang merupakan proses dari rtabmap yang telah dibuat sebelumnya oleh proses *mapping*. Selain itu dalam mencari *global\_costmap* dan *local\_costmap* diperlukan informasi dari sensor-sensor yang terdapat pada robot. Informasi dari *global\_planner* ini akan dipecah-pecah dan menjadi informasi pada *local\_planner* yang nantinya akan menentukan gerak dari sistem navigasi *Waiter-Bot*. Berikut tutorial perancangan algoritma dalam membangun sistem navigasi robot otomatis berbasis ROS.

#### a. Membuat *Package*

Langkah pertama untuk tutorial ini adalah membuat sebuah *package* di mana akan menyimpan semua konfigurasi dan meluncurkan file untuk *navigation stack*. *Package* ini akan memiliki ketergantungan pada *package* apa pun yang digunakan untuk memenuhi persyaratan di bagian pengaturan robot di atas serta pada *package move\_base* yang berisi antarmuka tingkat tinggi ke tumpukan navigasi. Jadi, lokasi untuk *package* dan jalankan perintah sebagai berikut:

```
catkin_create_pkg my_robot_name_2dnav move_base my_tf_configuration_dep my_odom_configuration_dep my_sensor_configuration_dep
```

Perintah ini akan membuat sebuah *package* dengan dependensi yang diperlukan untuk menjalankan tumpukan navigasi pada robot.

#### b. Membuat Konfigurasi Robot

Apabila sudah memiliki folder untuk semua file konfigurasi dan peluncuran, Buatlah file *roslaunch* yang akan memunculkan semua perangkat keras dan mengubah publikasi yang dibutuhkan robot. Kemudian tempelkan cuplikan berikut ini ke dalam file bernama *waiterbot\_configuration.launch*.

```
<launch>
  <node pkg="sensor_node_pkg" type="sensor_node_type" name="sensor_node_name" output="screen">
    <param name="sensor_param" value="param_value" />
  </node>
  <node pkg="odom_node_pkg" type="odom_node_type" name="odom_node" output="screen">
    <param name="odom_param" value="param_value" />
  </node>
```

```

<node pkg="transform_configuration_pkg" type="transform_configuration_type" name="transform_configuration_name" output="screen">

    <param name="transform_configuration_param" value="param_value" />

</node>
</launch>

```

Pada bagian ini, akan dipanggil semua node sensor yang akan digunakan robot untuk navigasi. Dapat diganti "*sensor\_node\_pkg*" dengan nama *package* untuk driver ROS untuk sensor, "*sensor\_node\_type*" dengan jenis driver untuk sensor yang ada, "*sensor\_node\_name*" dengan nama yang diinginkan untuk *node* sensor, dan "*sensor\_param*" dengan parameter apa pun yang mungkin diperlukan oleh node yang ada.

```

</node>

<node pkg="odom_node_pkg" type="odom_node_type" name="odom_node" output="screen">

<param name="odom_param" value="param_value" />

</node>

```

Pada bagian ini, dilakukan pemanggilan *odometry* untuk basis. Sebelumnya perlu dilakukan mengganti spesifikasi *pkg*, *tipe*, *nama*, dan *param* dengan spesifikasi yang relevan dengan node yang sudah dibuat.

```

<param name="transform_configuration_param" value="param_value" />

</node>

```

Pada bagian ini, kita akan meluncurkan konfigurasi transformasi untuk robot. Sekali lagi, perlu harus mengganti spesifikasi *pkg*, *type*, *name*, dan *param* dengan spesifikasi yang relevan dengan node yang diluncurkan.

### c. Konfigurasi *global\_costmap*

Pada bagian ini akan dibuat sebuah file untuk menyimpan opsi konfigurasi khusus untuk *global\_costmap*. Buka editor dengan file *global\_costmap\_params.yaml* dan tempelkan teks berikut ini:

```

global_costmap:
    global_frame: /map
    robot_base_frame: base_link

```

```
update_frequency: 5.0
static_map: true
```

Parameter "*global\_frame*" mendefinisikan *frame* koordinat apa yang harus dijalankan oleh *costmap*, dalam kasus ini, kita akan memilih *frame /map*. Parameter "*robot\_base\_frame*" mendefinisikan kerangka koordinat yang harus dirujuk oleh *costmap* untuk basis robot. Parameter "*update\_frequency*" menentukan frekuensi, dalam Hz, di mana *costmap* akan menjalankan perulangan pembaruan. Parameter "*static\_map*" menentukan apakah *costmap* harus menginisialisasi dirinya sendiri berdasarkan peta yang dilayani oleh *map\_server*. Jika tidak menggunakan peta atau server peta yang sudah ada, atur parameter *static\_map* menjadi *false*.

#### d. Konfigurasi *local\_costmap*

Selanjutnya dengan membuat file yang akan menyimpan opsi konfigurasi khusus untuk *costmap* lokal. Buka editor dengan file *local\_costmap\_params.yaml* dan tempelkan teks berikut ini:

```
local_costmap:
  global_frame: odom
  robot_base_frame: base_link
  update_frequency: 5.0
  publish_frequency: 2.0
  static_map: false
  rolling_window: true
  width: 6.0
  height: 6.0
  resolution: 0.05
```

Parameter "*global\_frame*", "*robot\_base\_frame*", "*update\_frequency*", dan "*static\_map*" sama seperti yang dijelaskan di bagian Konfigurasi Global di atas. Parameter "*publish\_frequency*" menentukan kecepatan, dalam Hz, di mana *costmap* akan mempublikasikan informasi visualisasi. Mengatur parameter "*rolling\_window*" menjadi *true* berarti bahwa *costmap* akan tetap berpusat di sekitar robot ketika robot bergerak di lingkungan global. Parameter "*width*," "*height*," dan "*resolution*" mengatur lebar (meter), tinggi (meter), dan resolusi

(meter/sel) dari costmap. Perhatikan bahwa tidak masalah jika resolusi grid ini berbeda dengan resolusi peta statis yang dibuat, tetapi sebagian besar waktu kita cenderung mengaturnya sama.

#### e. Konfigurasi *Base Local Planner*

*Base\_local\_planner* bertanggung jawab untuk mengkomputasi perintah kecepatan untuk dikirim ke basis *mobile robot* yang diberikan rencana dari sistem *high level*. Oleh karena itu, perlu mengatur beberapa opsi konfigurasi berdasarkan spesifikasi robot kita agar semuanya berjalan. Buka file bernama *base\_local\_planner\_params.yaml* dan tempelkan teks berikut ini ke dalamnya:

```
TrajectoryPlannerROS:
    max_vel_x: 0.45
    min_vel_x: 0.1
    max_vel_theta: 1.0
    min_in_place_vel_theta: 0.4

    acc_lim_theta: 3.2
    acc_lim_x: 2.5
    acc_lim_y: 2.5

    holonomic_robot: true
```

Bagian pertama dari parameter di atas mendefinisikan batas kecepatan robot.

Bagian kedua mendefinisikan batas akselerasi robot.

#### f. Menyatukan Konfigurasi Sistem *Navigation Stack*

Setelah memiliki semua berkas konfigurasi, maka selanjutnya perlu menyatukan semuanya ke dalam berkas peluncuran untuk tumpukan navigasi. Buka editor dengan berkas *move\_base.launch* dan tempelkan teks berikut ini ke dalamnya:

```
<launch>
    <master auto="start"/>
    <!-- Run the map server -->
    <node name="map_server" pkg="map_server" type="map_serve
r" args="$(find my_map_package) /my_map.pgm my_map_resolutio
n"/>
```

```

<!-- Run AMCL -->

<include file="$(find amcl)/examples/amcl_omni.launch"
/>

<node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">

    <rosparam file="$(find my_robot_name_2dnav)/costmap_common_params.yaml" command="load" ns="global_costmap" />

    <rosparam file="$(find my_robot_name_2dnav)/costmap_common_params.yaml" command="load" ns="local_costmap" />

    <rosparam file="$(find my_robot_name_2dnav)/local_costmap_params.yaml" command="load" />

    <rosparam file="$(find my_robot_name_2dnav)/global_costmap_params.yaml" command="load" />

    <rosparam file="$(find my_robot_name_2dnav)/base_local_planner_params.yaml" command="load" />

</node>

</launch>

```

#### g. Menjalankan *Navigation Stack*

Sekarang setelah kita menyiapkan semuanya, kita dapat menjalankan tumpukan navigasi. Untuk melakukan ini, kita membutuhkan dua terminal pada robot. Pada satu terminal, kita akan menjalankan file `my_robot_configuration.launch` dan pada terminal lainnya kita akan menjalankan file `move_base.launch` yang baru saja kita buat.

Terminal 1

```
roslaunch my_robot_configuration.launch
```

Terminal 2

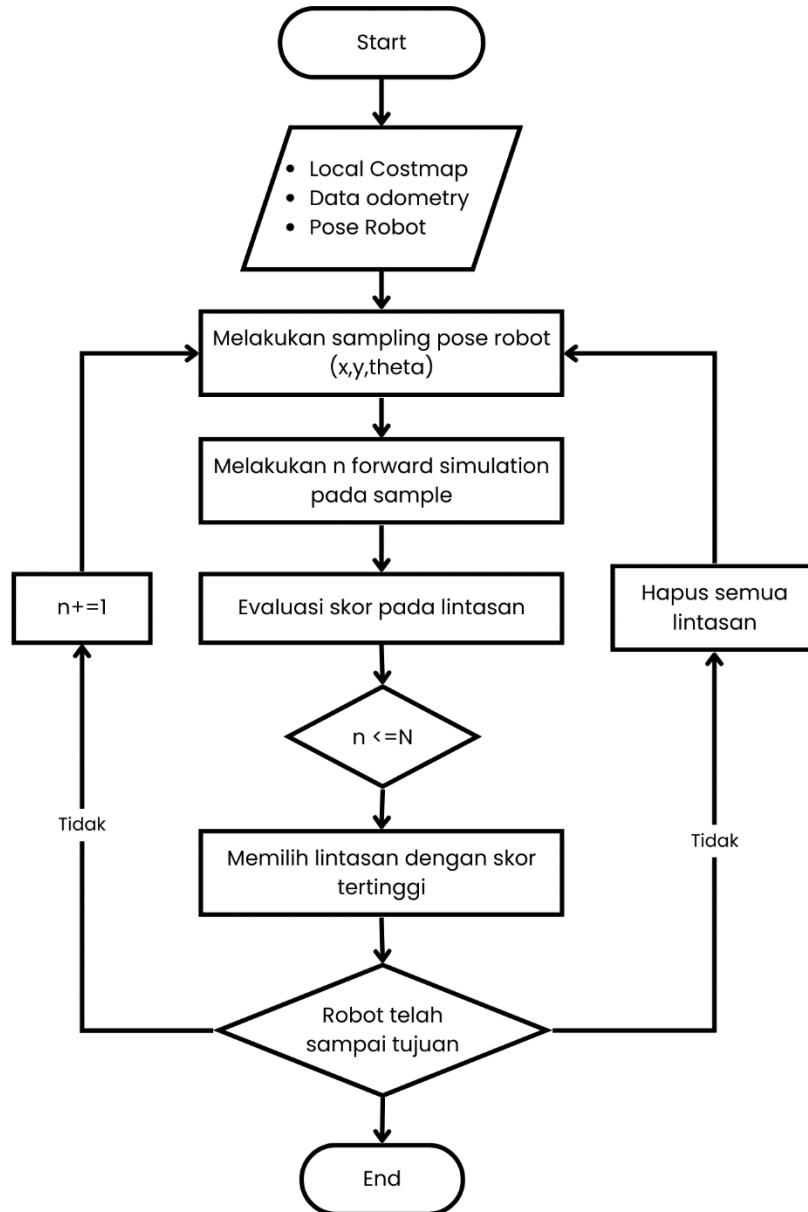
```
roslaunch my_robot_configuration.launch
```

### 3.5 Rancangan Pengujian

#### 3.5.1 Perencanaan Local Planner

Setelah rencana global dibuat, perencana lokal menerjemahkan jalur ini ke dalam perintah kecepatan untuk motor robot. Perencana lokal melakukan hal ini dengan membuat fungsi nilai di sekitar robot, mengambil sampel dan mensimulasikan lintasan di dalam ruang ini, memberi nilai pada setiap lintasan yang disimulasikan berdasarkan hasil yang diharapkan, mengirimkan lintasan dengan nilai tertinggi sebagai perintah kecepatan ke robot dan mengulanginya

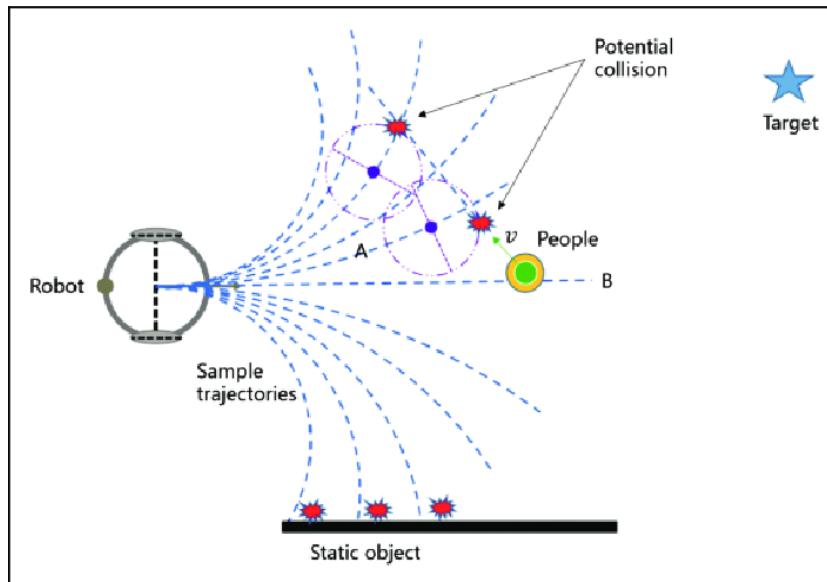
hingga tujuan tercapai. Alasan mengapa perencana lokal bekerja dengan cara ini adalah karena perencana lokal ditulis dengan sangat umum untuk digunakan oleh robot yang mungkin memiliki jejak kaki yang tidak beraturan, geometri kemudi *Ackerman*, pelengkap, dan konfigurasi lain yang tidak akan bekerja menggunakan algoritma yang dirancang khusus untuk robot dengan bentuk yang sederhana, sistem penggerak diferensial dan tanpa pelengkap.



Gambar 25 Flow chart perencanaan *local planner*

Dalam melakukan perencanaan jalur terpendek dalam cakupan lokal menggunakan metode yang telah ada di *library* ROS yakni *Dynamic Window*

*Approach* (DWA) yang membuat lintasan berdasarkan peta yang telah dibuat sebelumnya (Dieter, *et.al.*, 1997). DWA akan membuat beberapa macam lintasan dan membuat suatu fungsi nilai untuk menghitung jarak terdekat serta menghindari adanya penghalang baik itu penghalang yang statis maupun yang dinamis. Nilai dari jarak yang dibuat berdasarkan grid yang dibentuk oleh DWA. Fungsi nilai ini berupa  $dx$ ,  $dy$  dan  $d\theta$  yang akan secara simultan dikirimkan ke robot. Ide dasar dari algoritma DWA dapat dilihat pada Gambar 29.



Gambar 26 Visual *Dynamic Window Approach*

Dengan menggunakan algoritma DWA, robot dapat melakukan optimasi berbasis sampel. Ini mensimulasikan jalur dalam ruang kecepatan yang layak sesuai dengan panjang lintasan berdasarkan model gerakan robot. Pendekatan ini tidak dapat memprediksi pembalikan gerakan karena aksi kontrol dijaga konstan. Pendekatan ini memberikan kinerja yang layak untuk penggerak diferensial dan robot *omnidirectional*. Ini sangat cocok untuk evaluasi berbasis *grid*. DWA tidak terbatas dalam minimum lokal tidak seperti algoritma TEB. Solusi suboptimal berdasarkan tanpa pembalikan gerakan dan dukungan untuk biaya yang tidak mulus (Gupta, *et.al.* 2021).

Sangat sulit bagi robot untuk mencapai koordinat tujuannya secara tepat karena selalu ada sejumlah kecil kesalahan dalam gerakan fisik robot. Hal ini dapat menyebabkan robot berosilasi di sekitar tujuan karena terus berusaha mencapai lokasi yang tepat. Perilaku yang tidak diinginkan ini dapat dielakkan dengan

mengatur fungsi *latch* yang menyebabkan robot berhenti ketika berada dalam jarak tertentu dari tujuan.

### 3.5.2 Perencanaan *Global Planner*

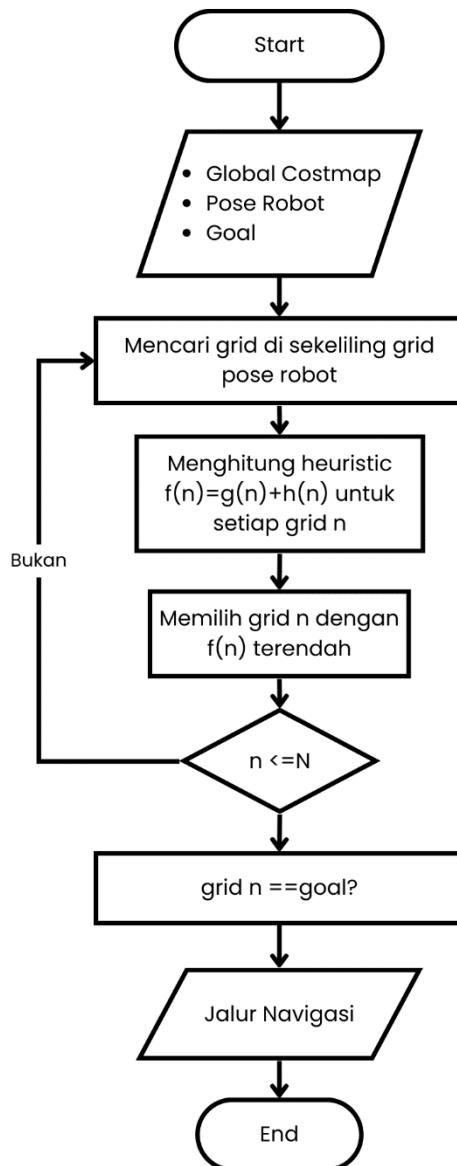
Setelah robot berhasil melokalisasi, robot dapat diberikan koordinat tujuan dan menggunakan perencana global untuk merencanakan jalur ke koordinat tersebut. Lingkungan direpresentasikan secara internal sebagai gambar lossless dua dimensi. Lingkungan berisi lima jenis entitas: ruang yang belum dijelajahi, ruang kosong yang diketahui, rintangan, radius inflasi, dan robot itu sendiri. Radius inflasi adalah penyangga yang memancar keluar dari semua rintangan. Robot memperlakukan radius inflasi sebagai rintangan dan tidak dapat memetakan jalur yang melaluinya. Radius inflasi sama dengan radius robot itu sendiri dan memiliki efek untuk memastikan bahwa robot selalu menghasilkan jalur dengan ruang yang cukup untuk membersihkan rintangan. Hal ini diimplementasikan sebagai penyangga di sekitar rintangan alih-alih diterapkan pada robot itu sendiri karena alasan kesederhanaan, dan mencapai efek yang sama.

Setelah melakukan lokalisasi, *Waiter-Bot* akan melakukan perencanaan jalur navigasi yang akan ditempuh menuju ke titik tujuan. Dalam melakukan perencanaan jalur sebelumnya diubah menjadi model diskrit. Terdapat tiga pendekatan dalam penentuan jalur navigasi ini yaitu pendekatan *road map*, pendekatan *cell decomposition* dan pendekatan *potential field*[5]. *Cell decomposition* merupakan pendekatan perencanaan jalur dengan memetakan sel-sel di bebagai titik hingga membentuk suatu rantai yang dapat dijadikan sebagai jalur. Sedangkan *potential field* merupakan suatu fungsi yang diterapkan di atas ruang bebas pada peta, di mana tujuan bertindak sebagai potensi yang menonjol (*sink*) dan rintangan bertindak sebagai potensi yang dijauhi (*source*). Jalur kemudian dibentuk berdasarkan turunan dari medan potensial, di mana arah yang paling menonjol atau grid paling banyak diikuti. Pendekatan ini pertama kali dikembangkan untuk penghindaran tabrakan dengan objek (Khatib, 1986).

Pendekatan *cell decomposition* akan membedakan antara area geometri yang nantinya disebut dengan sel yang bebas dan yang terisi oleh objek. Algoritma dari pendekatan *cell decomposition* yaitu:

- a. Langkah pertama adalah membagi peta menjadi daerah-daerah kecil dan terhubung yang disebut dengan sel.
- b. Tentukan sel-sel yang bebas yang berdekatan dan buat konektifitas antara sel-sel yang bebas tersebut. Sel-sel yang tidak bebas disebut dengan sel yang terdekomposisi.
- c. Tentukan sel konfigurasi awal dan tujuan dan buat grafik konektifitas antara kedua sel tersebut.
- d. Dari urutan sel tersebut hitung jalur di setiap sel yang dilewati serta buat urutan gerakan untuk urutan terbaik. Pendekatan berikutnya adalah pendekatan *potential field*.

Pendekatan ini memodelkan robot menjadi suatu partikel yang bergerak dibawah pengaruh medan potensial yang ditentukan oleh tujuan serta beberapa rintangan (Borenstein, *et. al.*, 1990). Setiap pergerakan robot akan ditentukan oleh medan potensial yang berada di lokasinya. Kelebihan dari metode ini adalah komputasi yang dihasilkan tidak terlalu besar karena setiap pergerakan robot hanya tergantung pada pendekatan medan potensialnya. Selain itu pada pendekatan ini juga memiliki respon yang baik pada halang-rintang yang dinamis karena setiap adanya halang-rintang maka medan potensial akan selalu berubah.



Gambar 27 Flow chart pengujian global planner

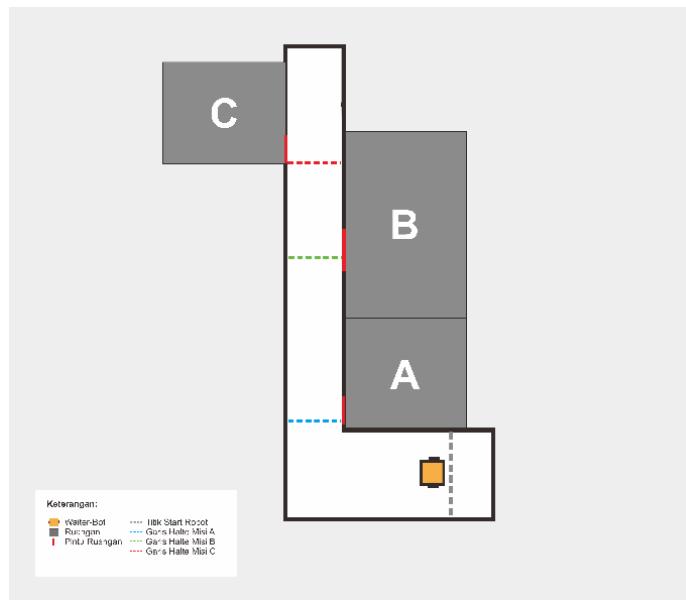
### 3.5.3 Skenario Pengujian Navigasi Robot

Pengujian navigasi *Waiter-Bot* Secara keseluruhan Pengujian sistem *Waiter-Bot* untuk mengetahui kinerja fungsional komponen keseluruhan. Fokus pada pengujian navigasi ini untuk kondisi lingkungan statis dan dinamis. Dalam simulasi pengoperasian *Waiter-Bot* akan dilakukan pengujian pada saat kondisi dengan rintangan yang statis dan dinamis. Lingkungan statis pada pengujian ini merupakan pengujian dengan mengetahui kemampuan robot melakukan navigasi berdasarkan peta yang telah berhasil dibangun pada tahap pemetaan. Sementara pengujian lingkungan dinamis adalah pengujian sistem navigasi robot untuk mengetahui

kemampuannya dalam mendekripsi dan menghindari objek tambahan diluar hasil pemetaan yang berperan sebagai penghalang.

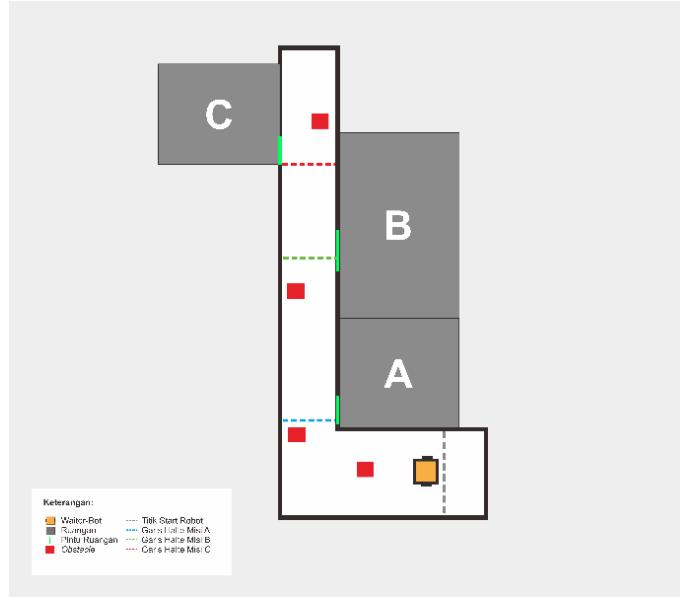
Kemudian dilakukan juga pengamatan atas perhitungan nilai kecepatan linear (m/s) dan kecepatan angular (rad/s) pada saat robot melakukan navigasi. Luaran yang diharapkan pada pengu bahwa robot dapat mengenali dan adaptif dengan kondisi lingkungan tersebut dalam menjalankan misinya. Berikut desain denah skenario navigasi robot pada lingkungan statis dan dinamis.

Terdapat tiga misi tujuan yang akan ditempuh oleh robot otonom *Waiter-Bot* yakni misi A, B dan C. Robot akan melakukan perjalanan secara otonom dengan terlebih dahulu melakukan perencanaan jalur global dan lokal pada sistem navigasi. *Waiter-Bot* melakukan perjalanan secara otonom dari titik start ke titik misi A, B dan C yang memiliki jarak tempuh yang berbeda beda tanpa ada objek tambahan diluar hasil pemetaan.



Gambar 28 Desain denah pengujian navigasi robot pada lingkungan statis

Selanjutnya pada pengujian skenario navigasi lingkungan dinamis oleh *Waiter-Bot* melakukan perjalanan secara otonom dari titik *start* ke titik misi A, B dan C yang memiliki jarak tempuh yang berbeda beda dengan perlakuan tambahan objek sebagai *obstacle* diluar hasil pemetaan yang telah dilakukan.



Gambar 29 Desain denah pengujian navigasi robot pada lingkungan dinamis

Pengamatan performa *Waiter-Bot* dilakukan pendekatan *roadmap* membuat suatu set kurva satu dimensi yang masing-masing akan menghubungkan satu *node*. Hubungan antara *node* akan menjadi suatu jalur dan setiap jalur akan diberikan bobot-bobot sesuai dengan parameter-parameter misalnya seperti jarak, hambatan ataupun parameter lainnya. Untuk mencari jarak terpendek menggunakan pendekatan *roadmap* maka akan dicari pendekatan bobot paling minimum yang berarti memiliki jarak serta hambatan paling minimum. Selain itu, kecepatan konstan motor robot dengan interval minimum dan maksimum telah diukur menggunakan *tachometer*. Kombinasi dari kecepatan linear dan kecepatan angular digunakan untuk mengontrol pergerakan robot dalam navigasi otonom. Misalnya jika robot perlu mengikuti jalur lengkung, maka akan menggabungkan kecepatan linear dan kecepatan angular dapat digunakan dalam mengetahui kecepatan resultant robot.

$$V_{resultan} = \sqrt{V^2_{linear} + (R \cdot V_{angular})^2}$$

Dimana:

$V_{resultan}$  adalah kecepatan resultant robot (m/s)

$V_{linear}$  adalah kecepatan linar robot (m/s)

$V_{angular}$  adalah kecepatan angular (rad/s)

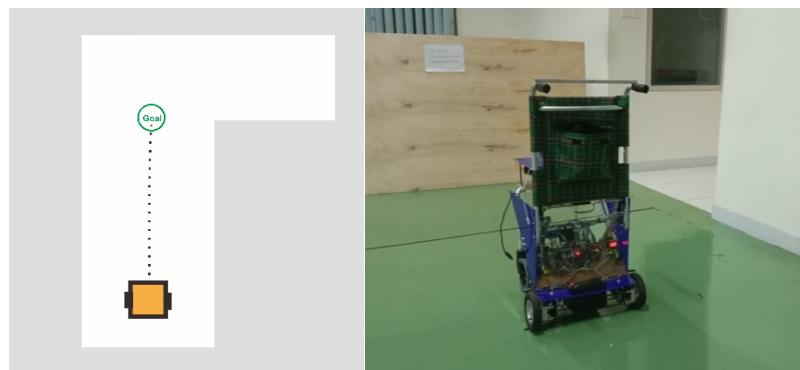
R adalah jari-jari lintasan (radius) dari gerakan putar robot (m)

## BAB IV

### ANALISIS DAN PEMBAHASAN

#### 4.1 Pengujian Deviasi Kecepatan Linear Robot

Deviasi kecepatan pada robot merujuk pada perbedaan antara kecepatan yang diinginkan dalam hal ini nilai kecepatan yang diatur dalam sistem navigasi ROS terhadap kecepatan eksisting yang terjadi pada robot. Sebelum melakukan pengujian navigasi robot secara otomatis, terlebih dahulu dilakukan pengujian deviasi robot dalam mengeksekusi perintah nilai kecepatan dari pengaturan ROS untuk memperoleh deviasi nilai kecepatan sehingga dapat menjadi acuan dalam pengaturan nilai kecepatan linear robot.



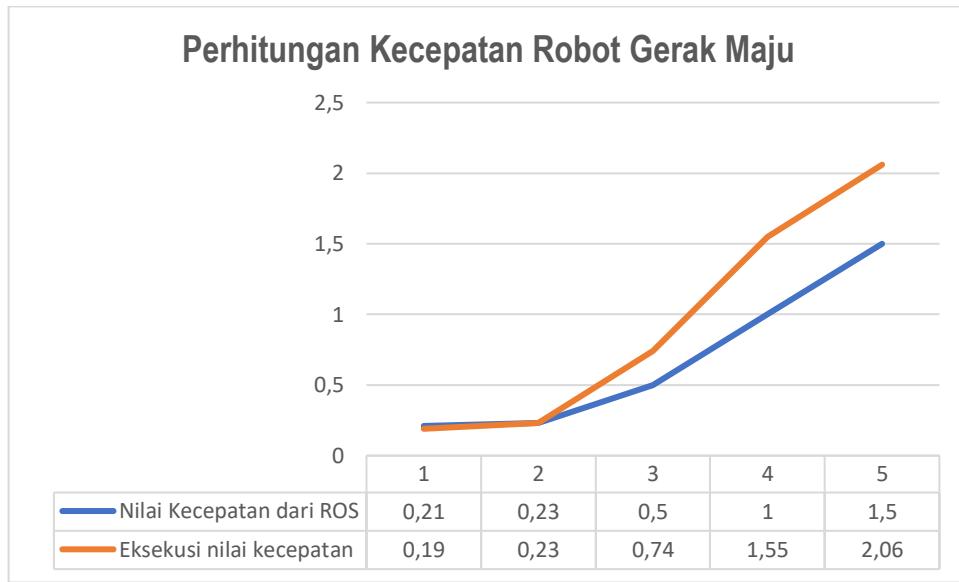
Gambar 30 Pengujian deviasi kecepatan robot

Tabel 5 Deviasi eksekusi perintah kecepatan linear robot

Perc.	Jarak Tempuh (m)	Perintah Nilai Kecepatan Dalam ROS (m/s)	Eksekusi Nilai Kecepatan dari ROS (m/s)	Persentase Eror (%)
1	1	0,21	0,19	0,09
2	1	0,23	0,23	0,00
3	1	0,50	0,74	0,48
4	1	1,00	1,55	0,55
5	1	1,50	2,06	0,37

Dari tabel di atas menunjukkan bahwa pengujian gerak maju robot diberikan perlakuan pengaturan nilai kecepatan pada ROS lalu kemudian dieksekusi oleh robot. Nilai kecepatan diberikan dari nilai 0,21 m/s hingga 1,50 m/s dengan skenario pengujian pada jarak tempuh konstan 1 meter dengan mengabaikan *node*

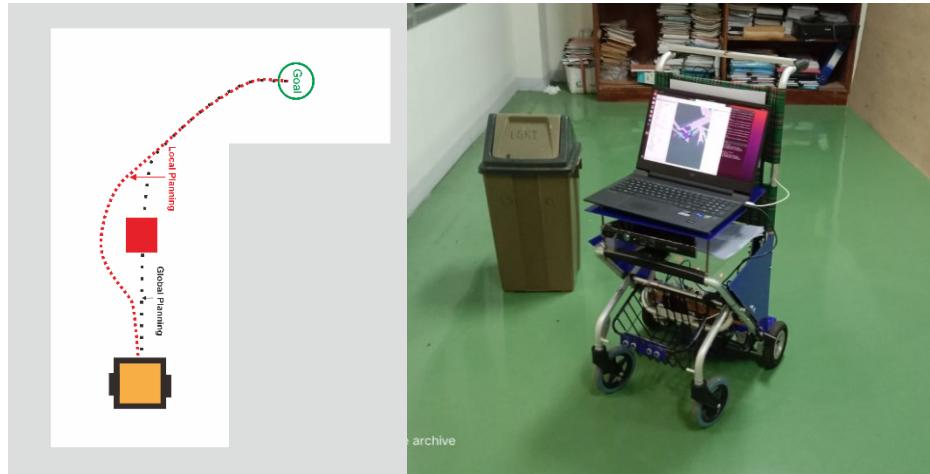
*navigation stack* dalam mendekteksi objek penghalang. Adapun grafik deviasi antara perintah nilai kecepatan dan nilai eksekusi kecepatan robot sebagai berikut.



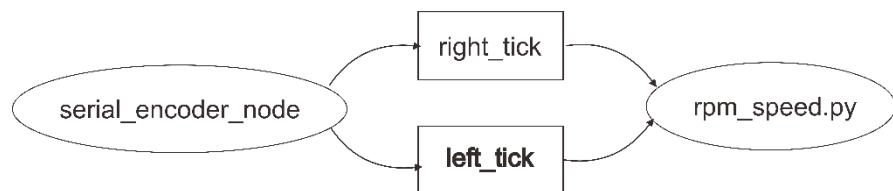
Gambar 31 Grafik deviasi kecepatan linear robot

Dari grafik kecepatan linear di atas, robot mengeksekusi perintah nilai kecepatan pada ROS secara linear dan berbanding lurus antara nilai kecepatan terhadap waktu tempuh. Didapatkan persentase eror paling tinggi 0,55 % yakni pada pengujian keempat. Adapun nilai perintah kecepatan yang lebih presisi yakni dengan nilai kecepatan 0,21-0,23 m/s dengan persentase eror 0,09 %. Sehingga interval nilai tersebut yang diperintahkan dalam eksekusi pergerakan robot dalam melakukan navigasi. Hasil yang diperoleh dalam pengujian sistem navigasi *Waiter-Bot* yakni kemampuan robot merencanakan jalur dan efektifitas dalam mencapai *goal* nya.

## 4.2 Pengujian Respon Kecepatan Robot Terhadap Objek Penghalang

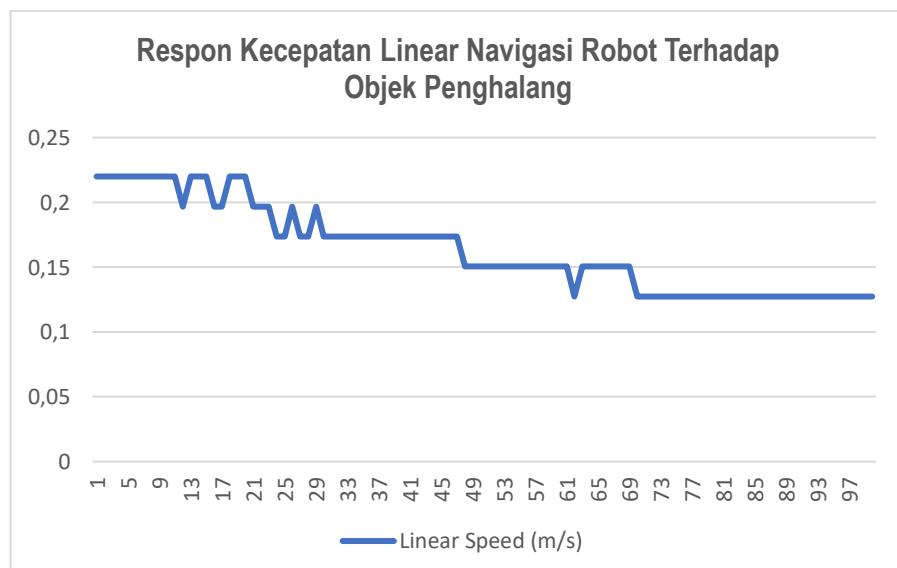


Gambar 32 Pengujian respon kecepatan robot terhadap objek penghalang



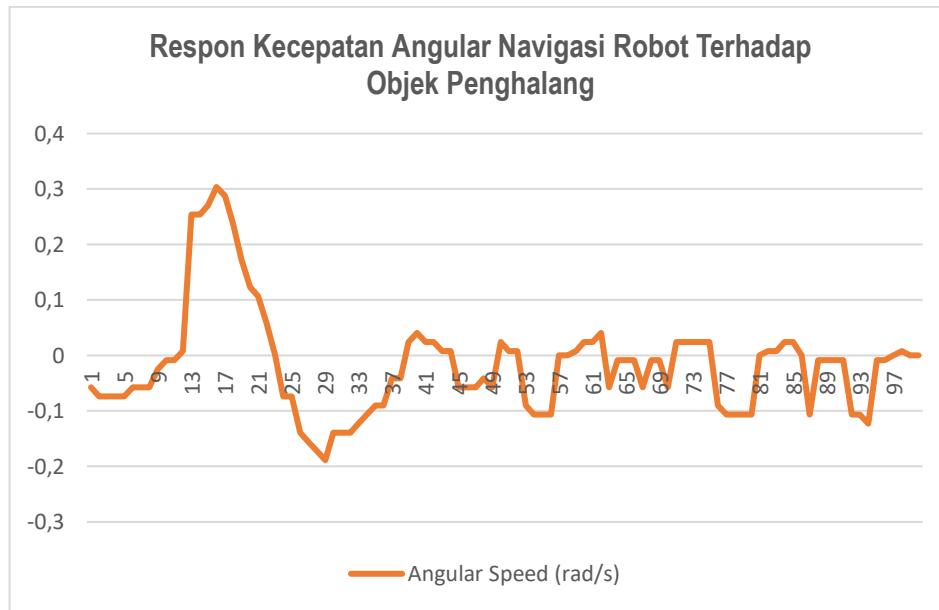
Gambar 33 sistem komunikasi ROS node pembacaan kecepatan robot

Pada skenario pengujian ini, *Waiter-Bot* melakukan navigasi dengan jarak tempuh 5,16 meter. Adapun data kecepatan resultan robot dapat dilihat pada Lampiran 1, sehingga diperoleh grafik sebagai berikut:



Gambar 34 Grafik kecepatan linear respon kecepatan

Nilai kecepatan linear robot jika semakin besar berarti robot sedang mengikuti trayektori dengan kecepatan maksimal begitupun sebaliknya. Dari grafik kecepatan linear robot pada saat melakukan navigasi menggambarkan bahwa kecepatan linear robot beroperasi pada interval 0.12-0.22 m/s selama perjalanan navigasi dengan jumlah data sampling yang diambil yakni hanya 100 *row* data dengan tujuan untuk melihat kondisi perubahan gerak robot dalam menjalankan navigasi secara *autonomous*.

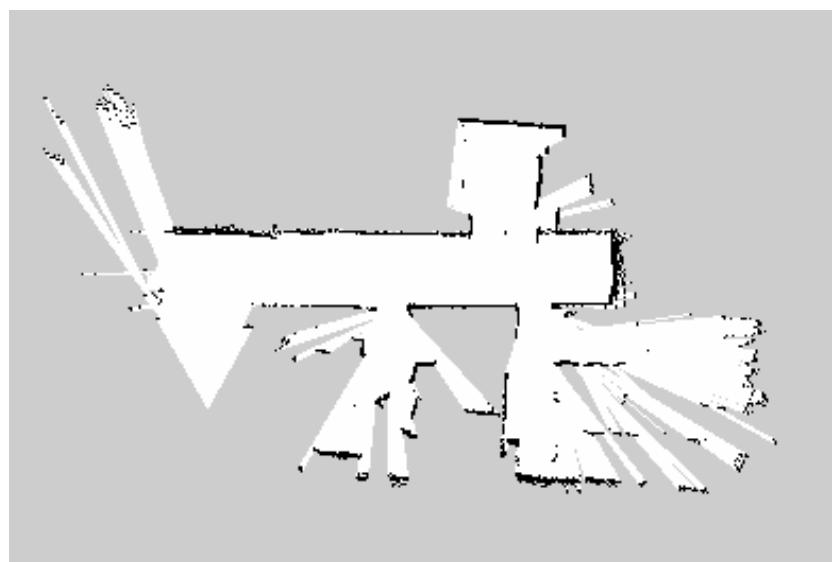


Gambar 35 Grafik kecepatan angular robot terhadap objek penghalang

Kecepatan angular mengukur seberapa cepat robot berputar atau berbelok. Nilai kecepatan ini diperlukan untuk mengubah arah robot. Kecepatan angular yang tinggi memungkinkan robot untuk berbelok dengan cepat dan hal penting dalam situasi di mana navigasi yang akurat dan berbelok cepat diperlukan. Pengaturan kecepatan angular yang terlalu tinggi dapat mengakibatkan robot bergerak tidak stabil atau sulit dikendalikan. Pada data grafik kecepatan angular robot terhadap objek penghalang ini menggambarkan bahwa interval nilai kecepatan angular robot beroperasi pada interval -0,2 hingga 0,3 rad/s dengan jumlah data sampling yang diambil yakni 100 *row* data. Dimana tanda negatif menunjukkan bahwa kecepatan angular robot sedang menghindari objek atau melakukan pembelokan dalam mengikuti trayektori lintasan jalur lokal.

### 4.3 Pengujian Skenario Navigasi Robot Pada Lingkungan Statis dan Dinamis

Dalam kondisi pengujian ini dibedakan antara kondisi robot menjalankan misi dalam lingkungan statis maupun dinamis. Skenario lingkungan statis merupakan kondisi pengujian menggunakan peta yang persis mengikuti hasil *gmaping* tanpa adanya objek tambahan. Sedangkan skenario lingkungan dinamis yang dimaksud disini adalah pengujian dengan penambahan objek penghalang tambahan diluar hasil *gmaping*. Pada tahap *gmaping* diperoleh gambar peta dengan ukuran 400 x 400 pixel sebagai representasi dari area pengujian eksisting yang telah dilakukan sebagai berikut Gambar 40.



Gambar 36 Peta pengujian navigasi robot



Gambar 37 *Waiter-Bot* membuat jalur trayektori navigasi dalam ROS

#### 4.3.1 Kemampuan *Waiter-Bot* melakukan Navigasi Pada Lingkungan Statis



Gambar 38 Navigasi robot pada skenario lingkungan statis

Misi A: Jalur Lintasan Tanpa Tambahan Objek Penghalang

Tabel 5 Pengujian skenario lingkungan statis navigasi *Waiter-Bot* misi A

Perc.	Jarak Tempuh (m)	Waktu Tempuh (s)	Kecepatan Linear (m/s)
1	5.3	87	0.060
2	5.3	74	0.071
3	5.3	86	0.061
$\bar{x}$		82.33	0.064

Tabel 6 Pengujian skenario lingkungan statis navigasi *Waiter-Bot* misi B

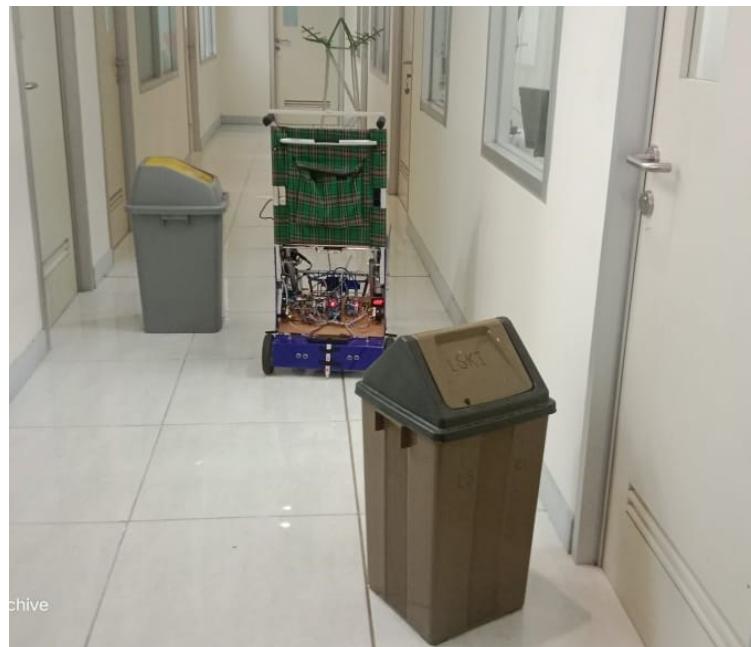
Perc.	Jarak Tempuh (m)	Waktu Tempuh (s)	Kecepatan Linear (m/s)
1	9.2	155	0.059
2	9.2	148	0.062
3	9.2	154	0.060
$\bar{x}$		152.33	0.061

Tabel 7 Pengujian skenario lingkungan statis navigasi *Waiter-Bot* misi C

Perc.	Jarak		Kecepatan Linear (m/s)
	Tempuh (m)	Waktu Tempuh (s)	
1	12.2	241	0.050
2	12.2	202	0.060
3	12.2	243	0.051
	$\bar{x}$	228.67	0.053

Pada pengujian sistem navigasi robot otomatis skenario ini yakni *Waiter-Bot* menjalankan misi A dengan rata-rata waktu tempuh 1 menit 37 detik pada jarak tempuh 530 cm. Dalam menjalankan misi B, diperoleh waktu tempuh rata-rata pada jarak 920 cm yakni 2 menit 53 detik. Kemudian saat menempuh misi C dengan jarak 1202 cm, diperoleh waktu 3 menit 81 detik. Seluruh pengujian navigasi robot dalam skenario perencanaan global berhasil mencapai goal sesuai trayektori yang dibangun dalam ROS.

#### 4.3.2 Kemampuan Navigasi *Waiter-Bot* Pada Lingkungan Dinamis

Gambar 39 Uji sistem navigasi *Waiter-Bot* pada lingkungan dinamis

Tabel 8 Pengujian skenario lingkungan dinamis navigasi *Waiter-Bot* misi A

<b>Perc.</b>	<b>Jarak</b>		<b>Kecepatan</b>
	<b>Tempuh (m)</b>	<b>Waktu</b>	
		<b>Tempuh (s)</b>	<b>Linear (m/s)</b>
1	5.3	86	0.061
2	5.3	90	0.059
3	5.3	91	0.058
$\bar{x}$		89	0.059

Tabel 9 Pengujian skenario lingkungan dinamis navigasi *Waiter-Bot* misi B

<b>Perc.</b>	<b>Jarak</b>		<b>Kecepatan</b>
	<b>Tempuh (m)</b>	<b>Waktu</b>	
		<b>Tempuh (s)</b>	<b>Linear (m/s)</b>
1	9.3	185	0.049
2	9.3	190	0.048
3	9.3	191	0.048
$\bar{x}$		188.67	0.048

Tabel 10 Pengujian skenario lingkungan dinamis navigasi *Waiter-Bot* misi C

<b>Perc.</b>	<b>Jarak</b>		<b>Kecepatan</b>
	<b>Tempuh (m)</b>	<b>Waktu</b>	
		<b>Tempuh (s)</b>	<b>Linear (m/s)</b>
1	12.2	245	0.050
2	12.2	248	0.049
3	12.2	250	0.048
$\bar{x}$		247.67	0.049

Pengujian navigasi *Waiter-Bot* dalam skenario lingkungan dinamis, robot menjalankan misi A dengan jarak 530 cm dengan waktu tempuh rata-rata misi A yakni 1 menit 48 detik. Misi B dengan jarak tempuh 920, diperoleh waktu tempuh rata-rata 3 menit 14 detik. Serta misi C dengan jarak tempuh 1202 cm, diperoleh waktu tempuh rata-rata 4 menit 12 detik. Sehingga pada pengujian sekanrio 1 dan 2 dinyatakan berhasil karena robot dapat mengenali objek dan merencanakan jalur trayektori terdekat dan teraman.

#### 4.4 Sistem Komunikasi Node Pada Navigation Stack Dalam ROS

Dalam menjalankan topik *navigation stack* pada ROS, dapat diketahui data status antar *node* yang berkomunikasi pada sistem navigasi *Waiter-Bot*.

##### 4.4.1 Data Status Komunikasi Node Saat Robot Tidak Bergerak



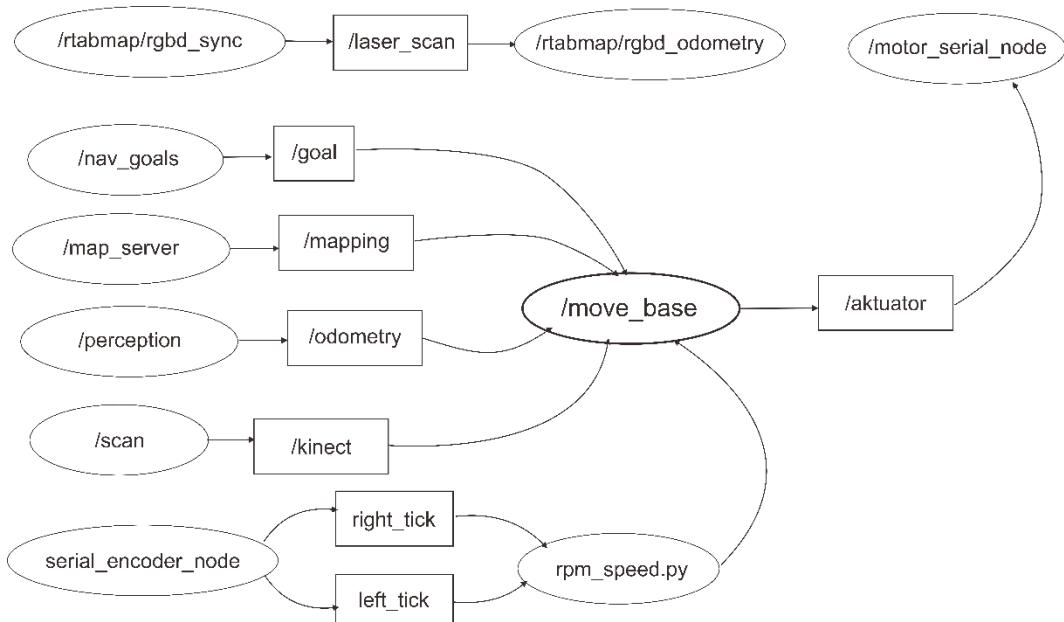
Gambar 40 Komunikasi node saat robot tidak bergerak

Berikut tabel keterangan topik yang digunakan dalam sistem navigasi robot berbasis ROS.

Tabel 11 Data status komunikasi node saat robot tidak bergerak

Topik	Publisher	Subscriber
Remot Kontrol	/joy_node	/x360joy_node
Velocity	/x360joy_node	/motor_serial_node

##### 4.4.2 Data Status Komunikasi Node Robot Saat Berjalan



Gambar 41 Komunikasi node saat robot bergerak

Dari gambar tersebut, diperoleh data aktivasi status node sebagai berikut.

Tabel 12 Data Status Komunikasi Node Saat Robot Berjalan

<b>Topik</b>	<b>Publisher</b>	<b>Subscriber</b>
Pemetaan lingkungan	<i>/map_server</i>	<i>/move_base</i>
<i>Odometry</i>	<i>/perception_node</i>	<i>/move_base</i>
Pembacaan kinect	<i>/scan</i>	<i>/move_base</i>
<i>Laser scanner</i>	<i>/rtabmap/rgbd_sync</i>	<i>/rtabmap/rgbd_odometry</i>
Penentuan goal	<i>/nav_goal</i>	<i>/move_base</i>
Perintah aktuator	<i>/move_base</i>	<i>/motor_serial_node</i>
Instruksi nilai kecepatan	<i>/serial_encoder_node</i>	<i>/rpm_speed.py</i>

Dalam pengamatan sistem komunikasi antar *node* dalam *ROS Navigation Stack*, diketahui jumlah *node* yang aktif saat robot tidak bergerak yakni 4 *node*. Sedangkan pada saat robot mulai bergerak secara otonom, diketahui jumlah *node* yang aktif yakni 14 *node* pada 7 topik pendukung navigasi.

## BAB V

### KESIMPULAN DAN SARAN

#### 5.1 Kesimpulan

Dari hasil pengujian dan Analisa sistem navigasi robot otonom *Waiter-Bot* berbasis ROS pada Bab 4, penulis mengambil beberapa kesimpulan sebagai berikut:

1. Deviasi kecepatan pada robot merujuk pada perbedaan antara kecepatan yang diinginkan dalam hal ini nilai kecepatan yang diatur dalam sistem navigasi ROS. Pada pengujian yang dilakukan diperoleh bahwa perintah nilai kecepatan linear yang ideal untuk digunakan dalam sistem navigasi robot otonom yakni interval 0.21 – 0.23 m/s dengan persentase eror 0.04 %. Adapun pengujian respon kecepatan linear robot terhadap objek penghalang, diperoleh data kecepatan linear robot yakni interval 0.12-0.22 m/s serta kecepatan angular robot diperoleh nilai interval -0.13 hingga 0.17 rad/s. Apabila robot mendeteksi objek penghalang maka sistem navigasi pada *based local planner* ROS akan memberikan instruksi nilai kecepatan linear lebih kecil. Begitupun nilai angular, apabila bernilai negatif atau bahkan positif maka roda robot menunjukkan bahwa robot sedang melakukan penyesuaian navigasi dengan kompnesasi adanya manuver.
2. Dalam pengujian skenario lingkungan statis, robot melakukan navigasi menuju misi A dengan rata-rata waktu tempuh 1 menit 37 detik pada jarak tempuh 530 cm. Dalam menjalankan misi B, diperoleh waktu tempuh rata-rata pada jarak 920 cm yakni 2 menit 53 detik. Kemudian saat menempuh misi C dengan jarak 1202 cm, diperoleh waktu 3 menit 81 detik. Seluruh pengujian navigasi robot dalam skenario perencanaan global berhasil mencapai goal sesuai trayektori yang dibangun dalam ROS. Adapun pada skenario navigasi lingkungan dinamis, robot menjalankan misi A dengan jarak 530 cm dengan waktu tempuh rata-rata 1 menit 48 detik. Misi B dengan jarak tempuh 920, diperoleh waktu tempuh rata-rata 3 menit 14 detik. Serta misi C dengan jarak tempuh 1202 cm, diperoleh waktu tempuh rata-rata 4 menit 12 detik. Sehingga pada pengujian sekanario ini dapat dinyatakan berhasil karena robot dapat mengenali objek dan merencanakan jalur trayektori terdekat dan teraman.

3. Dalam pengamatan sistem komunikasi antar *node* dalam *ROS Navigation Stack*, diketahui jumlah *node* yang aktif saat robot tidak bergerak yakni 4 *node*. Sedangkan pada saat robot mulai bergerak secara otonom, diketahui jumlah *node* yang aktif yakni 14 *node* pada 7 topik pendukung navigasi.

## 5.2 Saran

Adapun saran yang dapat penulis berikan terkait pengembangan penelitian ini kedepannya antara lain:

1. Dalam meningkatkan akurasi nilai kecepatan resultan robot maka sebaiknya perlu dilakukan perhitungan kombinasi nilai kecepatan linear dan anguler robot agar dalam penelitian berikutnya dapat diketahui fase dimana robot tepat melakukan manuver atau menjalankan navigasi berdasarkan jalur lokal.
2. Dalam penelitian ini, peneliti membatasi kecepatan roda robot akibat pengaruh terhadap odometri robot pada sistem ROS yang tidak akurat apabila pengaturan nilaqi kecepatan linear dan anguler ditambah. Oleh karena itu, kedepannya dapat dilakukan tuning pengujian kecepatan roda berdasarkan jenis aktuator yang digunakan sehingga pergerakan navigasi robot dapat lebih cepat namun tetap akurat.
3. Dalam mengamati pengaruh deviasi atau penyimpangan trayektori lintasan robot saat navigasi, sebaiknya perlu dilakukan pengamatan dengan menggunakan alat ukur kecepatan roda lalu dibandingkan dengan instruksi nilai pada sistem navigasi yang dibangun pada ROS serta penanda garis lintasan sepanjang jarak tempuh oleh robot.

## DAFTAR PUSTAKA

- Anggoro, B. (2013). Desain Pemodelan Kinematik dan Dinamik Humanoid Robot. Skripsi. Univesitas Diponegoro, Bandung.
- America, N. (2019). Executive Summary World Robotics 2019 Industrial Robots,” pp. 13–16.
- Brock, O and O. Khatib. (1999). High-speed navigation using the global dynamic window approach,” Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on, 1999, pp. 341-346 vol.1.
- Borenstein, J and Y. Koren, “Real-time obstacle avoidance for fast mobile robots in cluttered environments,” vol. 19, no. 5, pp. 572–577, 1990.
- Corke., P. (2017). Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised. Springer.
- Dieter, F., Wolfram, F., and Sebastian, T. (1997). The Dynamic WindowApproach to Collision Avoidance. pp. 137–146.
- Dudek, G. and Jenkin, M. (2010). Computational Principles of Mobile Robotics, 2nd ed. New York: Cambridge University Press.
- E. Dwiky, E., Endang, D. Rosalia, H., S. Sunarto, T. S, Kuat, G. Ferrianto. (2019). Sistem Navigasi Mobile Robot Dalam Ruangan Berbasis Autonomous Navigation. Journal of Mechanical Engineering and Mechatronics, pp 78-86.
- Gupta, K.V.N., Prasthech, K., Lakshmi, K.R.V., Kadam, S.S., Bailey, K. (2021). Autonomous Navigation in Dynamic Environment. International Research Journal of Engineering and Technology (IRJET). Vol. 08 (6). BMS College of Engineering, Karnataka, India.
- Ilham (2019). Navigasi Mobile Robot Darat Menggunakan Odometri Visual Berbasis Citra Stereo. Tesis. Institut Teknologi Sepuluh Nopember Surabaya
- Jianwei, Z., Shengyi, Jinyu., L. (2022). Research and Implementation of Autonomous Navigation for Mobile Robots Based on SLAM Algorithm under ROS.
- Joseph, Lentin. (2018). Robot Operating System for Absolute Beginners. Apress. India.

- Litman., T. (2019). Autonomous Vehicle Implementation Predictions: Implications for Transport Planning, Transp. Res. Board Annu. Meet., Vol. 42, no. January 2014, pp. 36–42, 2019.
- Minguez, J and L. Montano. (2000). Nearness Diagram Navigation (ND): A New Real Time Collision Avoidance Approach. In Proc. of the ieee/rsj international conference on intelligent robots and systems (iros'00, vol. 60, 2000, pp. 2094-- 2100.
- Muhammad, F., Wicaksana, C. A. (2021). Literature Review Sistem Navigasi Autonomous Mobile Robot Berbasis ROS (Robot Operating System). Jurnal Ilmiah Setrum. Vol.10 No.1 (2021) 103-112.
- Meng, Z. Sun, H., Qin, Z., Chen, C. Zhou, and M. H. (2018). Intelligent Robotic System for Autonomous Exploration and Active SLAM in Unknown Environments. SII 2017 - 2017 IEEE/SICE Int. Symp. Syst. Integr., Vol. 2018-Janua, pp. 651–656.
- M. Sorour, A. Cherubini, P. Fraisse and R. Passama. (2017). Motion Discontinuity-Robust Controller for Steerable Mobile Robots. IEEE Robotics and Automation Letters, vol. 2, pp. 452-459.
- Morales, L., Herrera, M., Camacho, O., Leica, P. and J. Aguilar. (2021). LAMDA Control Approaches Applied to Trajectory Tracking for Mobile Robots," IEEE Access, vol. 9, pp. 37179-37195.
- Rösmann, C., Feiten, W. Wösch., Hoffmann, F., and T. Bertram. (2012). "Trajectory Modification Considering Dynamic Constraints of Autonomous Robots," 7th Ger. Conf. Robot. Robot. 2012, pp. 74–79.
- Siegwart,R and Nourbakhsh., I., R. (2004). Introduction to Autonomous Mobile Robots. USA: Bradford Company.
- Sun, Y, Guan, L., Chang, Z. et al., (2019). Design of a low-cost indoor navigation system for food delivery robot based on multi-sensor information fusion," Sensors, vol. 19, no. 22, 4980 pages.
- Ulrich, I and J. Borenstein. (2000). VFH: local obstacle avoidance with look aheadverification, Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on, 2000.

- Wang, C. and Du, D. (2016). Research on Logistics Autonomous Mobile Robot System. IEEE Int. Conf. Mechatronics Autom. IEEE ICMA 2016, pp. 275–280.
- Wannacot, D., dkk.. (2012). Autonomous Navigation Planning with Ros. Journal.Michigan Technological University, USA.
- Y. Pyo, H. Cho, L. J. Jung, and D. Lim. (2017). ROS Robot Programming (English). ROBOTIS. [Online]. Available: <http://community.robotsource.org/t/download-the-ros-robot-programming-book-for-free/ 51>.

## LAMPIRAN

### Lampiran 1 Data Sampling Perhitungan Kecepatan Robot

a. Data sampling kecepatan robot pada skenario 1 Lingkungan Dinamis

<i>Time</i>	<i>Linear Speed (m/s)</i>	<i>Angular Speed (rad/s)</i>
1	0,219999999	-0,057435896
2	0,219999999	-0,073846154
3	0,219999999	-0,073846154
4	0,219999999	-0,073846154
5	0,219999999	-0,073846154
6	0,219999999	-0,057435896
7	0,219999999	-0,057435896
8	0,219999999	-0,057435896
9	0,219999999	-0,024615385
10	0,219999999	-0,008205128
11	0,219999999	-0,008205128
12	0,196842104	0,008205128
13	0,219999999	0,254358977
14	0,219999999	0,254358977
15	0,219999999	0,270769238
16	0,196842104	0,303589731
17	0,196842104	0,28717947
18	0,219999999	0,237948716
19	0,219999999	0,172307685
20	0,219999999	0,123076923
21	0,196842104	0,106666662
22	0,196842104	0,057435896
23	0,196842104	0
24	0,17368421	-0,073846154
25	0,17368421	-0,073846154
26	0,196842104	-0,139487177
27	0,17368421	-0,155897439
28	0,17368421	-0,172307685
29	0,196842104	-0,188717946
30	0,17368421	-0,139487177
31	0,17368421	-0,139487177
32	0,17368421	-0,139487177
33	0,17368421	-0,123076923
34	0,17368421	-0,106666662
35	0,17368421	-0,090256408
36	0,17368421	-0,090256408
37	0,17368421	-0,041025639
38	0,17368421	-0,041025639

39	0,17368421	0,024615385
40	0,17368421	0,041025639
41	0,17368421	0,024615385
42	0,17368421	0,024615385
43	0,17368421	0,008205128
44	0,17368421	0,008205128
45	0,17368421	-0,057435896
46	0,17368421	-0,057435896
47	0,17368421	-0,057435896
48	0,150526315	-0,041025639
49	0,150526315	-0,057435896
50	0,150526315	0,024615385
51	0,150526315	0,008205128
52	0,150526315	0,008205128
53	0,150526315	-0,090256408
54	0,150526315	-0,106666662
55	0,150526315	-0,106666662
56	0,150526315	-0,106666662
57	0,150526315	0
58	0,150526315	0
59	0,150526315	0,008205128
60	0,150526315	0,024615385
61	0,150526315	0,024615385
62	0,12736842	0,041025639
63	0,150526315	-0,057435896
64	0,150526315	-0,008205128
65	0,150526315	-0,008205128
66	0,150526315	-0,008205128
67	0,150526315	-0,057435896
68	0,150526315	-0,008205128
69	0,150526315	-0,008205128
70	0,12736842	-0,057435896
71	0,12736842	0,024615385
72	0,12736842	0,024615385
73	0,12736842	0,024615385
74	0,12736842	0,024615385
75	0,12736842	0,024615385
76	0,12736842	-0,090256408
77	0,12736842	-0,106666662
78	0,12736842	-0,106666662
79	0,12736842	-0,106666662
80	0,12736842	-0,106666662
81	0,12736842	0
82	0,12736842	0,008205128

83	0,12736842	0,008205128
84	0,12736842	0,024615385
85	0,12736842	0,024615385
86	0,12736842	0
87	0,12736842	-0,106666662
88	0,12736842	-0,008205128
89	0,12736842	-0,008205128
90	0,12736842	-0,008205128
91	0,12736842	-0,008205128
92	0,12736842	-0,106666662
93	0,12736842	-0,106666662
94	0,12736842	-0,123076923
95	0,12736842	-0,008205128
96	0,12736842	-0,008205128
97	0,12736842	0
98	0,12736842	0,008205128
99	0,12736842	0
100	0,12736842	0

## Lampiran 2 Dokumentasi Pelaksanaan Penelitian

### a. Tahap Persiapan



Riset & studi literatur



Diskusi dan konsultasi



Rancang Bangun Robot

**b. Tahap Perancangan**

Perancangan Awal: Redesign Kursi Roda Menjadi *Waiter-Bot*



Instalasi perangkat keras *Waiter-Bot*

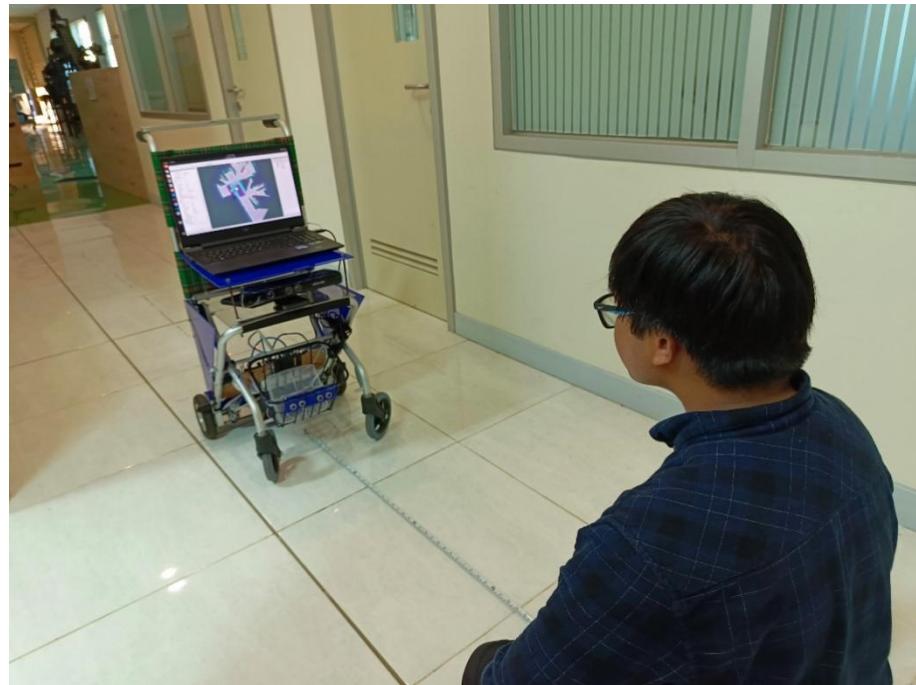
### c. Tahap Pengujian



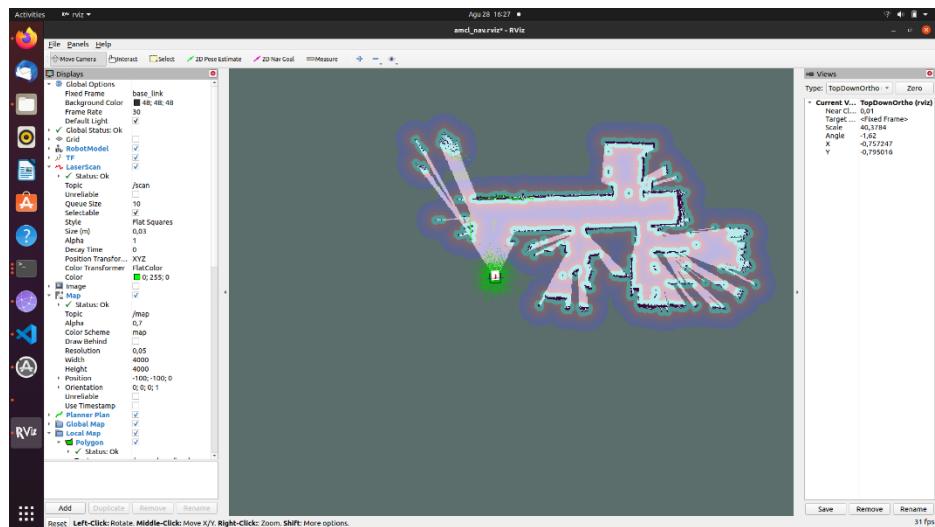
Pengujian Robot mengenali objek penghalang



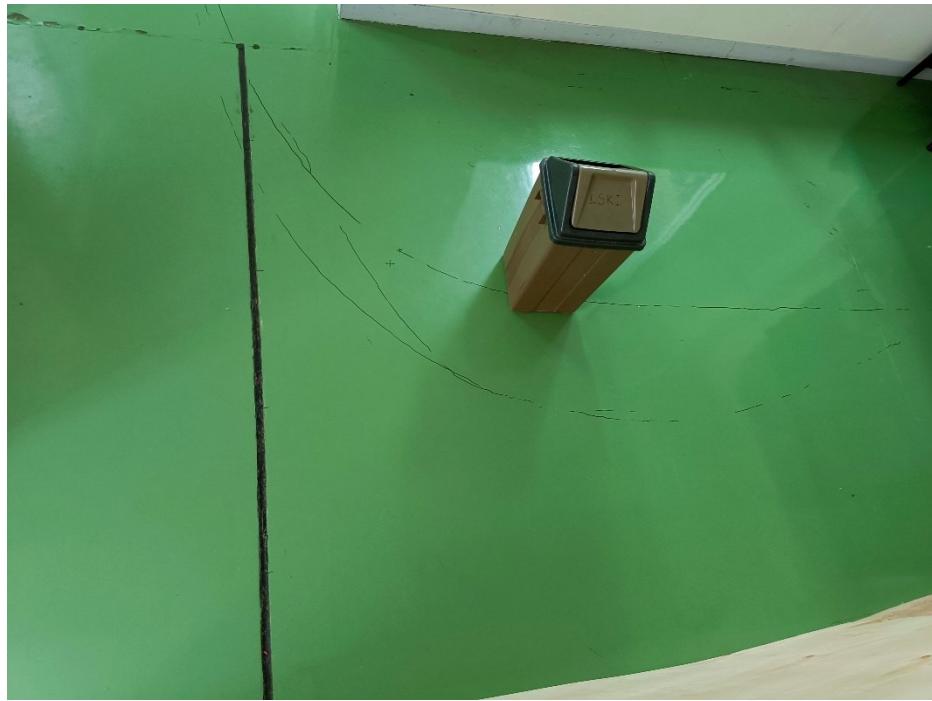
Pengujian robot melakukan pivot dalam menghindari *obstacle*



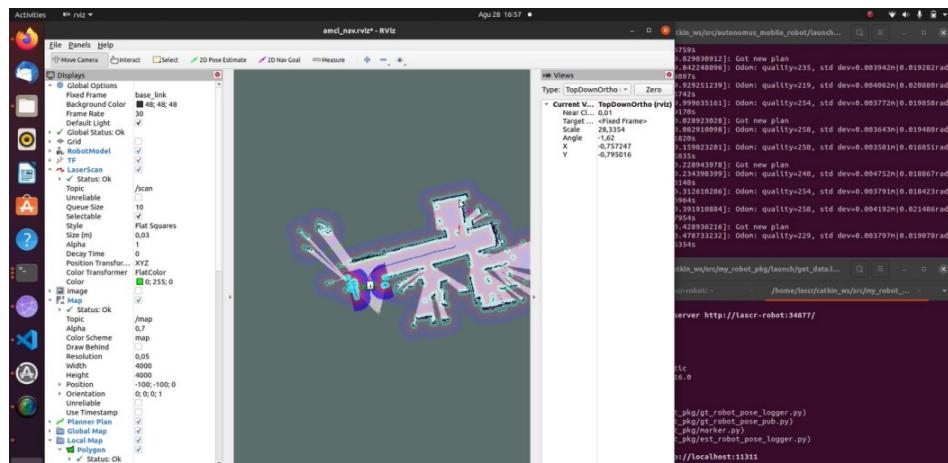
Pengukuran Jarak antara robot dan objek penghalang



Tampilan peta pengujian *Waiter-Bot* dalam RViz



Jejak Waiter-Bot dalam pengamatan trayektori perencanaan jalur



Waiter-Bot Sedang dalam melakukan navigasi menuju misi



Video Pengujian Sistem Navigasi *Waiter-Bot*

### Lampiran 3 Kode program *Waiter-Bot*

#### Program Arduino (*Low Level*)

```
// Variabel untuk menyimpan data dari Python
int pwm_left = 0;
int pwm_right = 0;
int direction_left = 0;
int direction_right = 0;

// Pin untuk mengontrol driver motor
#include <Servo.h> // You need to include Servo.h as it is used
by the HB-25 Library
#include <HB25MotorControl.h>

const byte controlPinL = 6;
const byte controlPinR = 7;
HB25MotorControl motorControlL(controlPinL);
HB25MotorControl motorControlR(controlPinR);

void setup() {
    motorControlL.begin();
    motorControlR.begin();

    // Inisialisasi Serial
    Serial.begin(57600);
}

void loop() {
    // Jika ada data yang diterima dari Python
    if (Serial.available()) {

        // Membaca data dari Serial
        String input = Serial.readStringUntil('\n');
        input.trim();

        // Memisahkan data menjadi PWM kiri, arah kiri, PWM kanan, dan
        arah kanan
        int separator1 = input.indexOf(',');
        int separator2 = input.indexOf(',', separator1 + 1);
```

```

        int separator3 = input.indexOf(',', separator2 + 1);

        pwm_left = input.substring(0, separator1).toInt();
        direction_left = input.substring(separator1 + 1,
                                         separator2).toInt();
        pwm_right = input.substring(separator2 + 1,
                                    separator3).toInt();
        direction_right = input.substring(separator3 + 1).toInt();

        // Mengontrol motor
        motorControlL.moveAtSpeed(pwm_left);
        motorControlR.moveAtSpeed(pwm_right);
    }
}

```

### **Ros\_serial\_encoder**

```

/*
 * Author: Automatic Addison
 * Website: https://automaticaddison.com
 * Description: ROS node that publishes the accumulated ticks for
each wheel
 * (right_ticks and left_ticks topics) using the built-in encoder
 * (forward = positive; reverse = negative)
 */

#include <ros.h>
#include <std_msgs/Int16.h>

// Handles startup and shutdown of ROS
ros::NodeHandle nh;

// Encoder output to Arduino Interrupt pin. Tracks the tick count.
#define ENC_IN_LEFT_A 2
#define ENC_IN_RIGHT_A 19

// Other encoder output to Arduino to keep track of wheel direction
// Tracks the direction of rotation.

```

```
#define ENC_IN_LEFT_B 3
#define ENC_IN_RIGHT_B 18

// True = Forward; False = Reverse
boolean Direction_left = true;
boolean Direction_right = true;

// Minimum and maximum values for 16-bit integers
const int encoder_minimum = -32768;
const int encoder_maximum = 32767;

// Keep track of the number of wheel ticks
std_msgs::Int16 right_wheel_tick_count;
ros::Publisher rightPub("right_ticks", &right_wheel_tick_count);

std_msgs::Int16 left_wheel_tick_count;
ros::Publisher leftPub("left_ticks", &left_wheel_tick_count);

// 100ms interval for measurements
const int interval = 100;
long previousMillis = 0;
long currentMillis = 0;

// Increment the number of ticks
void right_wheel_tick() {

    // Read the value for the encoder for the right wheel
    int val = digitalRead(ENC_IN_RIGHT_B);

    if(val == LOW) {
        Direction_right = false; // Reverse
    }
    else {
        Direction_right = true; // Forward
    }

    if (Direction_right) {

        if (right_wheel_tick_count.data == encoder_maximum) {
```

```
    right_wheel_tick_count.data = encoder_minimum;
}
else {
    right_wheel_tick_count.data++;
}
}

else {
    if (right_wheel_tick_count.data == encoder_minimum) {
        right_wheel_tick_count.data = encoder_maximum;
    }
    else {
        right_wheel_tick_count.data--;
    }
}

// Increment the number of ticks
void left_wheel_tick() {

    // Read the value for the encoder for the left wheel
    int val = digitalRead(ENC_IN_LEFT_B);

    if(val == LOW) {
        Direction_left = true; // Reverse
    }
    else {
        Direction_left = false; // Forward
    }

    if (Direction_left) {
        if (left_wheel_tick_count.data == encoder_maximum) {
            left_wheel_tick_count.data = encoder_minimum;
        }
        else {
            left_wheel_tick_count.data++;
        }
    }
    else {
        if (left_wheel_tick_count.data == encoder_minimum) {
```

```
    left_wheel_tick_count.data = encoder_maximum;
}
else {
    left_wheel_tick_count.data--;
}
}

void setup() {

// Set pin states of the encoder
pinMode(ENC_IN_LEFT_A , INPUT_PULLUP);
pinMode(ENC_IN_LEFT_B , INPUT);
pinMode(ENC_IN_RIGHT_A , INPUT_PULLUP);
pinMode(ENC_IN_RIGHT_B , INPUT);

// Every time the pin goes high, this is a tick
attachInterrupt(digitalPinToInterrupt(ENC_IN_LEFT_A),
left_wheel_tick, RISING);
attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGHT_A),
right_wheel_tick, RISING);

// ROS Setup
nh.getHardware()->setBaud(115200);
nh.initNode();
nh.advertise(rightPub);
nh.advertise(leftPub);
}

void loop() {

// Record the time
currentMillis = millis();

// If 100ms have passed, print the number of ticks
if (currentMillis - previousMillis > interval) {

previousMillis = currentMillis;
```

```

    rightPub.publish( &right_wheel_tick_count );
    leftPub.publish( &left_wheel_tick_count );
    nh.spinOnce();
}
}

```

### **ROS NODE: *serial\_encoder\_imu***

```

/*
 * Author: Automatic Addison
 * Website: https://automaticaddison.com
 * Description: ROS node that publishes the accumulated ticks for
each wheel
 * (right_ticks and left_ticks topics) using the built-in encoder
 * (forward = positive; reverse = negative)
 */

#include "MPU9250.h"
#include <ros.h>
#include <std_msgs/Int16.h>
#include <sensor_msgs/Imu.h>

// Handles startup and shutdown of ROS
ros::NodeHandle nh;

// Encoder output to Arduino Interrupt pin. Tracks the tick count.
#define ENC_IN_LEFT_A 2
#define ENC_IN_RIGHT_A 19

// Other encoder output to Arduino to keep track of wheel direction
// Tracks the direction of rotation.
#define ENC_IN_LEFT_B 3
#define ENC_IN_RIGHT_B 18

// True = Forward; False = Reverse
boolean Direction_left = true;
boolean Direction_right = true;

// Minimum and maximum values for 16-bit integers
const int encoder_minimum = -32768;

```

```
const int encoder_maximum = 32767;

// Keep track of the number of wheel ticks
std_msgs::Int16 right_wheel_tick_count;
ros::Publisher rightPub("right_ticks", &right_wheel_tick_count);

std_msgs::Int16 left_wheel_tick_count;
ros::Publisher leftPub("left_ticks", &left_wheel_tick_count);

// Keep track of mpu9250 imu sensor
sensor_msgs::Imu imu_msg;
ros::Publisher imu_pub("imu_data", &imu_msg);
MPU9250 imu;

// 100ms interval for measurements
const int interval = 100;
long previousMillis = 0;
long currentMillis = 0;

//imu sensor reading
void print_roll_pitch_yaw() {
    // Baca yaw, pitch, dan roll dari MPU9250
    float yaw = imu.getYaw();          // Mendapatkan nilai yaw menggunakan
    fungsi getYaw()
    float pitch = imu.getPitch();      // Mendapatkan nilai pitch
    menggunakan fungsi getPitch()
    float roll = imu.getRoll();        // Mendapatkan nilai roll
    menggunakan fungsi getRoll()

    // Baca kecepatan sudut (angular velocity) dari MPU9250
    float angular_velocity_x = imu.getGyroX(); // Mendapatkan nilai
    kecepatan sudut pada sumbu x menggunakan fungsi getGyroX()
    float angular_velocity_y = imu.getGyroY(); // Mendapatkan nilai
    kecepatan sudut pada sumbu y menggunakan fungsi getGyroY()
    float angular_velocity_z = imu.getGyroZ(); // Mendapatkan nilai
    kecepatan sudut pada sumbu z menggunakan fungsi getGyroZ()

    // Baca percepatan linier (linear acceleration) dari MPU9250
```

```
float linear_acceleration_x = imu.getLinearAccX(); //  
Mendapatkan nilai percepatan linier pada sumbu x menggunakan fungsi  
getLinearAccX()  
  
float linear_acceleration_y = imu.getLinearAccY(); //  
Mendapatkan nilai percepatan linier pada sumbu y menggunakan fungsi  
getLinearAccY()  
  
float linear_acceleration_z = imu.getLinearAccZ(); //  
Mendapatkan nilai percepatan linier pada sumbu z menggunakan fungsi  
getLinearAccZ()  
  
// Isi data IMU  
imu_msg.header.stamp = nh.now();  
imu_msg.header.frame_id = "imu_link";  
// Konversi Euler angles menjadi quaternion  
float cy = cos(yaw * 0.5);  
float sy = sin(yaw * 0.5);  
float cp = cos(pitch * 0.5);  
float sp = sin(pitch * 0.5);  
float cr = cos(roll * 0.5);  
float sr = sin(roll * 0.5);  
  
float qw = cy * cp * cr + sy * sp * sr;  
float qx = cy * cp * sr - sy * sp * cr;  
float qy = sy * cp * sr + cy * sp * cr;  
float qz = sy * cp * cr - cy * sp * sr;  
  
// Isi data quaternion ke pesan IMU  
imu_msg.orientation.x = qx;  
imu_msg.orientation.y = qy;  
imu_msg.orientation.z = qz;  
imu_msg.orientation.w = qw;  
  
imu_msg.angular_velocity.x = angular_velocity_x;  
imu_msg.angular_velocity.y = angular_velocity_y;  
imu_msg.angular_velocity.z = angular_velocity_z;  
  
imu_msg.linear_acceleration.x = linear_acceleration_x;  
imu_msg.linear_acceleration.y = linear_acceleration_y;  
imu_msg.linear_acceleration.z = linear_acceleration_z;
```

```
// Serial.print("Yaw, Pitch, Roll: ");
// Serial.print(yaw, 2);
// Serial.print(", ");
// Serial.print(pitch, 2);
// Serial.print(", ");
// Serial.println(roll, 2);
}

// Increment the number of ticks
void right_wheel_tick() {

    // Read the value for the encoder for the right wheel
    int val = digitalRead(ENC_IN_RIGHT_B);

    if (val == LOW) {
        Direction_right = false; // Reverse
    } else {
        Direction_right = true; // Forward
    }

    if (Direction_right) {

        if (right_wheel_tick_count.data == encoder_maximum) {
            right_wheel_tick_count.data = encoder_minimum;
        } else {
            right_wheel_tick_count.data++;
        }
    } else {
        if (right_wheel_tick_count.data == encoder_minimum) {
            right_wheel_tick_count.data = encoder_maximum;
        } else {
            right_wheel_tick_count.data--;
        }
    }
}

// Increment the number of ticks
void left_wheel_tick() {
```

```
// Read the value for the encoder for the left wheel
int val = digitalRead(ENC_IN_LEFT_B);

if (val == LOW) {
    Direction_left = true; // Reverse
} else {
    Direction_left = false; // Forward
}

if (Direction_left) {
    if (left_wheel_tick_count.data == encoder_maximum) {
        left_wheel_tick_count.data = encoder_minimum;
    } else {
        left_wheel_tick_count.data++;
    }
} else {
    if (left_wheel_tick_count.data == encoder_minimum) {
        left_wheel_tick_count.data = encoder_maximum;
    } else {
        left_wheel_tick_count.data--;
    }
}

void setup() {
    //set i2c connection from mpu9250
    Wire.begin();
    delay(2000);
    if (!imu.setup(0x68)) { // change to your own address
        while (1) {
            Serial.println("MPU connection failed. Please check your
connection with `connection_check` example.");
            delay(5000);
        }
    }

    // Set pin states of the encoder
    pinMode(ENC_IN_LEFT_A, INPUT_PULLUP);
    pinMode(ENC_IN_LEFT_B, INPUT);
```

```

pinMode(ENC_IN_RIGHT_A, INPUT_PULLUP);
pinMode(ENC_IN_RIGHT_B, INPUT);

// Every time the pin goes high, this is a tick
attachInterrupt(digitalPinToInterrupt(ENC_IN_LEFT_A),
left_wheel_tick, RISING);
attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGHT_A),
right_wheel_tick, RISING);

// ROS Setup
nh.getHardware()->setBaud(115200);
nh.initNode();
nh.advertise(rightPub);
nh.advertise(leftPub);
nh.advertise(imu_pub);
}

void loop() {
//update mpu value if there any change
if (imu.update()) {
    static uint32_t prev_ms = millis();
    if (millis() > prev_ms + 25) {
        print_roll_pitch_yaw();
        prev_ms = millis();
    }
}
// Record the time
currentMillis = millis();

// If 100ms have passed, print the number of ticks
if (currentMillis - previousMillis > interval) {

    previousMillis = currentMillis;

    rightPub.publish(&right_wheel_tick_count);
    leftPub.publish(&left_wheel_tick_count);
    imu_pub.publish(&imu_msg);
    nh.spinOnce();
}
}

```

**ROS NODE: *serial\_imu***

```
#include "MPU9250.h"
#include <ros.h>
#include <sensor_msgs/Imu.h>
ros::NodeHandle nh;
sensor_msgs::Imu imu_msg;
ros::Publisher imu_pub("imu_data", &imu_msg);

MPU9250 mpu;

void setup() {
    nh.initNode();
    nh.advertise(imu_pub);
    Serial.begin(115200);
    Wire.begin();
    delay(2000);

    if (!mpu.setup(0x68)) { // change to your own address
        while (1) {
            Serial.println("MPU connection failed. Please check your
connection with `connection_check` example.");
            delay(5000);
        }
    }
}

void loop() {
    if (mpu.update()) {
        static uint32_t prev_ms = millis();
        if (millis() > prev_ms + 25) {
            print_roll_pitch_yaw();
            imu_pub.publish(&imu_msg);
            nh.spinOnce();
            prev_ms = millis();
        }
    }
}

void print_roll_pitch_yaw() {
```

```

float yaw = mpu.getYaw();
float pitch = mpu.getPitch();
float roll = mpu.getRoll();
imu_msg.header.stamp = nh.now();
imu_msg.orientation.x = roll;
imu_msg.orientation.y = pitch;
imu_msg.orientation.z = yaw;
Serial.print("Yaw, Pitch, Roll: ");
Serial.print(yaw, 2);
Serial.print(", ");
Serial.print(pitch, 2);
Serial.print(", ");
Serial.println(roll, 2);
}

```

### **Kode Program Navigasi Dalam ROS (*High Level*)**

#### **Konfigurasi Node Perangkat Keras Robot**

```
#!/usr/bin/env python3
```

```

import rospy
from sensor_msgs.msg import JointState
from std_msgs.msg import Float64
import serial

class ROBOTHardwareInterface:
    def __init__(self):
        self.joint_names          = ['left_wheel_joint',
        'right_wheel_joint']
        self.joint_positions      = [0.0, 0.0]
        self.joint_velocities     = [0.0, 0.0]
        self.joint_efforts        = [0.0, 0.0]
        self.joint_velocity_commands = [0.0, 0.0]
        self.left_motor_pos       = 0
        self.right_motor_pos      = 0
        self.left_prev_cmd        = 0
        self.right_prev_cmd       = 0
        self.serial    = serial.Serial('/dev/ttyUSB0', 9600,
        timeout=0.1)

```

```

def init(self):
    rospy.init_node('robot_hardware_interface')

        self.joint_state_pub = rospy.Publisher('joint_states',
JointState, queue_size=10)
        self.left_velocity_sub =
rospy.Subscriber('left_wheel_joint_velocity_controller/command',
Float64, self.left_velocity_callback)
        self.right_velocity_sub =
rospy.Subscriber('right_wheel_joint_velocity_controller/command',
Float64, self.right_velocity_callback)

def left_velocity_callback(self, msg):
    self.joint_velocity_commands[0] = msg.data

def right_velocity_callback(self, msg):
    self.joint_velocity_commands[1] = msg.data

def read(self):
    data = self.serial.readline().decode('utf-8').strip()
    if data:
        left_encoder_value, right_encoder_value = map(int,
data.split(','))
        self.joint_position[0] = left_encoder_value
        self.joint_position[1] = right_encoder_value
    else:
        print("Invalid data received from Arduino")

def write(self):
    left_motor_velocity = int(self.joint_velocity_commands[0])
    right_motor_velocity =
int(self.joint_velocity_commands[1])

    if self.left_prev_cmd != left_motor_velocity or
self.right_prev_cmd != right_motor_velocity:
        command =
f'{left_motor_velocity},{right_motor_velocity}\n'
        self.serial.write(command.encode())

```

```

        self.left_prev_cmd = left_motor_velocity
        self.right_prev_cmd = right_motor_velocity

    def update(self):
        joint_state = JointState()
        joint_state.header.stamp = rospy.Time.now()
        joint_state.name = self.joint_names
        joint_state.position = self.joint_positions
        joint_state.velocity = self.joint_velocities
        joint_state.effort = self.joint_efforts

        self.joint_state_pub.publish(joint_state)

        self.read()
        self.write()

    def run(self):
        rate = rospy.Rate(10) # 10 Hz

        while not rospy.is_shutdown():
            self.update()
            rate.sleep()

    if __name__ == '__main__':
        robot_hw = ROBOTHardwareInterface()
        robot_hw.init()
        robot_hw.run()

```

### **Move Base Package**

```

<launch>

    <!-- Arguments -->
    <arg name="cmd_vel_topic" default="/cmd_vel" />
    <arg name="odom_topic" default="odom" />
    <arg name="move_forward_only" default="false"/>

    <!-- move_base -->
    <node      pkg="move_base"      type="move_base"      respawn="false"
      name="move_base" output="screen">

```

```

<param name="base_local_planner"
       value="dwa_local_planner/DWAPlannerROS" />
<rosparam file="$(find
autonomus_mobile_robot)/param/costmap_common_params.yaml"
command="load" ns="global_costmap" />
<rosparam file="$(find
autonomus_mobile_robot)/param/costmap_common_params.yaml"
command="load" ns="local_costmap" />
<rosparam file="$(find
autonomus_mobile_robot)/param/local_costmap_params.yaml"
command="load" />
<rosparam file="$(find
autonomus_mobile_robot)/param/global_costmap_params.yaml"
command="load" />
<rosparam file="$(find
autonomus_mobile_robot)/param/move_base_params.yaml"
command="load" />
<rosparam file="$(find
autonomus_mobile_robot)/param/dwa_local_planner_params.yaml"
command="load" />
<remap from="cmd_vel" to="$(arg cmd_vel_topic)"/>
<remap from="odom" to="$(arg odom_topic)"/>
<param name="DWAPlannerROS/min_vel_x" value="0.0" if="$(arg
move_forward_only)" />
</node>
</launch>
```

### **Global Costmap**

```

global_frame: map
rolling_window: false
track_unknown_space: true

plugins:
  - {name: static, type: "costmap_2d::StaticLayer"}
  - {name: inflation_g, type: "costmap_2d::InflationLayer"}
```

### ***Local Costmap***

```
global_frame: odom
rolling_window: true
width: 1.5
height: 1.5

plugins:
  - {name: obstacles_laser,
    "costmap_2d::ObstacleLayer"}
  - {name: inflation_l1,
    "costmap_2d::InflationLayer"}
```

### ***Algoritma DWA Local Planner***

```
# Robot Configuration Parameters
max_vel_x: 0.4
min_vel_x: -0.4

max_vel_y: 0.0
min_vel_y: 0.0

# The velocity when robot is moving in a straight line
max_vel_trans: 0.4
min_vel_trans: 0.3

max_vel_theta: 0.4
min_vel_theta: 0.3

acc_lim_x: 2.5
acc_lim_y: 0.0
acc_lim_theta: 3.2

# Goal Tolerance Parameters
xy_goal_tolerance: 0.05
yaw_goal_tolerance: 0.17
latch_xy_goal_tolerance: false

# Forward Simulation Parameters
sim_time: 2.0
vx_samples: 20
```

```

vy_samples: 0
vth_samples: 40
controller_frequency: 10.0

# Trajectory Scoring Parameters
path_distance_bias: 32.0
goal_distance_bias: 20.0
occdist_scale: 0.02
forward_point_distance: 0.325
stop_time_buffer: 0.2
scaling_speed: 0.25
max_scaling_factor: 0.2

# Oscillation Prevention Parameters
oscillation_reset_dist: 0.05

# Debugging
publish_traj_pc : true
publish_cost_grid_pc: true

```

### **Kode Program ROS: *base\_local\_planner\_params.yaml***

TrajectoryPlannerROS:

```

# Robot Configuration Parameters
max_vel_x: 0.18
min_vel_x: 0.08

max_vel_theta: 1.0
min_vel_theta: -1.0
min_in_place_vel_theta: 1.0

acc_lim_x: 1.0
acc_lim_y: 0.0
acc_lim_theta: 0.6

# Goal Tolerance Parameters
xy_goal_tolerance: 0.10
yaw_goal_tolerance: 0.05

```

```
# Differential-drive robot configuration
holonomic_robot: false

# Forward Simulation Parameters
sim_time: 0.8
vx_samples: 18
vtheta_samples: 20
sim_granularity: 0.05
```