



TUGAS AKHIR - EC184801

**PENGEMBANGAN LINGKUNGAN SIMULASI
UNTUK PENGUJIAN *SOCIALLY ASSISTIVE
ROBOTS* MENGGUNAKAN ROS 2 DAN GAZEBO**

**Muhammad Alfi Maulana Fikri
NRP 0721 17 4000 0009**

**Dosen Pembimbing
Prof. Dr. Ir. Mauridhi Hery Purnomo, M.Eng.
Dr. I Ketut Eddy Purnama, S.T., M.T.**

**DEPARTEMEN TEKNIK KOMPUTER
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2021**



TUGAS AKHIR - EC184801

PENGEMBANGAN LINGKUNGAN SIMULASI UNTUK PENGUJIAN *SOCIALLY ASSISTIVE ROBOTS* MENGGUNAKAN ROS 2 DAN GAZEBO

**Muhammad Alfi Maulana Fikri
NRP 0721 17 4000 0009**

**Dosen Pembimbing
Prof. Dr. Ir. Mauridhi Hery Purnomo, M.Eng.
Dr. I Ketut Eddy Purnama, S.T., M.T.**

**DEPARTEMEN TEKNIK KOMPUTER
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2021**

[Halaman ini sengaja dikosongkan]



FINAL PROJECT - EC184801

**DEVELOPMENT OF SIMULATION
ENVIRONMENT FOR SOCIALLY ASSISTIVE
ROBOTS TESTING USING ROS 2 AND GAZEBO**

**Muhammad Alfi Maulana Fikri
NRP 0721 17 4000 0009**

Advisors

**Prof. Dr. Ir. Mauridhi Hery Purnomo, M.Eng.
Dr. I Ketut Eddy Purnama, S.T., M.T.**

**DEPARTMENT OF COMPUTER ENGINEERING
Faculty of Intelligent Electrical and Information Technology
Sepuluh Nopember Institute of Technology
Surabaya 2021**

[Halaman ini sengaja dikosongkan]

PERNYATAAN KEASLIAN TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi buku Tugas Akhir dengan judul "**Pengembangan Lingkungan Simulasi untuk Pengujian *Socially Assistive Robots* Menggunakan ROS 2 dan Gazebo**" adalah benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka. Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, Juli 2021

Muhammad Alfi Maulana Fikri
0721 17 4000 0009

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

PENGEMBANGAN LINGKUNGAN SIMULASI UNTUK PENGUJIAN *SOCIALLY ASSISTIVE ROBOTS* MENGGUNAKAN ROS 2 DAN GAZEBO

Tugas Akhir ini disusun untuk memenuhi salah satu syarat memperoleh gelar Sarjana Teknik di Institut Teknologi Sepuluh Nopember Surabaya

Oleh: Muhammad Alfi Maulana Fikri (NRP. 0721 17 4000 0009)

Tanggal Ujian : 12 Juli 2021

Periode Wisuda : September 2021

Disetujui Oleh:

Prof. Dr. Ir. Mauridhi Hery Purnomo, M.Eng. (Pembimbing I)
NIP: 9580916 198601 1 001

Dr. I Ketut Eddy Purnama, S.T., M.T. (Pembimbing II)
NIP: 9690730 199512 1 001

..... (Penguji I)
NIP:
.....

..... (Penguji II)
NIP:
.....

..... (Penguji III)
NIP:
.....

Mengetahui,
Kepala Departemen Teknik Komputer

Dr. Supeno Mardi Susiki Nugroho, S.T., M.T.
NIP. 9700313 199512 1 001

[Halaman ini sengaja dikosongkan]

ABSTRAK

Nama	:	Muhammad Alfi Maulana Fikri
Judul	:	Pengembangan Lingkungan Simulasi untuk Pengujian <i>Socially Assistive Robots</i> Menggunakan ROS 2 dan Gazebo
Pembimbing	:	1. Prof. Dr. Ir. Mauridhi Hery Purnomo, M.Eng. 2. Dr. I Ketut Eddy Purnama, S.T., M.T.

Selama beberapa tahun terakhir, robot telah mengalami perkembangan yang cukup signifikan. Salah satu bentuk perkembangan tersebut adalah *socially assistive robots* (SARs) yang mampu memberikan bantuan kepada pengguna dalam bentuk interaksi sosial. Namun, karena sifatnya yang melibatkan interaksi langsung dengan pengguna, pengujian pada SARs akan menjadi sulit dan beresiko. Untuk itu, pada penelitian ini kami mengajukan lingkungan simulasi untuk pengujian SARs yang dibuat menggunakan simulator Gazebo. Di dalam lingkungan simulasi ini, model robot akan diujikan dengan model pengguna serta model-model objek lain secara virtual. Agar pengujian yang dilakukan di simulasi bisa diterapkan pada *real robot*, sistem kontroler yang ada pada robot akan dibuat secara terabstraksi dengan memisah setiap komponen menjadi *nodes* menggunakan ROS 2. Hasilnya, sistem yang dibuat mampu menghasilkan tindakan yang sama dalam menggerakkan model robot dan *real robot* dengan perbedaan *error* sebesar 2.6% di simulasi dan 12.5% di dunia nyata. Dengan performa yang dimiliki ROS 2, pengiriman citra dengan resolusi hingga 640 x 480 mampu menghasilkan *delay* di bawah 50 ms dan frekuensi di atas 90% pada sesama perangkat maupun antar-perangkat. Pada simulasi, model pengguna terbukti mampu digunakan untuk mensimulasikan pengguna melalui hasil percobaan deteksi pose, sedangkan lingkungan simulasi terbukti mampu digunakan untuk mensimulasikan ruangan melalui hasil percobaan SLAM.

Kata Kunci: Simulasi, *Assistive Robotics*, ROS2, Gazebo.

[Halaman ini sengaja dikosongkan]

ABSTRACT

Name : Muhammad Alfi Maulana Fikri
Title : Development of Simulation Environment for Socially Assistive Robots Testing Using ROS 2 and Gazebo
Advisors : 1. Prof. Dr. Ir. Mauridhi Hery Purnomo, M.Eng.
2. Dr. I Ketut Eddy Purnama, S.T., M.T.

Over the past few years, robots have undergone significant developments. One of this development is socially assistive robots (SARs) which are able to assist users in the form of social interaction. However, due to their nature which involves direct interaction with the user, testing SARs could be difficult and risky. For this reason, in this study we propose a simulation environment for testing SARs created using the Gazebo simulator. In this simulation environment, the robot model will be tested virtually with a user model and other object models. In order for the test performed in the simulation could be applied to real robots, the controller system in the robot will be abstracted by separating each component into nodes using ROS 2. As a result, the system created can produce the same action in moving the robot model and the real robot with an error difference of 2.6% in the simulation and 12.5% in the real world. With the performance of ROS 2, images delivery with a resolution of up to 640 x 480 can produce delays below 50 ms and frequencies above 90% on the same device and between devices. In the simulation, the user model proved capable of being used to simulate the user through the results of the pose detection experiment, while the simulation environment proved capable of being used to simulate the room through the results of the SLAM experiment.

Keywords: *Simulation, Assistive Robotics, ROS 2, Gazebo.*

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Puji dan syukur kehadirat Allah SWT atas segala limpahan berkah, rahmat, serta hidayah-Nya, penulis dapat menyelesaikan penelitian ini dengan judul **“Pengembangan Lingkungan Simulasi untuk Pengujian *Socially Assistive Robots* Menggunakan ROS 2 dan Gazebo”**. Penelitian ini disusun dalam rangka pemenuhan bidang riset di Departemen Teknik Komputer, serta digunakan sebagai persyaratan menyelesaikan pendidikan S1.

Dalam penyusunan buku ini, penulis mengucapkan terima kasih kepada Keluarga yang telah memberikan dorongan spiritual dan material dalam penyelesaian penelitian ini. Terutama kepada Ayah atas didikannya kepada penulis selama ini, semoga beliau husnul khatimah di sana, aamiin.

Penulis juga mengucapkan terima kasih kepada Bapak Prof. Dr. Ir. Mauridhi Hery Purnomo, M.Eng., Bapak Dr. I Ketut Eddy Purnama, S.T., M.T., dan Bapak Muhtadin ST., MT. atas arahan dan bimbingan selama pengerjaan penelitian tugas akhir ini. Serta kepada Bapak-ibu dosen pengajar Departemen Teknik Komputer atas pengajaran dan perhatian yang diberikan kepada penulis selama ini.

Dan terakhir, terima kasih kepada rekan-rekan ICHIRO ITS, Robotika ITS, dan B201 crew atas pengalamannya kepada penulis. Serta kepada rekan-rekan seperjuangan Teknik Komputer 2017, E57, dan penghuni rumah anak TK.

Kesempurnaan hanya milik Allah SWT, untuk itu penulis memohon segenap kritik dan saran yang membangun. Semoga penelitian ini dapat memberikan manfaat bagi kita semua, aamiin.

Surabaya, Juli 2021

Muhammad Alfi Maulana Fikri

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

ABSTRAK	i
ABSTRACT	iii
KATA PENGANTAR	v
DAFTAR ISI	x
DAFTAR GAMBAR	xiii
DAFTAR KODE	xv
DAFTAR TABEL	xvii
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Permasalahan	2
1.3 Penelitian Terkait	2
1.4 Tujuan	3
1.5 Batasan Masalah	3
1.6 Sistematika Penulisan	3
2 TINJAUAN PUSTAKA	5
2.1 <i>Assistive Robotics</i>	5
2.1.1 <i>Socially Assistive Robots (SARs)</i>	6
2.2 Robot Operating System 2 (ROS 2)	7
2.2.1 <i>ROS 2 Node</i>	8

2.2.2	<i>ROS 2 Middleware Implementations</i>	11
2.2.3	<i>ROS 2 Client Libraries</i>	13
2.2.4	<i>ROS 2 Interface</i>	13
2.2.5	<i>ROS 2 Command-line Interface</i>	15
2.2.6	RQt	16
2.3	Gazebo	17
2.3.1	<i>Gazebo Model</i>	18
2.3.2	<i>Gazebo Plugin</i>	19
2.4	<i>Holonomic Robot</i>	20
2.5	<i>Inertial Measurement Unit (IMU)</i>	21
2.6	<i>Smart Assistive Posture Device</i>	21
2.7	<i>Depth Camera</i>	22
2.7.1	Kinect V2	23
2.7.2	<i>Freenect 2 Library (libfreenect2)</i>	24
2.8	MediaPipe	24
2.8.1	MediaPipe Pose	25
2.9	SLAM	26
2.9.1	RTAB-Map	26
3	DESAIN DAN IMPLEMENTASI	27
3.1	Pengembangan Model Robot	28
3.1.1	Desain Robot <i>Dienen</i>	28
3.1.2	Prototipe Robot <i>Dienen</i>	30
3.1.3	Struktur SDFormat Robot	31
3.2	Pengembangan Model Pengguna	34
3.2.1	Struktur SDFormat Pengguna	35
3.3	Pengembangan Lingkungan Simulasi	36
3.3.1	Lingkungan Simulasi <i>Outdoor</i>	36

3.3.2	Lingkungan Simulasi <i>Indoor</i>	37
3.4	Integrasi <i>Plugin</i> untuk Simulasi	39
3.4.1	<i>Navigation Plugin</i>	40
3.4.2	<i>Camera Plugin</i>	41
3.4.3	<i>Depth Camera Plugin</i>	42
3.4.4	<i>Legs Plugin</i>	43
3.5	Pengembangan <i>Behavior Node</i>	45
3.5.1	<i>Move For Node</i>	46
3.5.2	<i>Patrol Position Node</i>	47
3.5.3	<i>Image Viewer Node</i>	48
3.5.4	<i>Legs Teleop Node</i>	49
3.5.5	<i>Pose Detector Node</i>	50
3.5.6	<i>RTAB-Map Node</i>	51
3.6	Integrasi Sistem pada <i>Real Robot</i>	52
3.6.1	<i>Navigation Node</i>	53
3.6.2	<i>V4L2 Camera Node</i>	54
3.6.3	<i>Kinect2 Node</i>	54
4	HASIL DAN PENGUJIAN	57
4.1	Pengujian Gerakan	58
4.1.1	Pengujian Gerakan Linier dan Estimasi Posisi pada Robot di Simulasi	58
4.1.2	Pengujian Gerakan Linier dan Estimasi Posisi pada <i>Real Robot</i>	61
4.1.3	Pengujian Gerakan Putar dan Estimasi Orientasi pada Robot di Simulasi	63
4.1.4	Pengujian Gerakan Putar dan Estimasi Orientasi pada <i>Real Robot</i>	65
4.2	Pengujian Citra Kamera	66

4.2.1	Pengujian Pengiriman Citra Kamera pada Robot di Simulasi	67
4.2.2	Pengujian Pengiriman Citra Kamera pada <i>Real Robot</i>	69
4.2.3	Pengujian Pengiriman Citra Kamera Antar-perangkat pada Robot di Simulasi	71
4.2.4	Pengujian Pengiriman Citra Kamera Antar-perangkat pada <i>Real Robot</i>	72
4.3	Pengujian <i>Depth Camera</i>	74
4.3.1	Pengujian Citra <i>Depth Camera</i> pada Robot di Simulasi	74
4.3.2	Pengujian Citra <i>Depth Camera</i> pada <i>Real Robot</i>	75
4.4	Pengujian Deteksi Pose	76
4.4.1	Pengujian Deteksi Pose Pengguna pada Robot di Simulasi	77
4.5	Pengujian SLAM	78
4.5.1	Pengujian Pemetaan Ruangan pada Robot di Simulasi	78
5	PENUTUP	81
5.1	Kesimpulan	81
5.2	Saran	82
DAFTAR PUSTAKA		88

DAFTAR GAMBAR

2.1	Pembagian kategori pada <i>assistive robots</i> menurut Heerink et al. [1].	6
2.2	Diagram komunikasi antar- <i>node</i> melalui <i>topic</i> dan <i>service</i> yang ada pada ROS 2 [2].	9
2.3	Diagram komunikasi antar- <i>node</i> melalui <i>action</i> yang ada pada ROS 2 [3].	10
2.4	Diagram sistem yang ada pada ROS 2 [4].	12
2.5	Diagram abstraksi interface yang ada pada ROS 2 [4].	14
2.6	Contoh tampilan rqt-graph (<i>ROS Graph</i>) [5].	16
2.7	Contoh tampilan GUI dari simulator Gazebo [6]. . .	17
2.8	Diagram kinematik robot dengan tiga dan empat roda <i>omni-directional</i> [7].	20
2.9	<i>Smart assistive posture device</i> dengan <i>human-chair system</i> [8].	21
2.10	Diagram cara kerja sistem <i>time of flight</i> [9].	22
2.11	Perangkat <i>depth camera</i> Kinect V2 [10].	23
2.12	Contoh penggunaan MediaPipe untuk deteksi <i>face mesh</i> , <i>iris</i> , tangan, pose tubuh, dan <i>holistic</i> [11]. .	24
2.13	Daftar <i>landmarks</i> yang ada pada BlazePose [12]. .	25
3.1	Blok diagram dari alur pekerjaan.	27
3.2	Desain dan diagram komponen dari robot <i>Dienen</i> . .	29
3.3	Diagram komponen dari prototipe robot <i>Dienen</i> . .	30
3.4	Tampilan lingkungan simulasi <i>outdoor</i> pada Gazebo.	37
3.5	Tampilan lingkungan simulasi <i>indoor</i> pada Gazebo..	39
3.6	Diagram integrasi <i>plugin</i> untuk simulasi.	39

3.7	Diagram integrasi <i>plugin</i> untuk model pengguna di simulasi.	43
3.8	Tampilan GUI dari <i>image viewer node</i>	48
3.9	Tampilan GUI dari <i>RTAB-Map node</i>	51
3.10	Diagram integrasi sistem pada <i>real robot</i>	52
4.1	Relasi antar- <i>node</i> dari pengujian gerakan linier dan estimasi posisi pada robot di simulasi.	58
4.2	Grafik estimasi posisi dari gerakan linier pada robot di simulasi.	60
4.3	Relasi antar- <i>node</i> dari pengujian gerakan linier dan estimasi posisi pada <i>real robot</i>	61
4.4	Grafik estimasi posisi dari gerakan linier pada <i>real robot</i>	63
4.5	Grafik estimasi orientasi dari gerakan putar pada robot di simulasi.	64
4.6	Grafik estimasi orientasi dari gerakan putar pada <i>real robot</i>	66
4.7	Relasi antar- <i>node</i> dari pengujian pengiriman citra kamera di simulasi.	67
4.8	Grafik <i>delay</i> dan frekuensi dari pengiriman citra pada robot di simulasi.	68
4.9	Relasi antar- <i>node</i> dari pengujian pengiriman citra kamera pada <i>real robot</i>	69
4.10	Grafik <i>delay</i> dan frekuensi dari pengiriman citra pada <i>real robot</i>	70
4.11	Grafik <i>delay</i> dan frekuensi dari pengiriman citra antar-perangkat pada robot di simulasi.	72
4.12	Grafik <i>delay</i> dan frekuensi dari pengiriman citra antar-perangkat pada <i>real robot</i>	74
4.13	Relasi antar- <i>node</i> dari pengujian citra <i>depth camera</i> pada robot di simulasi.	75

4.14 Perbandingan hasil tangkapan citra berwarna dan ci- tra kedalaman di simulasi.	75
4.15 Relasi antar- <i>node</i> dari pengujian citra <i>depth camera</i> pada <i>real robot</i>	76
4.16 Perbandingan hasil tangkapan citra berwarna dan ci- tra kedalaman pada <i>real robot</i>	77
4.17 Relasi antar- <i>node</i> dari pengujian deteksi pose peng- guna pada robot di simulasi.	77
4.18 Hasil deteksi pose pengguna pada robot di simulasi.	78
4.19 Relasi antar- <i>node</i> dari pengujian pemetaan ruangan pada robot di simulasi.	79
4.20 Proses dan hasil pemetaan ruangan pada robot di simulasi.	79

[Halaman ini sengaja dikosongkan]

DAFTAR KODE

2.1	Contoh <i>file msg</i> yang mendeskripsikan <i>interface</i> untuk suatu <i>topic</i>	13
2.2	Contoh <i>file SDFormat</i> yang mendeskripsikan komponen dari suatu model.	18
2.3	Contoh model <i>plugin</i>	19
3.1	<i>Link element</i> untuk bagian kepala robot.	32
3.2	<i>Joint element</i> yang menghubungkan bagian pundak kiri dan bagian badan atas robot.	32
3.3	<i>Joint element</i> yang menghubungkan bagian badan atas dan bagian badan bawah robot.	33
3.4	<i>Sensor element</i> dari sensor kamera.	33
3.5	<i>Sensor element</i> dari sensor <i>depth camera</i>	34
3.6	Struktur SDFormat dari model pengguna.	35
3.7	Struktur SDFormat dari lingkungan simulasi <i>outdoor</i>	36
3.8	Struktur SDFormat dari lingkungan simulasi <i>indoor</i>	38
3.9	<i>Class</i> dari <i>navigation plugin</i>	40
3.10	Integrasi <i>navigation plugin</i> pada model robot.	41
3.11	Integrasi <i>camera plugin</i> pada model robot.	41
3.12	Integrasi <i>depth camera plugin</i> pada model robot.	43
3.13	<i>Class</i> dari <i>legs plugin</i>	44
3.14	Integrasi <i>legs plugin</i> pada model pengguna.	45
3.15	Program <i>move for node</i>	46
3.16	Program <i>patrol position node</i>	47
3.17	Program <i>legs teleop node</i>	49
3.18	Program <i>pose detector node</i>	50
3.19	<i>Class</i> dari <i>navigation node</i>	53
3.20	<i>Class</i> dari <i>Kinect2 node</i>	54

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

4.1	Spesifikasi komputer untuk menjalankan simulator	57
4.2	Spesifikasi komputer yang ada pada prototipe robot	57
4.3	Hasil estimasi posisi dari gerakan linier pada robot di simulasi selama 3 detik.	59
4.4	Hasil estimasi posisi dari gerakan linier pada <i>real robot</i> selama 3 detik.	62
4.5	Hasil estimasi orientasi dari gerakan putar pada robot di simulasi selama 3 detik.	64
4.6	Hasil estimasi orientasi dari gerakan putar pada <i>real robot</i> selama 3 detik.	65
4.7	Hasil <i>delay</i> dan frekuensi dari pengiriman citra kamera pada robot di simulasi.	67
4.8	Hasil <i>delay</i> dan frekuensi dari pengiriman citra kamera pada <i>real robot</i>	69
4.9	Hasil <i>delay</i> dan frekuensi dari pengiriman citra kamera antar-perangkat pada robot di simulasi.	71
4.10	Hasil <i>delay</i> dan frekuensi dari pengiriman citra kamera antar-perangkat pada <i>real robot</i>	73

[Halaman ini sengaja dikosongkan]

BAB I

PENDAHULUAN

Penelitian ini dilatarbelakangi oleh berbagai kondisi yang menjadi acuan. Selain itu juga terdapat beberapa permasalahan yang akan dijawab sebagai luaran dari penelitian.

1.1 Latar Belakang

Selama beberapa tahun terakhir, robot telah mengalami perkembangan yang signifikan dari robot beroda untuk edukasi [13] hingga robot manipulator untuk skala industri [14]. Salah satu bentuk perkembangan lain dari robot tersebut adalah *socially assistive robots* (SARs). SARs merupakan jenis robot dalam bidang *socially assistive robotics* yang menggabungkan aspek yang ada pada *assistive robotics* dan *socially interactive robotics* sehingga menjadikan SARs sebagai robot yang mampu memberikan bantuan kepada pengguna dalam bentuk interaksi sosial [15].

Namun, karena sifat dari SARs yang melibatkan interaksi langsung dengan pengguna, maka pengujian dari robot akan menjadi sulit dan beresiko bagi pengguna yang ikut terlibat dalam pengujian tersebut [16]. Salah satu solusi untuk mengatasi masalah tersebut adalah dengan melakukan pengujian secara virtual melalui simulasi robot. Selain bisa meminimalisir resiko, penggunaan simulasi robot juga bisa mengurangi biaya yang dibutuhkan dan menghemat waktu pengujian selama pengembangan robot tersebut [17].

Hingga saat ini sudah ada beberapa simulator yang bisa digunakan untuk menjalankan simulasi robot seperti Webots [18], Gazebo [19], V-REP [20], OpenAI Gym [21], dan lain sebagainya. Namun, simulator-simulator tersebut hanyalah platform yang secara umum digunakan untuk membantu pengembangan robot melalui simulasi virtual. Sedangkan pengembangan dari lingkungan simulasi dan kontroler robot untuk simulasi tersebut harus dibuat sendiri oleh pengembang robot.

Untuk itu, pada penelitian ini kami mengajukan penelitian terkait pengembangan lingkungan simulasi untuk pengujian SARs menggunakan ROS 2 dan Gazebo. ROS 2 dan Gazebo sendiri dipilih karena tersedianya banyak *library* yang dapat membantu pengembangan maupun pengujian robot, terutama untuk simulasi. Selain itu, dengan adanya ROS 2, kontroler robot yang diuji melalui simulasi bisa dengan mudah dipindahkan ke robot fisik untuk diuji secara langsung pada pengguna [17].

1.2 Permasalahan

Dari latar belakang yang telah dipaparkan sebelumnya, maka permasalahan yang dapat diambil adalah pengujian SARs memiliki resiko terhadap keselamatan pengguna serta dapat memakan biaya dan waktu yang besar. Untuk itu, perlu adanya lingkungan simulasi yang dapat mensimulasikan lingkungan serta objek-objek yang ada di dalamnya sehingga pengujian SARs dapat dilakukan di lingkungan simulasi tersebut, sembari memastikan pengujian yang dilakukan di simulasi juga bisa dilakukan dengan hasil yang sama pada robot fisik.

1.3 Penelitian Terkait

Beberapa penelitian sebelumnya telah berhasil dalam mengembangkan lingkungan simulasi untuk robot menggunakan ROS (Pendahulu ROS 2) dan Gazebo. Seperti yang dilakukan oleh Qian et al. [22] yang mengembangkan simulasi untuk robot *manipulator*, Zhang et al. [23] yang mengembangkan simulasi untuk robot *quadrotor UAV*, dan Takaya et al. [17] yang mengembangkan lingkungan simulasi untuk pengujian terhadap *mobile robot*. Namun, berbeda dengan penelitian yang telah dilakukan sebelumnya, penelitian yang kami lakukan memilih menggunakan ROS 2 agar kontroler robot yang dibuat untuk simulasi memiliki performa yang lebih baik serta dapat bekerja secara *real-time* [24].

Selain itu, penelitian lain juga telah dilakukan oleh Erickson et al. [16] yang mengembangkan Assistive Gym, sebuah *framework* simulasi untuk *assistive robotics* berbasis OpenAI Gym. *Framework* simulasi tersebut kemudian digunakan oleh Clegg et al. [25] untuk

mengembangkan metode *learning* melalui simulasi pada kolaborasi antara robot dengan manusia dalam membantu pemakaian baju pada manusia. Namun, karena tidak menggunakan ROS, kontroler robot yang dibuat untuk simulasi yang menggunakan *framework* tersebut perlu dibuat ulang ketika akan diujikan secara langsung pada pengguna menggunakan robot fisik.

1.4 Tujuan

Tujuan dari penelitian ini adalah untuk membuat sebuah lingkungan simulasi menggunakan Gazebo yang dapat digunakan untuk melakukan pengujian pada SARS secara virtual. Kemudian, sistem kontroler yang digunakan oleh robot dibuat secara terpisah dan ter-abstraksi menjadi beberapa komponen dalam bentuk *ROS 2 nodes*. Terakhir, sistem yang telah dikembangkan akan diujikan dan dibandingkan hasilnya menggunakan data yang berasal dari simulasi maupun yang berasal dari kondisi *real*.

1.5 Batasan Masalah

Untuk memfokuskan permasalahan yang diangkat, maka penelitian ini dilakukan dengan mengembangkan lingkungan simulasi beserta kontroler untuk pengujian pada robot *Dienen* secara virtual. Lingkungan simulasi dan kontroler tersebut dikembangkan menggunakan Gazebo 11 dan ROS 2 Foxy Fitzroy pada *platform* Ubuntu 20.04. Agar bisa diujikan pada robot fisik, komponen pada robot yang diujikan di simulasi adalah komponen yang saat ini tersedia di prototipe robot *Dienen*. Pada penelitian ini, pengujian yang dilakukan terfokus pada kemampuan model robot di lingkungan simulasi untuk melakukan tindakan dengan hasil yang sama dengan yang bisa dilakukan di robot fisik.

1.6 Sistematika Penulisan

Buku penelitian tugas akhir ini disusun dalam sistematika yang terstruktur agar mudah dipahami dan dipelajari oleh pembaca. Bab 1 pada buku ini berisi uraian tentang latar belakang, permasalahan yang diangkat, penelitian terkait, tujuan, serta batasan masalah dari penelitian. Kemudian bab 2 menguraikan teori-teori penunjang

yang berhubungan dengan penelitian ini, seperti uraian tentang *assistive robotics*, simulator Gazebo, Robot Operating System (ROS), dan lain sebagainya. Pada bab 3, desain dan implementasi dari sistem yang dibuat diuraikan, dari perancangan model robot dan pengguna untuk simulasi, pengembangan lingkungan simulasi, dan integrasi sistem kontroler pada simulasi dan pada robot fisik. Lalu pada bab 4, dijelaskan pengujian yang dilakukan dari sistem yang sudah dibuat dan hasil yang didapatkan. pengujian ini dilakukan dengan beberapa cara seperti pengujian terhadap gerakan robot, tangkapan citra kamera, pemetaan ruangan, dan lain sebagainya. Terakhir, kesimpulan dari penelitian serta saran untuk penelitian berikutnya diuraikan pada bab 5 yang ada di buku ini.

BAB II

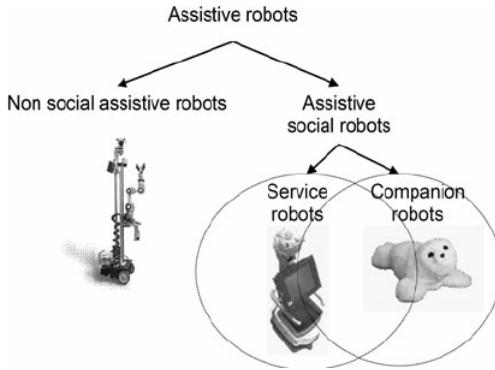
TINJAUAN PUSTAKA

Pada bab ini akan dijelaskan teori-teori penunjang yang digunakan sebagai bahan acuan dan referensi untuk penelitian yang dilakukan. Teori-teori yang dijelaskan pada bab ini akan dipaparkan dalam urutan yang sistematis, dimulai dari hal paling mendasar yang digunakan pada penelitian ini seperti penjelasan mengenai *assistive robotics*, Gazebo, dan ROS, hingga penjelasan lebih dalam yang berhubungan dengan sistem yang dibuat dan pengujian yang dilakukan seperti Kinect dan SLAM.

2.1 *Assistive Robotics*

Assistive robotics (AR) merupakan cabang dalam bidang robotika yang terfokus pada pengembangan robot untuk kegiatan *assistive*. Pada umumnya, kegiatan *assistive* tersebut berupa pemberian bantuan fisik kepada pengguna yang memiliki kekurangan fisik seperti lansia dan penyandang disabilitas. Feil-Seifer dan Mataric [15] memaparkan, riset pada *assistive robotics* meliputi robot untuk rehabilitasi, robot yang membantu mobilitas seperti pada kursi roda otomatis, robot pendamping, robot dengan lengan *manipulator* untuk membantu pengguna yang tidak mampu secara fisik, dan robot edukasi. Lebih lanjut mereka menjelaskan bahwa definisi *assistive robots* yang hanya memberikan bantuan secara fisik kepada pengguna dirasa kurang tepat karena ada juga *assistive robots* yang memberikan bantuan melalui interaksi sosial tanpa adanya kontak fisik seperti pada robot pendamping dan robot edukasi.

Secara kategori, Heerink et al. [1] membagi *assistive robots* ke dalam dua kategori, yakni *non social assistive robots* dan *social assistive robots*. Seperti yang terlihat pada gambar 2.1, *non social assistive robots* mencakup robot yang digunakan untuk memberikan bantuan secara fisik kepada pengguna tanpa adanya interaksi sosial, sedangkan *social assistive robots* mencakup robot yang mela-



Gambar 2.1: Pembagian kategori pada *assistive robots* menurut Herink et al. [1].

kukan tindakan *assistive* dengan melibatkan interaksi sosial terhadap pengguna. Lebih lanjut, *social assistive robots* terbagi lagi ke dalam dua kategori, yakni *service robots* dan *companion robot*. *Service robots* merupakan robot yang memberikan bantuan fisik dan kognitif kepada pengguna, serta bertindak seolah sebagai pelayan, sedangkan *companion robot* merupakan robot yang bertindak seolah sebagai pendamping, sahabat, dan media terapi secara sosial.

2.1.1 *Socially Assistive Robots* (SARs)

Socially assistive robots (SARs) merupakan jenis robot dalam bidang *socially assistive robotics* yang menggabungkan aspek yang ada pada *assistive robotics* dan *socially interactive robotics*. SARs memiliki tujuan yang sama dengan robot di bidang *assistive robotics*, yakni dalam hal memberikan bantuan kepada pengguna secara *assistive*, namun pada SARs, bantuan tersebut secara spesifik diberikan melalui interaksi sosial kepada pengguna. Karena adanya aspek interaksi sosial tersebut, SARs memiliki tujuan yang sama dengan robot di bidang *socially interactive robotics*.

Rich dan Sidner [26] memaparkan, SARs mampu memberikan bantuan kepada pengguna dalam berbagai cakupan. Cakupan tersebut terdiri atas kemampuan SARs untuk memberikan dukungan

fungsional dan kognitif kepada pengguna, memberikan kesempatan bagi pengguna untuk meningkatkan partisipasi sosial dan kesehatan psikologis, menyediakan pemantauan jarak jauh dan berkelanjutan atas status kesehatan pengguna, serta memfasilitasi pengguna untuk melakukan perilaku hidup sehat dan pencapaian tujuan yang berhubungan dengan kesehatan.

2.2 Robot Operating System 2 (ROS 2)

Robot Operating System (ROS) [27] merupakan sebuah *robotics middleware* bersifat *open source* yang digunakan untuk membantu pengembangan sistem yang ada pada robot. Fitur utama yang ada pada ROS adalah desain sistem dimana data yang diterima maupun dikirim ke setiap komponen yang ada pada robot dilakukan secara terabstraksi dengan adanya *hardware abstraction*. Dengan adanya hal tersebut, algoritma yang ada pada suatu program dapat digunakan pada robot yang berbeda, terlepas dari perbedaan perangkat yang ada di setiap robot.

Untuk memungkinkan adanya *hardware abstraction*, ROS menggunakan *graph architecture* yang memungkinkan suatu *node* untuk berkomunikasi dengan *node* lain melalui sebuah *topic* yang sama. Pada ROS, *hardware abstraction* tersebut terbentuk dengan menggunakan *topic* yang memiliki nama dan struktur data yang sama yang merepresentasikan jenis komponen yang dimiliki suatu robot. Nantinya, Setiap node yang ada dapat menerima maupun mengirimkan data melalui *topic* tersebut, terlepas dari bagaimana maupun darimana data yang dikirimkan tersebut berasal.

Pada ROS terdapat berbagai macam *tools* yang sudah tersedia untuk membantu pengembangan sistem yang ada pada robot seperti *command-line* yang membantu dalam *debugging* program, konfigurasi parameter secara dinamis ketika program sedang dijalankan, *monitoring* data internal yang ada pada robot secara visual, dan lain sebagainya. Selain itu, dengan *package management* yang ada pada ROS, setiap orang bisa menggunakan *package* yang sudah ada maupun menambahkan *package* baru untuk mempermudah pengembangan lebih lanjut dari suatu robot, tanpa perlu membuat ulang keseluruhan sistem yang dibutuhkan.

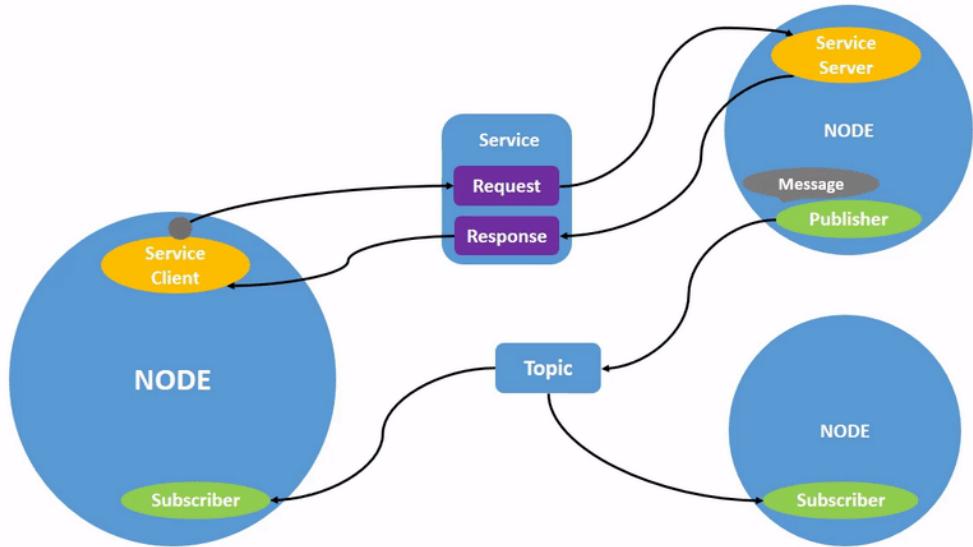
Generasi kedua dari Robot Operating System, ROS 2, merupakan kelanjutan dari ROS yang mengusung reliabilitas dan performa untuk penggunaan *real-time* pada robot sembari masih mendukung fitur yang dimiliki oleh ROS sebelumnya [24]. Berbeda dengan pendahulunya yang menggunakan TCPROS/UDPROS sebagai sistem komunikasi yang digunakan di setiap node, ROS 2 menggunakan *Data Distribution Service* (DDS) [28] [29], standar industri untuk sistem komunikasi *real-time* dan *end-to-end middleware*. Dengan penggunaan DDS tersebut, ROS 2 akan lebih terfokus pada penggunaan *middleware* di bidang robotika, sedangkan komunikasi tingkat rendah yang dilakukan diantara setiap *node* akan dikembangkan kepada implementasi DDS yang digunakan.

2.2.1 *ROS 2 Node*

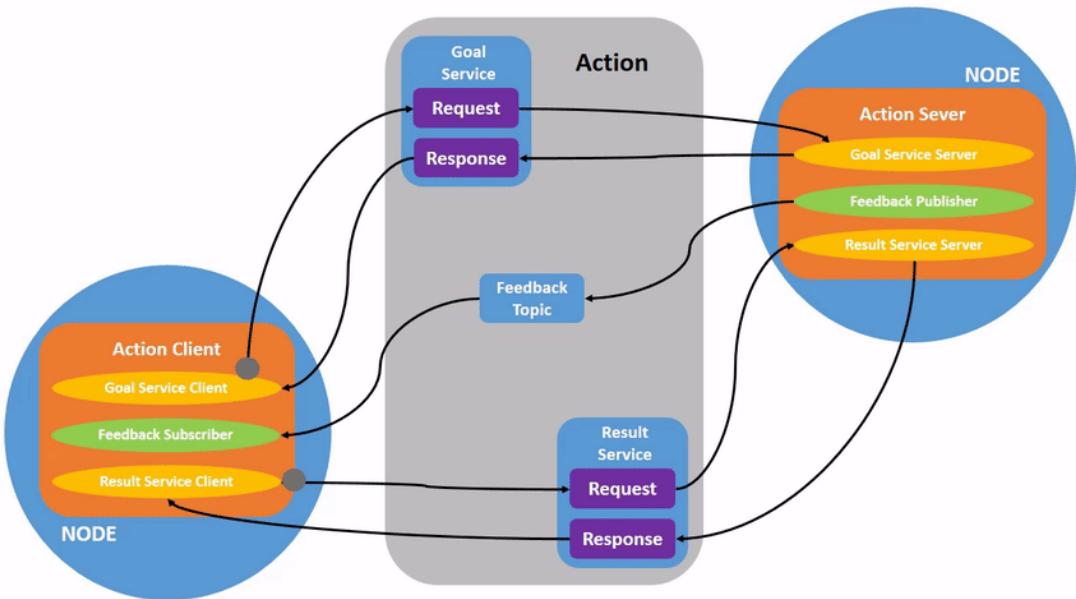
ROS 2 node merupakan komponen utama dari *graph architecture* yang ada pada ROS 2. *ROS 2 node* umumnya digunakan untuk merepresentasikan suatu proses tunggal yang ada pada robot, seperti untuk mengakses motor, mengolah gambar, dan sebagainya [2]. Setiap *node* yang ada, nantinya dapat berkomunikasi dengan node lain melalui dua cara, seperti yang terlihat pada gambar 2.2, sebuah *node* dengan *publisher* dapat mengirimkan sebuah data melalui *topic* yang nantinya akan disalurkan ke *node* lain dengan *subscriber* yang terhubung dengan *topic* tersebut. Selain itu, sebuah *node* dengan *service client* juga dapat mengirimkan sebuah data *request* melalui *service* yang nantinya akan diolah dan dikirim balik dalam bentuk data *response* oleh *node* lain yang memiliki *service server* yang terhubung dengan *service* tersebut.

Proses komunikasi antar-*node* yang ada tersebut akan dilakukan secara terabstraksi, dimana suatu *node* akan mengirimkan data ke suatu *topic* maupun *service*, terlepas dari apakah data tersebut akan diterima oleh *node* lain atau tidak. Setiap *node* nantinya cukup mengirimkan data ke *topic* atau *service* dengan nama dan tipe *interface* yang sesuai agar data tersebut dapat dikirimkan secara terabstraksi kepada *node* lain.

Selain *topic* dan *service*, ROS 2 juga mengenal bentuk lain dari jalur komunikasi antar-*node* yang disebut sebagai *action*. *Action* merupakan bentuk kompleks dari *topic* dan *service* dimana selain



Gambar 2.2: Diagram komunikasi antar-node melalui *topic* dan *service* yang ada pada ROS 2 [2].



Gambar 2.3: Diagram komunikasi antar-node melalui *action* yang ada pada ROS 2 [3].

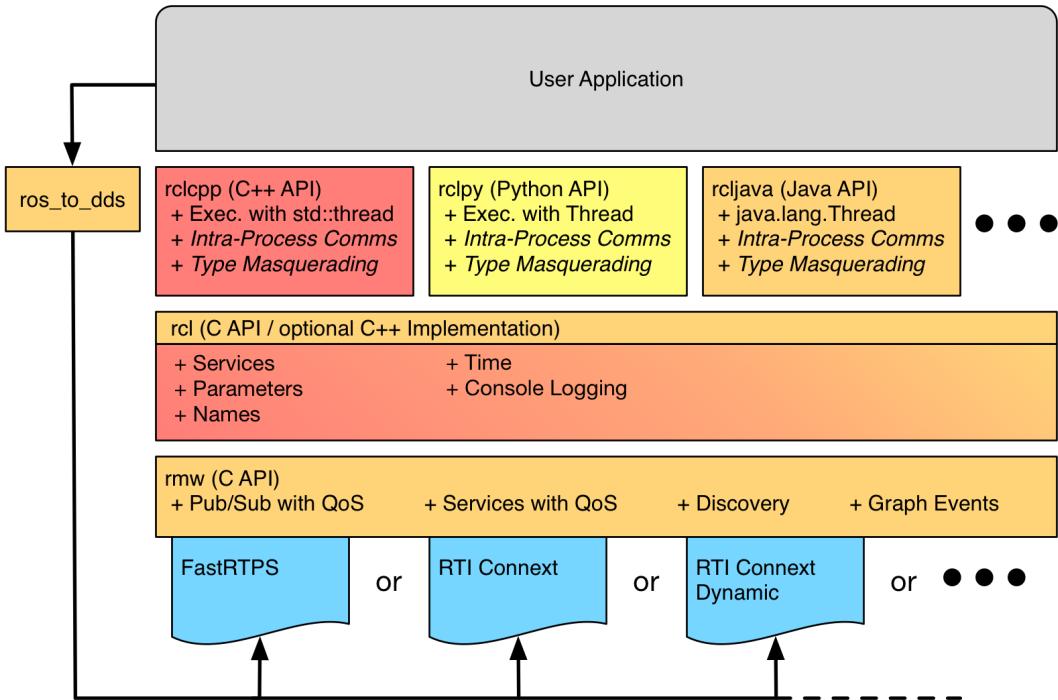
menerima data *response*, suatu *node* juga dapat menerima data *feedback* secara terus menerus sampai proses yang dijalankan selesai. Seperti yang terlihat pada gambar 2.3, suatu *node* dengan *action client* akan mengirimkan data *goal service* melalui sebuah *action* kepada *node* lain dengan *action server* yang terhubung kepada *action* tersebut. Nantinya, selama proses yang dilakukan oleh *node* dengan *action server* belum selesai, *node* tersebut dapat mengirimkan data *feedback topic* kepada *node* dengan *action client*. Terakhir, jika proses selesai, *node* dengan *action client* dapat mengirimkan data *result service* yang nantinya dapat direspon oleh *node* dengan *action server*.

2.2.2 ROS 2 Middleware Implementations

ROS 2 middleware implementations (RMW) merupakan implementasi dari sistem komunikasi *middleware* yang ada pada ROS 2, menggantikan TCPROS/UDPROS yang ada pada ROS yang sebelumnya [30]. Pada ROS 2, *middleware* yang digunakan berbasis pada sistem yang dimiliki oleh DDS, dimana DDS sendiri memiliki berbagai macam implementasi seperti RTI Connexx DDS [30] eProxima Fast DDS [31], Eclipse Cyclone DDS [32], dan sebagainya.

Seperti yang terlihat pada gambar 2.4, sistem komunikasi yang ada pada ROS 2 terdiri atas *user application* di bagian paling atas, kemudian *ROS 2 client library* (RCL) dengan implementasi bahasanya masing-masing, dan terakhir RMW yang terhubung dengan berbagai macam implementasi DDS. Di sini, peran RMW adalah sebagai penghubung antara RCL dengan implementasi DDS yang digunakan. Sehingga komunikasi pada tingkat *middleware* seperti *publish/subscribe* dengan QoS, *discovery*, dan *graph event* dapat dilakukan terlepas dari implementasi yang digunakan.

Saat ini ROS 2 sudah mendukung beberapa implementasi DDS yang bisa digunakan sebagai RMW dari sistem yang digunakan. Implementasi tersebut dibentuk sebagai *ROS 2 package* seperti pada *package* rmw_connexxdds yang digunakan untuk menjalankan implementasi DDS dari RTI Connexx DDS, *package* rmw_fastrtps yang digunakan untuk menjalankan implementasi DDS dari eProxima Fast DDS, dan lain sebagainya.



* *Intra-Process Comms* and *Type Masquerading* could be implemented in the client library, but may not currently exist.

Gambar 2.4: Diagram sistem yang ada pada ROS 2 [4].

2.2.3 ROS 2 Client Libraries

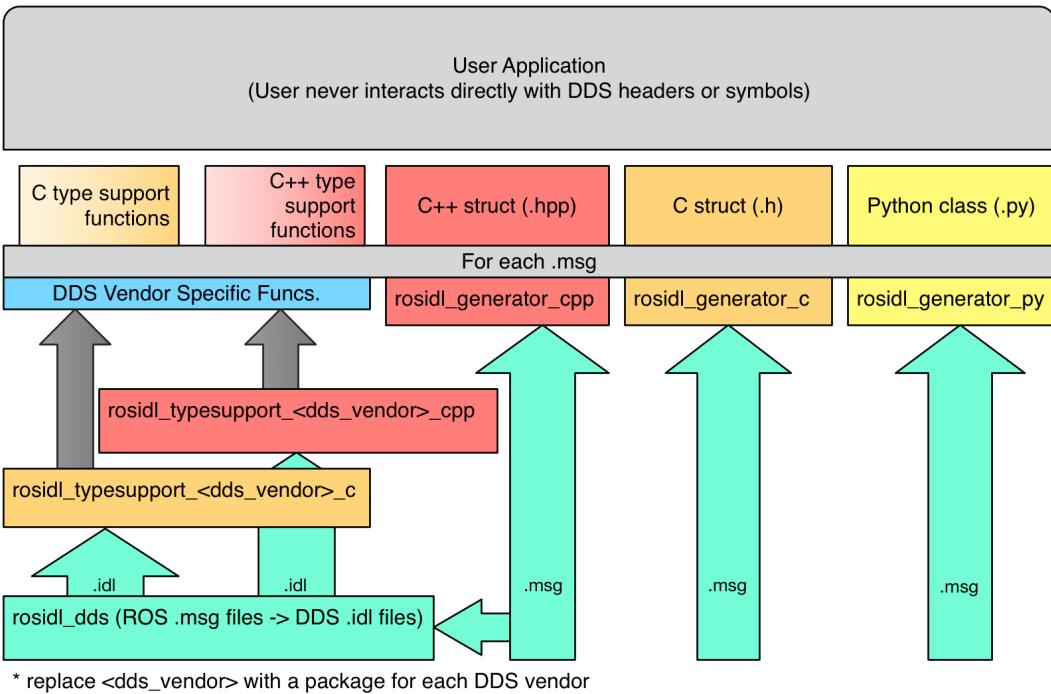
ROS 2 client libraries (RCL), atau *ROS 2 client interfaces*, merupakan sekumpulan *high-level libraries* yang digunakan untuk menjalankan fungsionalitas yang ada di ROS 2 seperti pembuatan *node*, *publisher*, *subscription*, dan lain sebagainya [33]. Seperti yang terlihat pada gambar 2.4, RCL berperan untuk menghubungkan *user application* dengan RMW, dimana *user space* ini merupakan keseluruhan lingkup yang digunakan oleh pengguna untuk menjalankan sistem yang ada pada robot.

Untuk saat ini, RCL dapat digunakan pada berbagai macam bahasa pemrograman, semuanya dibentuk sebagai *ROS 2 package* yang berisi *headers*, *modules*, maupun *libraries* dari bahasa pemrograman tersebut. Sebagai contoh, untuk bahasa pemrograman C bisa menggunakan *package* rcl, untuk bahasa pemrograman C++ bisa menggunakan *package* rclcpp, dan untuk bahasa pemrograman Python bisa menggunakan *package* rclpy. Selain itu ada juga implementasi RCL untuk bahasa pemrograman lain yang masih dalam tahap pengembangan seperti pada *package* rclnodejs [34] untuk bahasa pemrograman JavaScript yang berbasis Node.js.

2.2.4 ROS 2 Interface

ROS 2 interface merupakan suatu bentuk struktur pesan yang digunakan pada sistem komunikasi yang ada di ROS 2. *Interface* yang ada di ROS 2 terdiri dari tiga jenis, *msg* dengan *file format* .msg yang digunakan pada *ROS 2 topic*, *srv* dengan *file format* .srv yang digunakan pada *ROS 2 service*, dan *action* dengan *file format* .action yang digunakan pada *ROS 2 Action*.

ROS 2 interface dapat dideskripsikan dalam suatu file yang nantinya akan dikompilasi sesuai dengan format yang dibutuhkan oleh RCL maupun RMW. Seperti pada potongan kode 2.1, sebuah *file interface* berisi beberapa baris yang menunjukkan tipe data, nama, serta nilai *default* dari setiap *field* yang dimiliki oleh *interface* tersebut. Sebuah *interface* yang ada di ROS 2 bisa memiliki tipe data yang berasal dari *interface* lain, sehingga memungkinkan bentuk *interface* yang lebih kompleks seperti yang digunakan pada *interface* untuk informasi kamera dan data citra.



Gambar 2.5: Diagram abstraksi interface yang ada pada ROS 2 [4].

Kode 2.1: Contoh *file msg* yang mendeskripsikan *interface* untuk suatu *topic*.

```
1 uint8 x 42
2 int16 y -2000
3 string full_name
4 float32 [] samples
```

Seperti yang dijelaskan pada paragraf sebelumnya, *ROS 2 interface* akan dikompilasi menjadi bentuk lain agar bisa digunakan di setiap implementasi yang ada di RCL maupun RMW. Seperti yang terlihat pada gambar 2.5, sebuah *file .msg* akan dikompilasi menjadi *file .idl* yang digunakan oleh implementasi DDS pada RMW, serta akan dikompilasi menjadi *struct* dan *class* dari bahasa yang digunakan sebagai implementasi dari *RCL*. Dengan adanya kompilasi ini, isi dari sebuah pesan yang dikirim maupun diterima pada ROS 2 dapat dideskripsikan menggunakan satu format yang nantinya bisa digunakan di semua implementasi RMW maupun RCL.

2.2.5 *ROS 2 Command-line Interface*

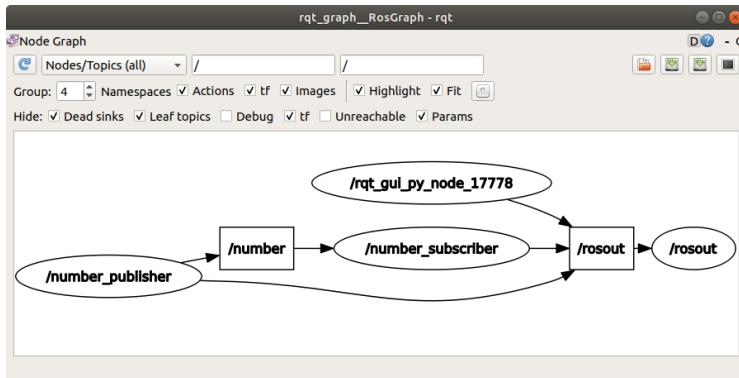
ROS 2 CLI (command-line interface) merupakan sekumpulan *tools* dalam bentuk *command-line* yang digunakan untuk membantu *debugging* pada komunikasi antar-*node* yang ada pada ROS 2. *Tools* ini terdiri dari beberapa perintah yang dapat digunakan untuk melakukan berbagai macam hal dari melihat daftar *node* yang ada pada sistem, isi dari data yang dikirim melalui suatu *topic*, daftar parameter yang dapat dilihat dan diubah, dan lain sebagainya.

Secara sistem, *ROS 2 CLI* terbentuk sebagai *ROS 2 packages* yang ditulis menggunakan bahasa pemrograman Python. Agar dapat bekerja untuk mengetahui berbagai *node*, *topic*, maupun *service* yang ada, *ROS 2 CLI* menggunakan *distributed discovery process* [35]. Berbeda dengan ROS sebelumnya yang bekerja secara terpusat menggunakan *ROS Master*, sistem ini bekerja lebih baik karena tidak terpusat, namun membutuhkan waktu yang lebih lama untuk melakukan proses pencarian *node*, *topic* maupun *service* yang ada. Oleh karena itu, untuk mempercepat proses pencarian, maka akan ada *daemon* yang bekerja secara terus menerus dan menyimpan *cache* dari struktur *graph* yang ada.

2.2.6 RQt

RQt merupakan *framework GUI* berbasis Qt [36] yang digunakan untuk mengimplementasikan *tools* dalam mengatur *node* yang ada pada ROS 2 [37]. RQt memiliki fungsi yang relatif sama dengan *ROS 2 CLI*, yakni untuk keperluan *debugging* pada sistem komunikasi yang ada pada ROS 2, namun RQt ini bekerja secara visual dalam bentuk GUI.

Pada RQt, terdapat beberapa *tools* standar yang dapat digunakan untuk melakukan berbagai macam hal seperti melihat data gambar secara visual, melihat hubungan komunikasi antar-*node* dalam bentuk diagram, dan lain sebagainya. Seperti yang terlihat pada gambar 2.6, RQt memiliki tools bernama *rqt_graph* (*ROS Graph*) yang dapat digunakan untuk melihat hubungan yang ada pada setiap *node* dalam bentuk *topic*, *service*, maupun *action*.



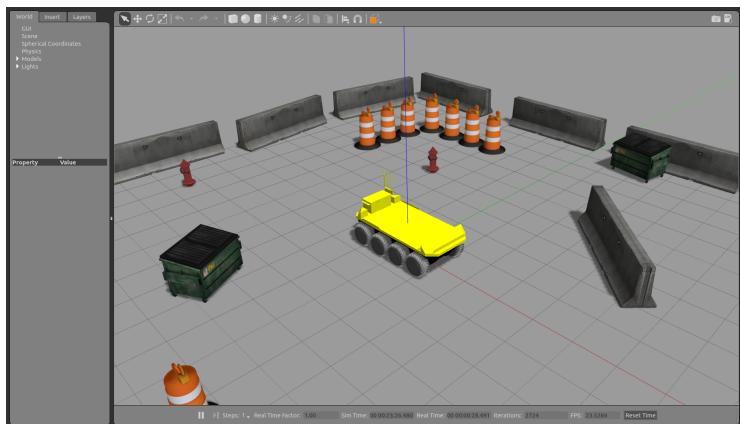
Gambar 2.6: Contoh tampilan *rqt_graph* (*ROS Graph*) [5].

Selain *tools* standar yang sudah disediakan, *tools* berbasis GUI pada RQt juga bisa dibuat sendiri maupun diubah menyesuaikan dengan kebutuhan yang diinginkan. Perubahan tersebut dapat dilakukan melalui *plugin* yang dibuat khusus untuk RQt. Nantinya *plugin* tersebut dapat disematkan pada *tools* RQt yang dikembangkan dan secara langsung dapat digunakan untuk mengakses sistem komunikasi yang ada pada ROS 2.

2.3 Gazebo

Gazebo [19] merupakan sebuah simulator robot bersifat *open source* yang dapat digunakan untuk mensimulasikan model robot secara virtual di lingkungan 3D. Awalnya, Gazebo merupakan bagian dari Player Project [38] yang memungkinkan sebuah simulasi robot 2D (Stage) dan 3D (Gazebo) dapat bekerja dengan sebuah program yang dibuat secara terabstraksi untuk robot (Player).

Untuk memungkinkan sebuah simulator robot dapat bekerja, Gazebo mengintegrasikan ODE dan Bullet sebagai physics engine, OpenGL untuk rendering model dan lingkungan, serta beberapa program lain untuk mensimulasikan sensor dan aktuator secara virtual. Gazebo memiliki berbagai macam sensor dan aktuator virtual yang dapat digunakan di lingkungan simulasi seperti sensor kamera, motor, *lidar*, IMU, dan lain sebagainya.



Gambar 2.7: Contoh tampilan GUI dari simulator Gazebo [6].

Pada Gazebo, keseluruhan sistem yang ada di simulasi akan dijalankan secara *headless* menggunakan program *gzserver*. Selain itu, seperti yang terlihat pada gambar 2.7, Gazebo juga menyediakan *gzclient*, sebuah program GUI yang memungkinkan pengguna untuk melihat tampilan dan memanipulasi model maupun lingkungan simulasi yang sedang berjalan di simulator.

2.3.1 Gazebo Model

Gazebo model merupakan bagian dari simulator Gazebo yang merepresentasikan objek yang ada di lingkungan simulasi seperti robot, pengguna, ruangan, meja, kursi, dan lain sebagainya. Selain *Gazebo Model*, dikenal juga istilah *Gazebo World* yang merepresentasikan keseluruhan lingkungan simulasi beserta model-model yang ada di dalamnya.

Kode 2.2: Contoh *file SDFormat* yang mendeskripsikan komponen dari suatu model.

```
1 <?xml version='1.0'?>
2 <sdf version="1.4">
3   <model name="my_model">
4     <pose>0 0 0.5 0 0 0</pose>
5     <link name="link">
6       <inertial>
7         <mass>1.0</mass>
8         <inertia>
9           ...
10        </inertia>
11      </inertial>
12      <collision name="collision">
13        ...
14      </collision>
15      <visual name="visual">
16        ...
17      </visual>
18    </link>
19  </model>
20</sdf>
```

Gazebo Model dibentuk dari sebuah *file SDFormat*, yang merupakan sebuah format berbasis XML untuk mendeskripsikan setiap komponen yang ada pada objek di lingkungan simulasi. Pada Gazebo, SDFormat menggantikan format URDF yang digunakan untuk mendeskripsikan robot yang ada pada ROS. Untuk mendeskripsikan sebuah model, seperti yang terlihat pada potongan kode 2.2, model dari suatu objek akan terbagi menjadi beberapa *link*, yang mana pada setiap *link* akan memiliki nilai inertia, massa, bentuk *collision*, serta tampilan visual yang digunakan untuk mensimulasikan objek tersebut secara virtual.

2.3.2 Gazebo Plugin

Gazebo plugin merupakan bagian dari simulator Gazebo yang memungkinkan dibuatnya program yang dapat memperoleh data maupun memanipulasi sistem yang ada di lingkungan simulasi [39]. *Plugin* tersebut dapat dibuat menggunakan bahasa pemrograman C++ yang nantinya akan dikompilasi menjadi sebuah *shared library* yang dapat diintegrasikan secara langsung pada simulator Gazebo.

Kode 2.3: Contoh model *plugin*.

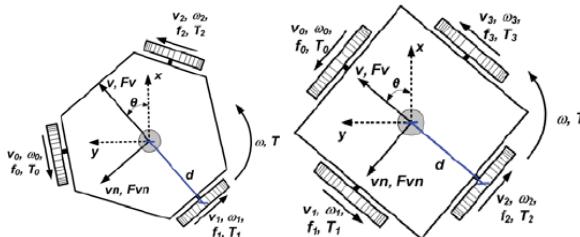
```
1 #include <gazebo/gazebo.hh>
2
3 namespace gazebo
4 {
5     class MyModelPlugin : public ModelPlugin
6     {
7         public: MyModelPlugin() : ModelPlugin()
8         {
9             ...
10        }
11
12        public: void Load(physics::WorldPtr world, sdf::ElementPtr sdf)
13        {
14            ...
15        }
16    };
17
18 GZ_REGISTER_WORLD_PLUGIN(MyModelPlugin)
19 }
```

Gazebo plugin terbagi menjadi beberapa jenis plugin, yakni *world plugin*, *model plugin*, *sensor plugin*, *system plugin*, *visual plugin*, dan *GUI plugin*. Sebagai contoh seperti pada potongan kode 2.3, sebuah *plugin* dibuat dari sebuah *class* yang diturunkan dari *class ModelPlugin*. *Class* tersebut memiliki sebuah fungsi *constructor* yang dilakukan untuk menginisialisasi variabel yang dimiliki objek *class* tersebut, dan sebuah fungsi *Load()* yang digunakan untuk mendapatkan informasi dari *element* yang ada pada SDFormat suatu model. Agar bisa digunakan sebagai *plugin* pada simulator Gazebo, *class* yang dibuat nantinya perlu diregistrasi menggunakan *macro GZ_REGISTER_WORLD_PLUGIN*.

2.4 Holonomic Robot

Holonomic robot merupakan jenis robot yang memiliki kemampuan untuk bergerak secara *holonomic*, yakni sebuah kondisi dimana jumlah *degree of freedom* (DoF) yang dapat dikendalikan sama dengan total DoF yang dimiliki. Salah satu bentuk dari *holonomic robot* adalah robot dengan roda *omni-directional*. Robot jenis ini memiliki kemampuan manuver yang bagus dan efisien dengan mengorbankan kompleksitas pada desain pergerakan robot. Menurut Oliveira et al. [7], sebuah robot dengan tiga atau lebih roda *omni-directional* mampu memiliki kemampuan pergerakan tangensial, normal, dan *angular* secara hampir independen ke segala arah.

Robot *holonomic* dengan roda *omni-directional* pada umumnya memiliki tiga hingga empat buah motor roda. Robot dengan tiga buah motor roda secara mekanik memiliki desain yang lebih sederhana, namun dengan adanya empat buah motor roda, robot dapat memiliki akselerasi yang lebih tinggi sembari mengurangi kemungkinan adanya *wheel slippage*.



Gambar 2.8: Diagram kinematik robot dengan tiga dan empat roda *omni-directional* [7].

Pada robot dengan roda *omni-directional*, model kinematik robot dapat dibentuk seperti pada gambar 2.8. Dari gambar tersebut, dihasilkan persamaan 2.1 untuk robot dengan tiga buah roda dan persamaan 2.2 untuk robot dengan empat buah roda. Pada persamaan tersebut, v_i merupakan kecepatan setiap motor roda yang dimiliki robot, $v(t)$ dan $vn(t)$ merupakan kecepatan linier robot, dan $w(t)$ merupakan kecepatan putar robot.

$$\begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} -\sin(\pi/3) & \cos(\pi/3) & d \\ 0 & -1 & d \\ \sin(\pi/3) & \cos(\pi/3) & d \end{bmatrix} \begin{bmatrix} v(t) \\ vn(t) \\ \omega(t) \end{bmatrix} \quad (2.1)$$

$$\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & d \\ -1 & 0 & d \\ 0 & -1 & d \\ 1 & 0 & d \end{bmatrix} \begin{bmatrix} v(t) \\ vn(t) \\ \omega(t) \end{bmatrix} \quad (2.2)$$

2.5 Inertial Measurement Unit (IMU)

Inertial measurement unit (IMU) merupakan sebuah perangkat elektronik yang digunakan untuk mengukur besaran gaya, kecepatan putar, dan terkadang orientasi dari suatu objek. IMU pada umumnya terdiri atas sebuah sensor akselerometer, sebuah sensor giroskop, dan terkadang juga sebuah sensor magnetometer.

Pada robot, IMU pada umumnya digunakan untuk melakukan perhitungan estimasi orientasi. Perhitungan tersebut bisa dilakukan menggunakan beberapa metode, salah satunya adalah dengan menggunakan *kalman filter* yang sudah dimodifikasi untuk mendapatkan orientasi robot dalam bentuk *quaternion* [40].

2.6 Smart Assistive Posture Device

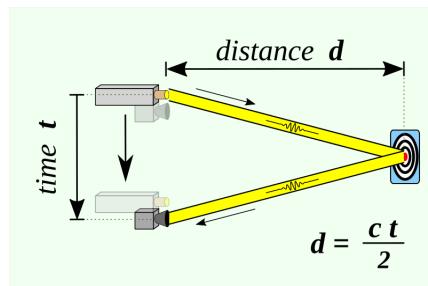


Gambar 2.9: *Smart assistive posture device* dengan *human-chair system* [8].

Smart assistive posture device merupakan sebuah perangkat cerdas yang umumnya digunakan di industri untuk membantu pekerja dalam meringankan ketegangan otot. Berbagai perangkat dengan fungsi tersebut telah banyak dikembangkan, salah satunya adalah perangkat dengan *human-chair system* [8]. Perangkat dengan sistem tersebut memiliki fungsi untuk mengurangi ketegangan pada otot kaki bagi pengguna ketika posisi duduk. Pada perangkat tersebut, pengguna dimodelkan sebagai empat batang saling terhubung yang terdiri atas bagian atas tubuh, paha, betis, dan telapak kaki. Seperti pada gambar 2.9, Perangkat dengan sistem tersebut sendiri merepresentasikan dua batang pada paha dan betis untuk membantu menyangga pengguna di kedua bagian tersebut.

2.7 Depth Camera

Depth camera merupakan salah satu jenis kamera yang dapat digunakan untuk menghasilkan citra dua dimensi yang menampilkan jarak suatu titik dari pusat kamera. Terdapat berbagai metode yang dapat digunakan untuk menghasilkan citra kedalaman (*depth image*) dari sebuah *depth camera*, salah satunya adalah dengan menangkap pantulan cahaya menggunakan sistem *time of flight* [41]. Seperti yang terlihat pada gambar 2.10, kamera dengan sistem tersebut akan memancarkan cahaya dan mengukur waktu tempuh dari pantulan cahaya tersebut ketika tertangkap kembali oleh kamera.



Gambar 2.10: Diagram cara kerja sistem *time of flight* [9].

Selain cara tersebut, citra kedalaman juga dapat diperoleh de-

ngan cara melakukan perhitungan triangulasi pada dua atau lebih citra yang dihasilkan oleh sistem multi-kamera. Jarak suatu titik pada kamera dengan sistem ini dapat dihitung dengan melakukan korespondensi pada citra-citra yang dihasilkan. Berbeda dengan metode sebelumnya, *depth camera* dengan metode ini memiliki keunggulan dimana estimasi yang didapatkan bersifat kurang lebih pasif, dimana estimasi tersebut tidak banyak dipengaruhi oleh kondisi pencahayaan yang ada pada lingkungan.

2.7.1 Kinect V2

Kinect merupakan seri perangkat *motion sensing input* yang dikembangkan oleh Microsoft dan pertama kali dirilis pada 2010. Perangkat ini terdiri atas sebuah kamera RGB, pemancar dan detektor cahaya inframerah yang memetakan kedalaman menggunakan perhitungan *time of flight*, sebuah mikrofon, beserta perangkat lunak dan kecerdasan buatan dari Microsoft yang memungkinkan perangkat ini untuk melakukan *gesture recognition*, *speech recognition*, dan *body skeletal detection* pada satu hingga empat orang secara *real-time*.



Gambar 2.11: Perangkat *depth camera* Kinect V2 [10].

Kinect V2 merupakan seri Kinect yang dirilis pada 15 Juli 2014 untuk sistem operasi Windows. Perangkat ini dikembangkan dari Kinect untuk XBox One dan merupakan pengganti bagi Kinect untuk Windows yang awal. Seperti yang terlihat pada gambar 2.11, Kinect V2 memiliki fitur yang sama dengan seri sebelumnya, yak-

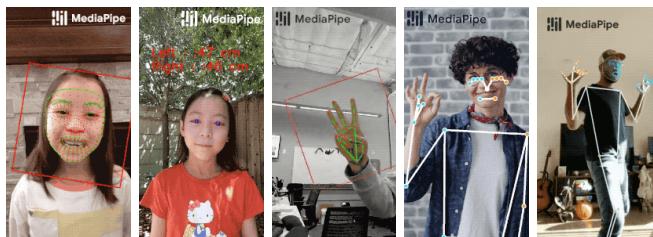
ni terdiri atas sebuah kamera RGB dan pemancar serta detektor cahaya inframerah yang memungkinkan perangkat ini untuk menghasilkan sebuah citra RGBD.

2.7.2 Freenect 2 Library (libfreenect2)

Freenect 2 library (libfreenect2) merupakan *library* yang berisi sebuah *driver* yang dapat digunakan untuk mengakses dan mengontrol perangkat Kinect V2 [42]. Library ini merupakan bagian dari OpenKinect, sebuah proyek yang memungkinkan penggunaan perangkat Kinect pada berbagai *platform* melalui berbagai *library* yang bersifat *open source*.

Library ini ditulis menggunakan bahasa pemrograman C++ dan dapat digunakan pada berbagai macam *platform* dari Windows, Linux, hingga MacOS. Fitur utama dari *library* ini adalah kemampuan untuk menerima data citra RGB, citra inframerah, serta *depth image* yang didapat dari perangkat Kinect V2. Selain itu, *library* ini juga bisa digunakan untuk melakukan registrasi citra RGB dan inframerah pada perangkat tersebut.

2.8 MediaPipe



Gambar 2.12: Contoh penggunaan MediaPipe untuk deteksi *face mesh*, *iris*, tangan, pose tubuh, dan *holistic* [11].

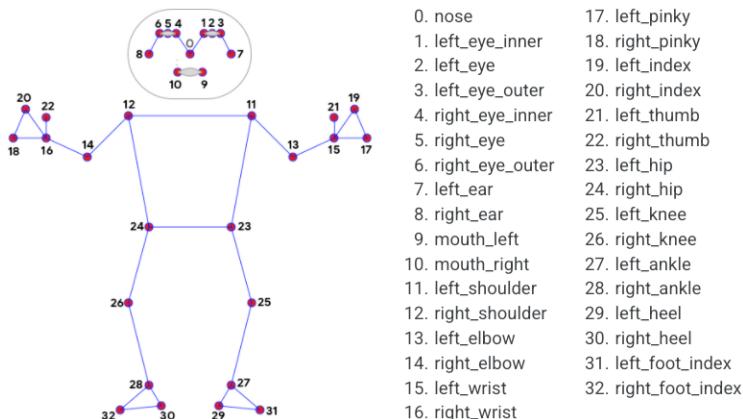
MediaPipe merupakan sebuah *machine learning solutions* untuk media *live* dan *streaming*. MediaPipe memiliki fitur *end-to-end acceleration* yang memungkinkan pemrosesan secara cepat pada perangkat dengan spesifikasi standar serta dapat di-build sekali dan di-deploy pada berbagai macam *platform* seperti Android, iOS,

desktop, web, hingga IoT. MediaPipe merupakan *framework* bersifat *open source*, menggunakan lisensi Apache 2.0, sehingga dapat dengan mudah dikembangkan sesuai dengan kebutuhan.

Framework ini dapat digunakan pada berbagai macam *platform* dan bahasa pemrograman, mulai dari Android, iOS, C++, Python, hingga JavaScript. Seperti yang terlihat pada gambar 2.12, *framework* ini dapat digunakan untuk melakukan berbagai macam hal mulai dari deteksi wajah, *face mesh*, *iris*, tangan, pose tubuh, *holistic*, dan lain sebagainya.

2.8.1 MediaPipe Pose

MediaPipe Pose merupakan salah satu *machine learning solutions* yang dimiliki oleh MediaPipe. MediaPipe Pose memungkinkan deteksi pose tubuh manusia melalui masukan video RGB menggunakan BlazePose [12]. Saat ini deteksi pose yang diproses menggunakan solusi ini mampu menghasilkan performa *real-time* pada perangkat mobile, desktop, dan web.



Gambar 2.13: Daftar *landmarks* yang ada pada BlazePose [12].

Pada MediaPipe Pose, *pipeline* yang dilakukan adalah pertama dengan menentukan *region-of-interest* (ROI) dari seseorang atau pose pada suatu *frame*. Kemudian dari ROI yang dihasilkan, *landmark* untuk setiap bagian dari pose akan dideteksi dan dilacak. Se-

perti yang terlihat pada gambar 2.13, terdapat total 33 *landmarks* yang dihasilkan oleh proses deteksi ini. *Landmarks* tersebut terdiri atas setiap bagian yang ada pada pose manusia. Dari proses tersebut, hasilnya adalah daftar *landmarks* yang terdeteksi beserta koordinat dari masing-masing *landmark* dalam ruang 3D.

2.9 SLAM

Simultaneous localization and mapping (SLAM) merupakan sebuah permasalahan komputer dalam menyusun peta dari lingkungan yang tidak diketahui sembari secara sekaligus melacak lokasi agen yang sedang memetakan lingkungan tersebut. Terdapat beberapa metode yang diketahui mampu memecahkan permasalahan tersebut, diantaranya yang umum digunakan adalah dengan menggunakan *particle filter*, *extended kalman filter*, *covariance intersection*, dan *GraphSLAM*. Algoritma SLAM pada umumnya berdasarkan konsep geometri komputasi dan visi komputer serta kebanyakan digunakan pada navigasi robot, pemetaan robotika, dan odometri untuk *virtual reality* maupun *augmented reality*.

2.9.1 RTAB-Map

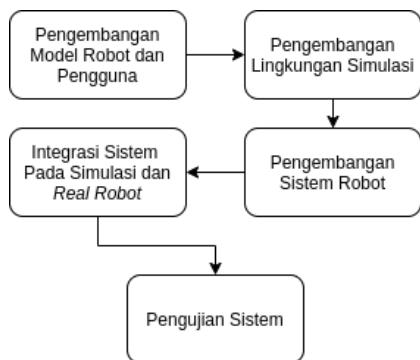
RTAB-Map merupakan sebuah *library* untuk *lidar* dan visual SLAM yang bersifat *open source* [43]. *Library* ini menggunakan pendekatan berbasis *graph* untuk memecahkan permasalahan SLAM menggunakan metode *loop closure detector* berbasis *incremental appearance*. *Loop closure detector* yang digunakan menggunakan pendekatan *bag-of-words* untuk menentukan kemungkinan sebuah citra berasal dari lokasi sebelumnya atau lokasi yang baru.

Ketika hipotesa *loop closure* diterima, sebuah *constraint* baru akan ditambahkan pada *graph* dari peta, kemudian sebuah *graph optimizer* akan meminimalisir *error* pada peta tersebut. Pendekatan *memory management* digunakan untuk membatasi jumlah lokasi yang digunakan untuk deteksi *loop closure* dan *graph optimization*. RTAB-Map dapat digunakan pada perangkat Kinect, kamera stereo, *3D lidar* untuk pemetaan dengan enam *degree of freedom* (DoF), atau *laser rangefinder* untuk pemetaan dengan tiga DoF.

BAB III

DESAIN DAN IMPLEMENTASI

Pada bab ini akan diuraikan desain dan implementasi dari sistem yang telah dibuat. Seperti yang terlihat pada gambar 3.1, desain dan implementasi yang akan diuraikan ini dimulai dari pengembangan model robot dan pengguna yang dibuat sebagai file dengan bentuk SDFormat. Pengembangan kemudian akan dilakukan untuk lingkungan simulasi yang terdiri dari sebuah lingkungan *outdoor* dan lingkungan *indoor*.



Gambar 3.1: Blok diagram dari alur pekerjaan.

Kemudian, sistem robot berbasis ROS 2 akan dikembangkan. Pengembangan sistem tersebut dilakukan secara terabstraksi yang mana nantinya akan terpisah menjadi 3 bagian. Sistem yang mengatur komponen yang ada di simulasi akan dibentuk sebagai *Gazebo Plugins* yang akan terintegrasi dengan model robot dan model pengguna. Sistem yang mengatur komponen yang ada di *real robot* akan dibentuk sebagai *controller node*. Dan terakhir, Sistem yang mengatur tindakan robot secara umum untuk simulasi dan *real robot* akan dibentuk sebagai *behavior node*.

Bagian akhir dari pekerjaan ini adalah pengujian untuk sistem yang telah dibuat. Pengujian nantinya akan terdiri dari beberapa bagian yang menguji berbagai aspek yang ada pada sistem yang telah dikembangkan. Bagian ini secara lebih lanjut akan dijelaskan pada bab 4.

3.1 Pengembangan Model Robot

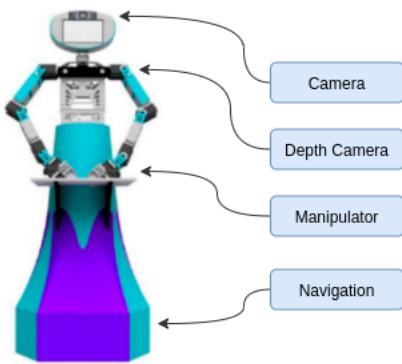
Pengembangan model robot untuk simulasi dilakukan dengan memperhatikan desain dari robot yang digunakan pada penelitian. Dari desain tersebut, model 3D yang ada perlu dipisah menyesuaikan bagian yang dapat bergerak bebas di robot seperti tangan dan kepala. Terakhir, file SDFormat (*simulation description format*) akan dibuat dengan isi yang menyesuaikan sensor, aktuator, dan setiap bagian 3D yang ada pada robot sehingga model tersebut bisa digunakan di simulator Gazebo.

3.1.1 Desain Robot *Dienen*

Robot yang digunakan pada penelitian ini adalah robot *Dienen*, yang merupakan kelanjutan dari robot *IRIS* [44][45] dengan penambahan bagian badan, lengan, dan kepala dari robot *ICHIRO* [46] di bagian atas robot. Desain seperti ini secara umum dikenal sebagai desain *mobile humanoid robot* [47], yang merupakan desain gabungan antara robot *mobile* dan robot *humanoid*. Seperti yang terlihat pada gambar 3.2, robot *Dienen* memiliki beberapa komponen yang bisa dikelompokkan menjadi 4 bagian terpisah, yakni sebuah kamera, *depth camera*, *manipulator*, dan navigasi.

Komponen kamera yang ada pada robot digunakan untuk menangkap citra yang nantinya bisa digunakan untuk melakukan proses visi komputer. Dengan adanya kamera ini, robot diharapkan mampu memperoleh masukan yang bisa berupa deteksi pengguna maupun deteksi objek yang ada di lingkungan.

Komponen *depth camera* yang ada pada robot dapat digunakan untuk menangkap dua macam citra, yakni citra berwarna dan citra kedalaman (*depth image*). Citra berwarna yang ditangkap memiliki format RGBA sedangkan citra kedalaman yang ditangkap memiliki format *grayscale* dengan nilai gelap yang semakin menunjukkan



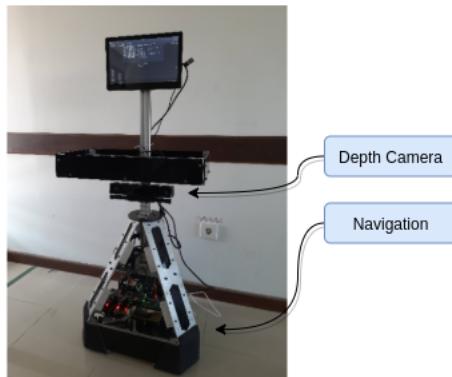
Gambar 3.2: Desain dan diagram komponen dari robot *Dienen*.

jarak yang lebih jauh dari kamera. *Depth Camera* ini nantinya bisa digunakan untuk melakukan pemetaan ruangan sekaligus meningkatkan keakuratan posisi dan orientasi dari robot menggunakan metode SLAM (*simultaneous localization and mapping*).

Komponen *manipulator* yang ada pada robot terdiri dari dua buah lengan dengan lima buah *joints* di setiap lengan. Lengan *manipulator* ini memiliki sebuah *gripper* di ujung lengan yang dapat digunakan untuk menggenggam sebuah objek. Dengan adanya lengan *manipulator* ini, robot diharapkan mampu memberikan tindakan *assistive* seperti membantu untuk membawakan maupun memberikan sebuah objek kepada pengguna.

Komponen navigasi yang ada pada robot merujuk pada komponen yang memungkinkan pergerakan yang ada pada robot. Komponen ini terdiri atas tiga buah *omnidirectional wheels* yang terpasang secara *holonomic* sehingga memungkinkan pergerakan ke semua arah, dua buah sensor *rotary encoder* yang dapat digunakan untuk memperkirakan posisi dari robot, dan sebuah sensor IMU (*inertial measurement unit*) yang dapat digunakan untuk memperkirakan orientasi dari robot. Dengan adanya komponen Navigasi ini, robot diharapkan mampu berpindah tempat secara mudah dari suatu posisi ke posisi yang lain.

3.1.2 Prototipe Robot *Dienan*



Gambar 3.3: Diagram komponen dari prototipe robot *Dienan*.

Prototipe robot *Dienan* ini adalah pengembangan sementara dari robot *Dienan*. Agar sesuai dengan *real robot* yang diujikan, spesifikasi sensor dan aktuator yang digunakan oleh model robot di simulasi akan disesuaikan dengan spesifikasi komponen yang ada di prototipe robot ini. Seperti yang terlihat pada gambar 3.3, prototipe robot ini saat ini baru terdiri atas komponen *depth camera* dan *navigation*. Untuk komponen kamera, sebagai alternatif sementara, sebuah *webcam* dapat digunakan dan dipasang di komputer robot melalui sambungan USB. Sedangkan untuk komponen *manipulator* untuk saat ini belum bisa digunakan.

Komponen *depth camera* yang digunakan di prototipe robot ini adalah Kinect V2 yang mampu menangkap citra berwarna dan citra kedalaman. Spesifikasi kedua citra yang ditangkap tersebut memiliki perbedaan pada sisi resolusi maksimal dan *field of view* (FoV), dimana citra berwarna bisa ditangkap hingga resolusi 1920 x 1080 dan FoV 84.1° x 53.8°, sedangkan citra kedalaman hanya bisa ditangkap pada resolusi 512 x 424 dan FoV 70.6° x 60°.

Komponen navigasi yang digunakan di prototipe robot ini adalah sekumpulan komponen elektronik yang digunakan di robot *IRIS* [44]. Komponen tersebut terdiri atas beberapa bagian seperti motor

DC, *motor driver*, IMU, dsb. yang terhubung pada sebuah *controller* berbasis STM32F4. Nantinya, *controller* tersebut akan terhubung dengan komputer utama dari robot yang berbasis Intel NUC melalui sambungan *ethernet* dan komunikasi UDP (*user datagram protocol*).

Seperti yang dijelaskan sebelumnya, kamera yang digunakan di robot adalah sebuah *webcam* yang dipasang ke komputer robot melalui sambungan USB. Kamera tersebut saat ini belum ditetapkan untuk digunakan di prototipe robot, namun untuk memenuhi keperluan pengujian, maka kamera tersebut dipilih untuk digunakan. kamera yang digunakan tersebut adalah Logitech C922 yang mampu digunakan untuk menangkap citra dengan resolusi hingga 1920 x 1080. Kamera tersebut dipilih karena kamera tersebut adalah kamera yang sama yang digunakan di robot *ICHIRO* [46] yang merupakan setengah bagian asal dari robot *Dienen* ini.

3.1.3 Struktur SDFormat Robot

Dari paparan desain yang dijelaskan di bagian 3.1.1 dan paparan spesifikasi dari prototipe robot yang dijelaskan di bagian 3.1.2, model robot untuk simulasi kemudian dibuat menggunakan model 3D dari desain yang ada dan disusun berdasarkan bagian dan komponen yang ada pada desain tersebut.

Penyusunan SDFormat dilakukan dengan memisahkan bagian yang dapat bergerak secara bebas di robot menjadi *link element* secara terpisah. Untuk setiap *link element* yang ada, beberapa *element* lain perlu ditambahkan ke *element* tersebut sehingga model yang dibuat bisa digunakan di simulasi. Seperti yang terlihat pada potongan kode 3.1, setiap *link element* yang dibuat akan terdiri dari beberapa *element* lain. *Element* tersebut dapat terdiri atas *pose element* yang digunakan untuk menentukan posisi serta orientasi dari bagian robot yang diukur dari titik pusat robot, *inertial element* yang digunakan untuk menentukan inersia dan massa dari bagian robot, *collision element* yang berisi *geometry element* yang digunakan untuk menentukan *bounding box* yang digunakan untuk perhitungan *physics* dari bagian robot, dan terakhir *visual element* yang digunakan untuk menentukan tampilan dari bagian robot melalui proses *rendering*.

Kode 3.1: *Link element* untuk bagian kepala robot.

```
1 <link name="head_link">
2   <pose>-0.012258 0 1.46717 0 0 0</pose>
3   <inertial>
4     <mass>1.0</mass>
5   </inertial>
6   <collision name="head_collision">
7     <geometry>
8       <mesh>
9         <uri>model://dienen_robot/collisions/←
10          head_collision.dae</uri>
11        </mesh>
12      </geometry>
13    </collision>
14    <visual name="head_visual">
15      <geometry>
16        <mesh>
17          <uri>model://dienen_robot/meshes/head.dae</uri>
18        </mesh>
19      </geometry>
20    </visual>
21 </link>
```

Agar terhubung satu sama lain, sebuah *joint element* perlu ditambahkan untuk menentukan hubungan antara dua *element*. Seperti yang terlihat pada potongan kode 3.2 dan 3.3, setiap *joint element* memiliki *type attribute* yang bisa bernilai *fixed* maupun *revolute*. *Joint element* dengan *type attribute* bernilai *fixed* digunakan untuk bagian yang tidak bisa bergerak bebas namun secara model terpisah seperti bagian bawah robot dan bagian atas robot, sedangkan *joint element* dengan *type attribute* bernilai *revolute* digunakan untuk bagian yang bisa bergerak berputar pada satu sumbu, seperti bagian *joint* yang ada di tangan.

Kode 3.2: *Joint element* yang menghubungkan bagian pundak kiri dan bagian badan atas robot.

```
1 <joint name="left_shoulder_joint" type="revolute">
2   <parent>upper_body_link</parent>
3   <child>left_shoulder_link</child>
4   <axis>
5     <xyz>0 -1 0</xyz>
6   </axis>
7 </joint>
```

Kode 3.3: *Joint element* yang menghubungkan bagian badan atas dan bagian badan bawah robot.

```
1 <joint name="upper_body_joint" type="fixed">
2   <parent>lower_body_link</parent>
3   <child>upper_body_link</child>
4 </joint>
```

Terakhir, penyusunan SDFormat dilakukan melalui penambahan sensor kamera dan *depth camera* yang digunakan pada robot. Sensor tersebut ditambahkan dengan menyematkan *sensor element* pada *link element* dari masing-masing bagian kamera dan *depth camera*. Seperti yang terlihat pada potongan kode 3.4 dan 3.5, sensor kamera dan *depth camera* memiliki jumlah dan jenis *child element* yang sama, hanya berbeda di beberapa nilai dari setiap *element* dan nilai *type attribute* yang digunakan. Pada kamera, *sensor element* yang digunakan memiliki *type attribute* bernilai *camera*, sedangkan pada *depth camera*, *sensor element* yang digunakan memiliki *type attribute* bernilai *depth*. Sisanya, nilai setiap *element* yang ada di kedua sensor tersebut didapatkan dari spesifikasi kamera dan *depth camera* yang digunakan di prototipe robot seperti yang dijelaskan di bagian 3.1.2.

Kode 3.4: *Sensor element* dari sensor kamera.

```
1 <link name="camera_link">
2 ...
3   <sensor name="camera" type="camera">
4     <pose>0.01 0 0 0 0 0</pose>
5     <update_rate>30</update_rate>
6     <camera>
7       <horizontal_fov>1.047198</horizontal_fov>
8       <image>
9         <width>1920</width>
10        <height>1080</height>
11        </image>
12        <clip>
13          <near>0.1</near>
14          <far>100</far>
15        </clip>
16      </camera>
17    </sensor>
18 </link>
```

Kode 3.5: *Sensor element* dari sensor *depth camera*.

```
1 <link name="depth_camera_link">
2 ...
3   <sensor name="depth_camera" type="depth">
4     <pose>0.01 0 0 0 0 0</pose>
5     <update_rate>30</update_rate>
6     <camera>
7       <horizontal_fov>1.047198</horizontal_fov>
8       <image>
9         <width>512</width>
10        <height>424</height>
11      </image>
12      <clip>
13        <near>0.5</near>
14        <far>4.5</far>
15      </clip>
16    </camera>
17  </sensor>
18 </link>
```

3.2 Pengembangan Model Pengguna

Model pengguna yang ada di simulasi akan dibuat menyerupai bentuk manusia yang dapat berinteraksi dengan robot. Pada kondisi pengujian menggunakan robot *real*, pengguna akan berinteraksi dengan robot melalui *smart assistive posture device* yang digunakan untuk mengirimkan data ke robot yang berupa koordinat posisi dan orientasi pengguna, masukan suara yang diberikan pengguna melalui *voice recognition*, dan kondisi postur kaki dari pengguna, apakah sedang berdiri atau duduk. Dengan adanya model pengguna ini, pengujian yang ada di simulasi dapat dilakukan terhadap model pengguna dengan perilaku yang sudah diprogram di awal, serta terhadap *real user* yang berinteraksi dengan model robot di simulasi melalui data yang dikirim dari *smart assistive posture device*.

Pada simulator Gazebo terdapat objek *Gazebo Actor* yang dapat digunakan untuk menampilkan model manusia yang dapat bergerak dan melakukan animasi seperti berjalan, duduk, dan lain sebagainya. Namun, objek tersebut sulit untuk diubah animasinya ketika simulasi sudah berjalan, sehingga kondisi perubahan postur

kaki dari pengguna tidak bisa ditampilkan di model yang ada di simulasi. Oleh karena itu, pada penelitian ini model pengguna dibuat dengan cara yang sama dengan model robot, yakni sebagai *Gazebo Model* dengan memisahkan setiap bagian pengguna yang dapat bergerak bebas menjadi *link element* dan kemudian akan saling terhubung menggunakan *joint element*.

3.2.1 Struktur SDFormat Pengguna

Struktur SDFormat pada model pengguna dibuat dengan cara yang sama dengan yang dilakukan pada model robot. Seperti yang terlihat pada potongan kode 3.6, model pengguna akan terdiri dari beberapa *link element* yang berisi setiap bagian yang dapat bergerak bebas pada model pengguna seperti lengan atas, paha, leher, dan lain sebagainya. Kemudian, *link element* tersebut akan terhubung satu sama lain menggunakan *joint element*. Setiap *link element* maupun *joint element* yang ada nantinya juga memiliki *child element* lain seperti *collision element*, *axis element*, dsb. Sesuai dengan yang dibutuhkan oleh *link element* dan *joint element*.

Kode 3.6: Struktur SDFormat dari model pengguna.

```
1 <?xml version="1.0" ?>
2 <sdf version="1.7">
3   <model name="dienen_human">
4
5     <link name="torso_link">
6       ...
7     </link>
8
9     <link name="left_upper_leg_link">
10    ...
11   </link>
12
13   ...
14
15   <joint name="left_upper_leg_joint" type="revolute">
16     ...
17   </joint>
18
19   ...
20
21   </model>
22 </sdf>
```

3.3 Pengembangan Lingkungan Simulasi

Pengembangan lingkungan simulasi dilakukan dengan menyusun setiap model yang telah dibuat pada *file SDFFormat*. Terdapat dua jenis lingkungan simulasi yang akan digunakan pada penelitian ini, yakni lingkungan simulasi *outdoor* yang hanya berisi robot tanpa adanya objek lain, dan lingkungan simulasi *indoor* yang membentuk suatu ruangan tertutup dan berisi robot serta berbagai macam objek lain. Keduanya dibuat sebagai *Gazebo World* yang dapat berisi beberapa *child element* lain dan beberapa *include file* dari model yang sudah ada.

3.3.1 Lingkungan Simulasi *Outdoor*

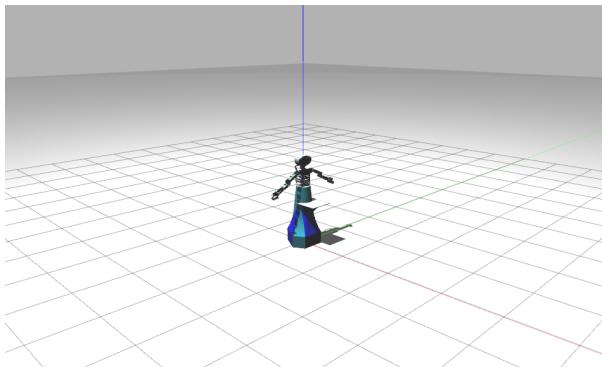
Lingkungan simulasi *outdoor* dibuat dengan menyusun *file SDFFormat* yang berisi lingkungan paling dasar yang dibutuhkan oleh robot. Lingkungan simulasi ini dibuat untuk memudahkan pengujian gerakan pada robot sehingga robot dapat bergerak secara bebas tanpa terhalang oleh *obstacle* lain.

Kode 3.7: Struktur SDFFormat dari lingkungan simulasi *outdoor*.

```
1 <?xml version="1.0"?>
2 <sdf version="1.7">
3   <world name="outdoor">
4
5     <light name="sun" type="directional">
6       <pose>0 0 10 0 -0 0</pose>
7       <direction>0 0.5 -0.9</direction>
8       ...
9     </light>
10
11     <include>
12       <uri>model://ground_plane</uri>
13     </include>
14
15     <include>
16       <pose>0.0 0 1.0 0 0 0</pose>
17       <uri>model://dienen_robot</uri>
18     </include>
19
20   </world>
21 </sdf>
```

Seperti yang terlihat pada potongan kode 3.7, lingkungan ini terdiri atas sebuah objek pencahayaan yang berupa *light element*, dan dua *include file* pada *path* model://ground_plane dan pada *path* model://dienen_robot. Kedua *include file* tersebut digunakan untuk memasukkan model *ground plane* yang sudah disediakan oleh Gazebo dan model robot yang telah dibuat sebelumnya di bagian 3.1. Di lingkungan ini, model *ground plane* diperlukan untuk memberikan *collision* pada dasar lingkungan sehingga model robot tidak jatuh terus menerus ke bawah karena pengaruh gravitasi di simulasi.

Setelah *file SDFormat* dibuat, hasil yang didapatkan setelah dijalankan pada simulator tampak seperti pada gambar 3.4. Sesuai dengan susunan yang diisi pada *file SDFormat*, tampak adanya model robot dengan objek *ground plane* yang ada di bawah, sedangkan objek pencahayaan tampak dengan adanya bayangan dari robot dan gradasi warna cerah pada objek *ground plane*.



Gambar 3.4: Tampilan lingkungan simulasi *outdoor* pada Gazebo.

3.3.2 Lingkungan Simulasi *Indoor*

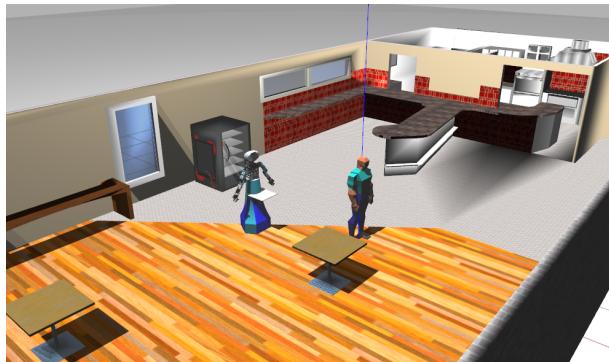
Lingkungan simulasi *indoor* dibuat dengan menyusun *file SDFormat* yang berisi ruangan tertutup dengan berbagai macam objek di dalamnya. Seperti yang terlihat pada potongan kode 3.8, lingkungan ini terdiri atas objek pencahayaan dan *include file* pada *path* model://cafe, model://cafe_table, model://dienen_robot, dan model://dienen_human. Dua *path* pertama merupakan *path* dari

model yang sudah disediakan oleh Gazebo, yang membentuk lingkungan kafe beserta perabotan yang ada di dalamnya, sedangkan dua *path* terakhir merupakan *path* dari model robot dan model pengguna yang sebelumnya sudah dibuat di bagian 3.1 dan 3.2.

Kode 3.8: Struktur SDFormat dari lingkungan simulasi *indoor*.

```
1 <?xml version="1.0"?>
2 <sdf version="1.7">
3   <world name="indoor">
4
5     <light name="sun" type="directional">
6       ...
7     </light>
8
9     <include>
10       <uri>model://cafe</uri>
11     </include>
12
13     <include>
14       <pose>0.5 -1.6 0.2 0 0 0</pose>
15       <uri>model://cafe_table</uri>
16     </include>
17
18     ...
19
20     <include>
21       <pose>-2 -1 1.0 0 0 0</pose>
22       <uri>model://dienen_robot</uri>
23     </include>
24
25     <include>
26       <pose>0 0 1.0 0 0 0</pose>
27       <uri>model://dienen_human</uri>
28     </include>
29
30   </world>
31 </sdf>
```

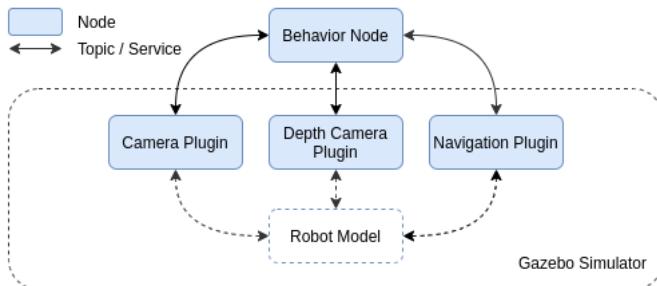
Setelah *file* SDFormat dibuat, hasil yang didapatkan setelah dijalankan pada simulator tampak seperti pada gambar 3.5. Sesuai dengan susunan yang diisi pada *file* SDFormat, tampak adanya model robot dan pengguna di dalam sebuah ruangan kafe dengan berbagai macam perabotan yang ada di dalamnya seperti meja, lemari, jendela, dan lain sebagainya.



Gambar 3.5: Tampilan lingkungan simulasi *indoor* pada Gazebo.

3.4 Integrasi *Plugin* untuk Simulasi

Agar sistem yang ada di simulasi dapat bekerja dan terabstraksi selayaknya sistem yang ada pada *real robot*, model yang ada di simulasi perlu diintegrasikan dengan *plugin* yang memungkinkan sensor dan aktuator yang ada di simulasi berkomunikasi dengan *node* lain menggunakan sistem komunikasi antar-proses ROS 2. Agar dapat bekerja dengan model yang telah dibuat, *plugin* tersebut nantinya perlu disematkan pada *file SDFFormat* dari model robot sebagai *child element* dalam bentuk *plugin element*.



Gambar 3.6: Diagram integrasi *plugin* untuk simulasi.

Seperti yang terlihat pada gambar 3.6, model robot akan diintegrasi dengan tiga buah *plugin*, *camera plugin* yang akan terabstraksi sebagai komponen kamera, *depth camera plugin* yang akan terabstraksi sebagai komponen *depth camera*, dan *navigation plugin* yang akan terabstraksi sebagai komponen navigasi. Ketiga plugin tersebut nantinya akan dibentuk sebagai *ROS 2 node* sehingga memungkinkan komunikasi dengan *node* lain, termasuk *behavior node* yang akan mengatur tingkah laku dari robot selama pengujian.

3.4.1 Navigation Plugin

Navigation plugin merupakan *Gazebo plugin* yang digunakan untuk mengabstraksi komponen navigasi di simulasi. *Plugin* ini ditulis dalam bahasa C++ dan dibuat menjadi *class* dengan nama *dienen_gazebo_plugins::NavigationPlugin*. Seperti yang terlihat pada potongan kode 3.9, *class* ini dibuat dengan menurunkan *class* *gazebo::ModelPlugin* sebagai *parent class* dari *class* ini sehingga memungkinkan *class* ini untuk mengakses maupun memanipulasi data yang ada pada *Gazebo model*.

Kode 3.9: *Class* dari *navigation plugin*.

```
1 #include <gazebo/common/Plugin.hh>
2 #include <rclcpp/rclcpp.hpp>
3 ...
4
5 namespace dienen_gazebo_plugins
6 {
7
8     class NavigationPlugin : public gazebo::ModelPlugin
9     {
10         public:
11             ...
12
13         private:
14             rclcpp::Node::SharedPtr node;
15
16             rclcpp::Publisher<Odometry>::SharedPtr odometry_pub;
17             rclcpp::Subscription<Twist>::SharedPtr twist_sub;
18
19             ...
20     };
21
22 } // namespace dienen_gazebo_plugins
```

Plugin ini digunakan untuk mengirimkan estimasi posisi dan orientasi dari robot sebagai odometri melalui *topic* /odom dan menerima masukan kecepatan (*twist*) melalui *topic* /cmd_vel. Odometri yang dikirimkan didapatkan dari posisi dan orientasi mutlak dari model robot yang ada di simulasi, sedangkan masukan kecepatan yang diterima akan diterjemahkan sebagai memberikan *force* kepada model robot yang mengakibatkan model tersebut bergerak ke arah yang sesuai dengan *force* yang diberikan.

Setelah *plugin* dibuat, file SDFormat dari model robot perlu diubah dengan menyematkan *plugin element* di file tersebut. Seperti yang terlihat pada potongan kode 3.10, sebuah *plugin element* ditambahkan di model robot dengan *name attribute* berupa nama *plugin* sekaligus nama *node* yang akan digunakan dan *filename attribute* yang merujuk pada nama *shared library* dari *plugin* yang telah dikompilasi.

Kode 3.10: Integrasi *navigation plugin* pada model robot.

```
1 <?xml version="1.0" ?>
2 <sdf version="1.7">
3   <model name="dienen_robot">
4
5     <plugin name="navigation_plugin" filename="←
6       libdienen_navigation_plugin.so" />
7
8     ...
9   </model>
10 </sdf>
```

3.4.2 Camera Plugin

Camera plugin merupakan *Gazebo plugin* yang digunakan untuk mengabstraksi komponen kamera di simulasi. *Plugin* yang digunakan ini akan mengirimkan data citra melalui *topic* /image_raw dan mengirimkan informasi kamera seperti resolusi citra, FoV, matriks proyeksi, dan lain sebagainya melalui *topic* /camera.info. *Plugin* ini didapatkan dari *package* gazebo_ros, sehingga yang perlu dilakukan hanyalah mengintegrasikan *plugin* tersebut ke model robot yang sudah dibuat di bagian 3.1.

Kode 3.11: Integrasi *camera plugin* pada model robot.

```
1 <link name="camera_link">
2 ...
3   <sensor name="camera" type="camera">
4     ...
5     <plugin name="camera_plugin" filename="←
6       libgazebo_ros_camera.so">
7       <camera_name>camera</camera_name>
8     </plugin>
9   </sensor>
9 </link>
```

Seperti yang terlihat pada potongan kode 3.11, sebuah *plugin element* ditambahkan sebagai *child element* dari *sensor element*. *Plugin* ini menggunakan *name attribute* untuk menentukan nama node dan *filename attribute* untuk menentukan *shared library* dari *plugin* yang telah dikompilasi. Dalam hal ini *plugin* yang digunakan adalah *plugin gazebo_ros_camera* dari *package gazebo_ros*. Sebagai tambahan, *plugin* ini menggunakan *camera name element* untuk menentukan *namespace* dari *topic* yang digunakan, sehingga dengan *camera name element* bernilai *camera* maka *topic* yang digunakan adalah /camera/image_raw dan /camera/camera_info.

3.4.3 Depth Camera Plugin

Depth camera plugin merupakan *Gazebo plugin* yang digunakan untuk mengabstraksi komponen *depth camera* di simulasi. Sama seperti yang digunakan pada *camera plugin*, *plugin* ini juga menggunakan *plugin gazebo_ros_camera* dari *package gazebo_ros*. Hanya saja *plugin* ini mengirimkan dua macam data citra, yakni citra berwarna dengan data citra yang dikirim melalui *topic* /image_raw dan informasi kamera yang dikirim melalui *topic* /camera_info, serta citra kedalaman (*depth image*) dengan data citra yang dikirim melalui *topic* /depth/image_raw dan informasi kamera yang dikirim melalui *topic* /depth/camera_info.

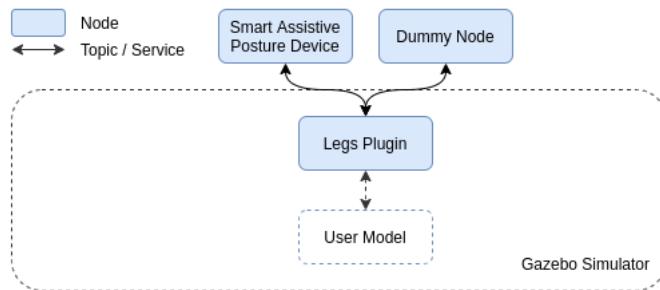
Seperti yang terlihat pada potongan kode 3.12, cara yang sama seperti yang ada pada *camera plugin* di bagian 3.4.2 juga dilakukan pada *plugin* ini. Perbedaannya terletak pada *name attribute* yang digunakan dan nilai dari *camera name* yang menentukan *namespace* dari *topic* yang digunakan.

Kode 3.12: Integrasi *depth camera plugin* pada model robot.

```
1 <link name="depth_camera_link">
2 ...
3   <sensor name="depth_camera" type="depth">
4     ...
5     <plugin name="depth_camera_plugin" filename="←
6       libgazebo_ros_camera.so">
7       <camera_name>depth_camera</camera_name>
8     </plugin>
9   </sensor>
9 </link>
```

3.4.4 Legs Plugin

Legs plugin merupakan *Gazebo plugin* yang digunakan untuk menghubungkan model pengguna dengan sistem komunikasi antarproses ROS 2. Seperti yang terlihat pada gambar 3.7, *plugin* ini dibuat agar model pengguna yang ada di simulasi bisa terabstraksi untuk diakses dan dimanipulasi oleh *smart assistive posture device* maupun *dummy node* yang mengirimkan data yang sama dengan yang dikirim oleh *smart assistive posture device*.



Gambar 3.7: Diagram integrasi *plugin* untuk model pengguna di simulasi.

Legs plugin memiliki dua kegunaan utama. Yang pertama adalah untuk mengubah posisi dan orientasi dari model pengguna sesuai dengan posisi dan orientasi yang diterima dari perhitungan *smart assistive posture device* maupun *dummy node*. Sedangkan yang ke-

dua adalah untuk mengubah posisi *joints* di kaki menjadi duduk jongkok maupun berdiri sesuai dengan nilai postur kaki yang dikirim oleh *smart assistive posture device* maupun *dummy node*.

Kode 3.13: Class dari *legs plugin*.

```
1 #include <beine_cpp/beine_cpp.hpp>
2 #include <gazebo/common/Plugin.hh>
3 #include <rclcpp/rclcpp.hpp>
4
5 ...
6
7 namespace beine_gazebo_plugins
8 {
9
10 ...
11
12 class LegsPlugin : public gazebo::ModelPlugin
13 {
14 public:
15     LegsPlugin();
16
17 ...
18
19 private:
20     rclcpp::Node::SharedPtr node;
21
22     std::shared_ptr<beine_cpp::LegsConsumer> ←
23         legs_consumer;
24
25 ...
26
27 } // namespace beine_gazebo_plugins
```

Sama seperti yang ada pada *navigation plugin*, *plugin* ini juga dibuat dengan menurunkan *class gazebo::ModelPlugin* sebagai *parent class* dari *class* ini. seperti yang terlihat pada potongan kode 3.13, *Plugin* ini menggunakan objek *beine_cpp::LegsConsumer* yang memudahkan *subscription* dari *topic* yang berhubungan dengan data yang dikirim oleh *smart assistive posture device*. Data tersebut berupa data posisi yang dikirim melalui *topic /position*, data orientasi yang dikirim melalui *topic /orientation*, data perintah suara yang dikirim melalui *topic /command*, dan data postur kaki yang dikirim melalui *topic /stance*.

Setelah *plugin* dibuat, file SDFormat dari model robot perlu diubah dengan menyematkan *element* plugin di file tersebut. Seperti yang terlihat pada potongan kode 3.14, *element* plugin disematkan sebagai *child element* dari *element* model. Pada *legs plugin*, beberapa *child element* lain perlu disematkan pada plugin tersebut, seperti *element joint_force_strength* dan *element joint_force_smoothness* yang menentukan bagaimana transisi postur kaki terjadi, serta *element left_hip_pitch_joint*, dan *element left_knee_pitch_joint*, dan lain sebagainya yang menentukan *element joint* yang akan diubah ketika terjadi transisi pada postur kaki pengguna.

Kode 3.14: Integrasi *legs plugin* pada model pengguna.

```
1 <?xml version="1.0" ?>
2 <sdf version="1.7">
3   <model name="dienen_human">
4
5     <plugin name="dienen_human_legs" filename="libbeine_legs_plugin.so">
6       <joint_force_strength>1000.0</joint_force_strength>
7       <joint_force_smoothness>0.2</joint_force_smoothness>
8       <left_hip_pitch_joint>left_upper_leg_joint</left_hip_pitch_joint>
9       <left_knee_pitch_joint>left_lower_leg_joint</left_knee_pitch_joint>
10      ...
11    </plugin>
12
13  ...
14
15  </model>
16 </sdf>
```

3.5 Pengembangan *Behavior Node*

Behavior node merupakan *ROS 2 node* yang dibuat untuk mengatur tindakan yang akan dilakukan oleh SARs berdasarkan data yang diterima dari *node* lain. Seperti yang dijelaskan di bagian 3.4, *behavior node* akan terhubung dengan *node* lain yang secara abstrak merepresentasikan komponen yang dimiliki oleh robot *Dienen* seperti penjelasan di bagian 3.1.

Behavior node dijalankan secara terpisah dari *node* yang mengatur komponen yang ada di robot seperti komponen kamera, *depth camera*, *manipulator*, dan navigasi. Terpisahnya *behavior node* ini dilakukan untuk membuat pengujian pada SARs bisa dilakukan menggunakan berbagai macam data yang diterima, baik itu data *real-time* yang berasal dari *real robot*, data yang direkam dari robot, maupun data yang ada di simulasi.

3.5.1 Move For Node

Move for node merupakan *behavior node* yang digunakan untuk memberikan perintah gerakan linier dan gerakan putar selama kurun waktu yang ditentukan. *Node* ini ditulis dalam bahasa pemrograman C++ dan akan mengirimkan data gerakan kecepatan pada topic /cmd_vel secara terus menerus hingga durasi waktu yang ditentukan sudah terpenuhi.

Kode 3.15: Program *move for node*.

```
1 #include <rclcpp/rclcpp.hpp>
2
3 ...
4
5 int main(int argc, char ** argv)
6 {
7     ...
8
9     rclcpp::init(argc, argv);
10
11    auto node = ...;
12    auto twist_publisher = ...;
13    auto update_process = ...;
14
15    // Update process
16    if (use_sim_time) {
17        auto clock_subscription = ...;
18        rclcpp::spin(node);
19    } else {
20        auto update_timer = ...;
21        rclcpp::spin(node);
22    }
23
24    ...
25 }
```

Seperti yang terlihat pada potongan kode 3.15, *node* ini menggunakan *package rclcpp* untuk mengakses sistem komunikasi ROS 2 pada program yang ditulis menggunakan bahasa pemrograman C++. *Node* ini memiliki sebuah *twist publisher* yang akan mengirimkan data perintah kecepatan gerak pada *topic /cmd_vel*. Agar durasi proses bisa tersinkronisasi dengan waktu yang ada di simulasi, *node* ini akan diproses sesuai dengan *clock topic* yang diterima dari simulasi. Selain itu, *node* ini akan diproses sesuai dengan *update timer* yang dimiliki *node* tersebut.

3.5.2 Patrol Position Node

Patrol position node merupakan *behavior node* yang digunakan untuk memberikan perintah pada robot untuk bergerak menuju setiap posisi yang ditentukan sampai posisi terakhir. *Node* ini ditulis dalam bahasa pemrograman C++ dan akan menerima data odometri melalui *topic /odom*, kemudian memproses target kecepatan yang diperlukan, dan mengirimnya melalui *topic /cmd_vel*.

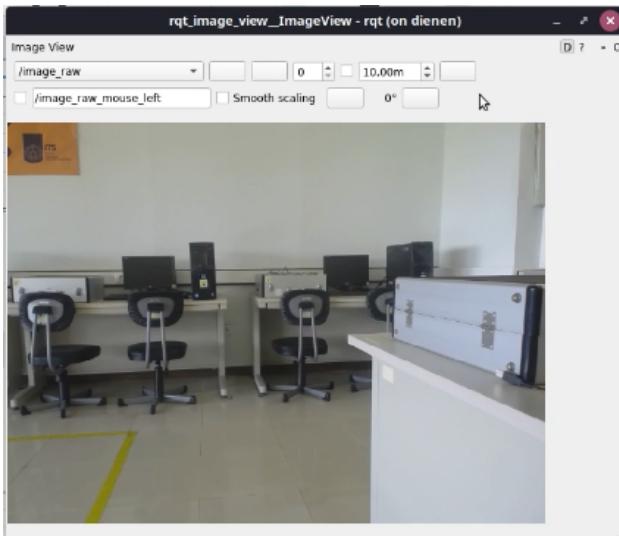
Kode 3.16: Program *patrol position node*.

```
1 #include <rclcpp/rclcpp.hpp>
2
3 ...
4
5 int main(int argc, char ** argv)
6 {
7     ...
8
9     rclcpp::init(argc, argv);
10
11    auto node = ...;
12    auto odometry_subscription = ...;
13    auto twist_publisher = ...;
14
15    auto update_timer = node->create_wall_timer(
16        10ms, [&](){
17            ...
18        });
19
20    rclcpp::spin(node);
21
22    ...
23 }
```

Seperti yang terlihat pada potongan kode 3.16, *node* ini menggunakan *package* rclcpp untuk mengakses sistem komunikasi ROS 2 pada program yang ditulis menggunakan bahasa pemrograman C++. *Node* ini memiliki sebuah *odometry subscription* untuk menerima data odometri pada *topic* /odom dan sebuah *twist publisher* yang akan mengirimkan data perintah kecepatan gerak pada *topic* /cmd_vel. Kemudian proses dari *node* ini akan dilakukan secara terus menerus sampai posisi terakhir tercapai melalui *callback* yang diberikan pada *update timer*.

3.5.3 Image Viewer Node

Image viewer node merupakan *behavior node* yang digunakan untuk melihat tampilan dari data citra yang dikirimkan pada suatu *topic*. *Image viewer node* yang digunakan ini merupakan program rqt_image_view yang berasal dari *package* rqt_image_view. *Node* tersebut akan menerima data citra pada suatu *topic* yang memiliki *interface* sensor_msgs/msg/Image, dan kemudian menampilkannya dalam bentuk GUI.



Gambar 3.8: Tampilan GUI dari *image viewer node*.

Seperti yang terlihat pada gambar 3.8, *Node* tersebut memiliki tampilan GUI yang dapat digunakan untuk memilih *topic* dari data citra yang akan ditampilkan. Jika *image viewer node* mendapatkan data yang berasal dari *topic* tersebut, *node* tersebut akan memproses data yang diterima serta menampilkannya dalam bentuk GUI. Selain itu *node* ini juga bisa digunakan untuk menyimpan data citra yang diterima tersebut dalam bentuk file gambar.

3.5.4 Legs Teleop Node

Legs teleop node merupakan *behavior node* yang digunakan untuk memungkinkan teleoperasi dari model pengguna yang ada di simulasi untuk mengubah posisi, orientasi, serta pose kaki dari model pengguna. *Node* ini merupakan implementasi dari *dummy node* untuk data yang berasal dari *smart assistive posture device* seperti yang terlihat pada gambar 3.7.

Kode 3.17: Program *legs teleop node*.

```
1 #include <beine_cpp/beine_cpp.hpp>
2 #include <rclcpp/rclcpp.hpp>
3
4 ...
5
6 int main(int argc, char ** argv)
7 {
8     ...
9
10    rclcpp::init(argc, argv);
11
12    auto node = ...;
13    auto legs_provider = ...;
14
15    auto update_timer = node->create_wall_timer(
16        100ms, [&](){
17            ...
18        });
19
20    rclcpp::spin(node);
21
22    rclcpp::shutdown();
23
24    return 0;
25 }
```

Node ini ditulis dalam bahasa pemrograman C++ dan akan mengirimkan data *dummy* dari posisi, orientasi, serta pose kaki pengguna. Seperti yang terlihat pada potongan kode 3.17, *node* ini menggunakan *package rclcpp* untuk mengakses sistem komunikasi ROS 2 pada program yang ditulis menggunakan bahasa pemrograman C++. *Node* ini menggunakan *legs provider* yang merupakan objek *class* yang berisi berbagai macam *publisher* untuk mengirimkan data yang berasal dari *smart assistive posture device*. Proses teleoperasi pada program ini sendiri dilakukan pada *update timer* yang nantinya akan memproses masukan *keyboard* dari pengguna dan mengubah data yang dikirimkan sesuai dengan nilai tersebut.

3.5.5 Pose Detector Node

Pose detector node merupakan *behavior node* yang digunakan untuk mendeteksi pose tubuh dari citra pengguna yang diterima menggunakan metode BlazePose [12]. *Node* ini ditulis dalam bahasa pemrograman Python dan akan menerima data citra melalui *topic* yang ditentukan, kemudian memproses dan mendeteksi data yang diterima, dan terakhir menampilkan hasilnya dalam bentuk GUI.

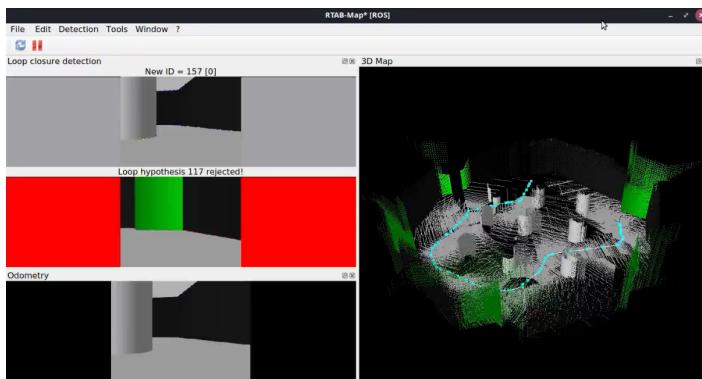
Kode 3.18: Program *pose detector node*.

```
1 import mediapipe as mp
2 from rclpy.node import Node
3
4 ...
5
6
7 class PoseDetector(Node):
8
9     def __init__(self, **kwargs):
10         super().__init__(...)
11
12         self.image_subscription = ...
13         self.pose = mp.solutions.pose.Pose(...)
14
15         ...
16
17     def image_callback(self, msg):
18         image = ...
19         results = self.pose.process(image)
20
21         ...
```

Seperti yang terlihat pada potongan kode 3.18, *node* ini menggunakan *package* `rclpy` untuk mengakses sistem komunikasi ROS 2 pada program yang ditulis menggunakan bahasa pemrograman Python. *Node* ini memiliki sebuah *image subscription* dan sebuah fungsi *image callback* yang digunakan untuk menerima data citra. Pada fungsi tersebut, *node* ini akan memproses data citra yang diterima dan mendeteksi pose tubuh pengguna menggunakan objek `mp.solutions.pose.Pose` yang berasal dari *library* MediaPipe [48].

3.5.6 RTAB-Map Node

RTAB-Map node merupakan *behavior node* yang digunakan untuk melakukan pemetaan dan lokalisasi menggunakan metode SLAM. *Node* yang digunakan ini merupakan program `rtabmap` yang berasal dari *package* `rtabmap_ros`. *Package* tersebut sendiri merupakan *wrapper* untuk ROS dari *library* RTAB-Map, sebuah library SLAM untuk kamera RGB-D, kamera stereo, dan sensor *lidar* berbasis *loop closure detector* [43].

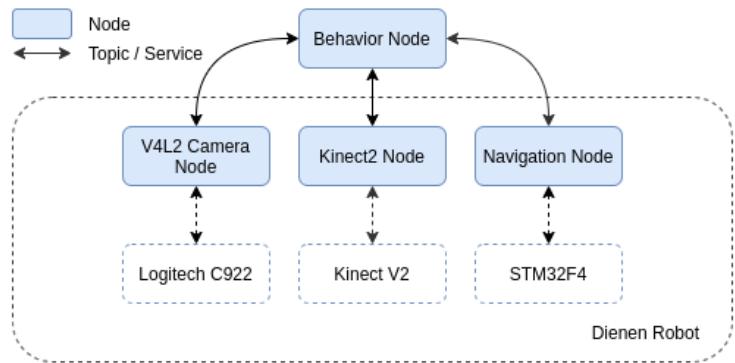


Gambar 3.9: Tampilan GUI dari *RTAB-Map node*.

Dalam melakukan pemetaan dan lokalisasi, *node* ini akan menerima data citra berwarna, data citra kedalaman (*depth image*), data informasi kamera, dan data odometri. Kemudian data yang diterima tersebut akan diproses menggunakan metode yang ada pada *library* RTAB-Map, dan kemudian akan menghasilkan peta lingku-

ngan dalam bentuk 3D serta hasil perkiraan lokalisasi. Seperti yang terlihat pada gambar 3.9, *node* ini memiliki tampilan GUI yang menampilkan data citra berwarna dan citra kedalaman, peta 3D yang dihasilkan dalam bentuk *point cloud*, serta lintasan dari posisi yang telah dilalui berdasarkan data odometri yang diterima.

3.6 Integrasi Sistem pada *Real Robot*



Gambar 3.10: Diagram integrasi sistem pada *real robot*.

Integrasi sistem pada *real robot* dapat dilakukan dengan mengganti *node* yang digunakan di simulasi dengan *node* yang mengakses komponen yang ada pada *real robot*. Seperti yang terlihat pada gambar 3.10, *behavior node* yang digunakan adalah *node* yang sama seperti yang ada pada gambar 3.6, perbedaannya adalah digantinya *camera plugin* dengan *V4L2 camera node*, *depth camera plugin* dengan *Kinect2 node*, dan *navigation plugin* dengan *navigation node*. Penggantian ini dapat dilakukan dengan mudah, karena seperti yang dijelaskan di bagian 3.5, *behavior node* mengakses setiap *node* yang mewakili komponen pada robot secara abstrak, yang mana ke-duanya dianggap *node* yang sama oleh *behavior node* terlepas dari bagaimana dan darimana data tersebut berasal, baik dari simulasi maupun dari *real robot*.

3.6.1 Navigation Node

Navigation node merupakan *node* yang digunakan untuk mengakses komponen navigasi yang ada pada robot *Dienan*. *Node* ini akan berkomunikasi dengan *controller* robot berbasis STM32F4 melalui sambungan *ethernet* dan menggunakan protokol UDP. *Node* ini ditulis dalam bahasa pemrograman C++ dan akan menerima data perintah kecepatan melalui *topic* /cmd_vel untuk kemudian diteruskan ke *controller* robot, serta menerima data odometri dari *controller* robot untuk kemudian diteruskan melalui *topic* /odom.

Kode 3.19: *Class* dari *navigation node*.

```
1 #include <rclcpp/rclcpp.hpp>
2 #include <tosshin/tosshin.hpp>
3
4 ...
5
6 class Navigation : public rclcpp::Node
7 {
8 private:
9     void listen_process();
10    void broadcast_process();
11
12    rclcpp::Subscription<tosshin::msg::Twist>::SharedPtr ←
13        twist_subscription;
14    rclcpp::Publisher<tosshin::msg::Odometry>::SharedPtr ←
15        odometry_publisher;
16
17    ...
18};
```

Seperti yang terlihat pada potongan kode 3.19, *node* ini menggunakan *package* rclcpp untuk mengakses sistem komunikasi ROS 2 pada program yang ditulis menggunakan bahasa pemrograman C++. *Node* ini memiliki sebuah *twist subscription* yang akan menerima data perintah kecepatan gerak melalui *topic* /cmd_vel dan sebuah *odometry publisher* yang akan mengirim data odometri melalui *topic* /odom. Bagian utama dari *node* ini terletak pada *update timer* yang nantinya akan memanggil fungsi *listen_process()* serta *broadcast_process()* secara terus menerus untuk bertukar data dengan *controller* robot menggunakan protokol UDP.

3.6.2 V4L2 Camera Node

V4L2 camera node merupakan *node* yang digunakan untuk mengakses komponen kamera yang dapat disambungkan pada komputer robot *Dienen* menggunakan sambungan USB. *Node* yang digunakan ini berasal dari *package* v4l2_camera, sebuah *ROS 2 package* yang memungkinkan akses kamera menggunakan sistem komunikasi yang ada pada ROS 2. *Package* tersebut menggunakan API Video4Linux (V4L) [49] untuk memungkinkan akses kamera pada perangkat Linux.

Node ini dapat dikonfigurasi secara dinamis ketika *node* sedang dijalankan, sehingga konfigurasi parameter seperti *device port* kamera, ukuran citra, dan lain sebagainya dapat dilakukan tanpa perlu menjalankan ulang *node* yang sudah dijalankan. Ketika *node* ini terkoneksi dengan perangkat kamera, *node* ini secara terus-menerus akan mengirimkan data citra melalui *topic* /image_raw dan data informasi kamera melalui *topic* /camera_info.

3.6.3 Kinect2 Node

Kinect2 node merupakan *node* yang digunakan untuk mengakses perangkat *depth camera* Kinect V2 yang dimiliki robot *Dienen*. *Node* ini ditulis dalam bahasa pemrograman C++ dan menggunakan *library* libfreenect2 untuk memungkinkan akses terhadap perangkat Kinect V2 [42]. *Node* ini akan mengirimkan sebuah data citra berwarna melalui *topic* /kinect2/image_raw, sebuah data citra kedalaman (*depth image*) melalui *topic* /kinect2/depth↔/image_raw, dan dua buah data informasi kamera masing-masing untuk citra berwarna dan kedalaman.

Seperti yang terlihat pada potongan kode 3.20, *node* ini menggunakan *package* rclcpp untuk mengakses sistem komunikasi ROS 2 dan objek libfreenect2 :: Freenect2 untuk mengakses perangkat Kinect V2. *Node* ini memiliki dua buah *image publisher* dan *camera info publisher* masing-masing untuk citra berwarna dan citra kedalaman. Bagian utama dari *node* ini terletak pada fungsi OnNewFrame() yang akan dipanggil ketika *node* menerima data citra dari perangkat Kinect V2, dan kemudian menyalurkan data citra tersebut kepada *topic* yang telah dijelaskan sebelumnya.

Kode 3.20: *Class* dari *Kinect2 node*.

```
1 #include <libfreenect2/libfreenect2.hpp>
2 #include <rclcpp/rclcpp.hpp>
3
4 ...
5
6 class Kinect2 : ...
7 {
8 public:
9     bool onNewFrame(...);
10
11 private:
12     rclcpp::Publisher<...>::SharedPtr rgb_image_publisher ←
13         ;
14     rclcpp::Publisher<...>::SharedPtr ←
15         rgb_camera_info_publisher;
16     rclcpp::Publisher<...>::SharedPtr ←
17         depth_image_publisher;
18     rclcpp::Publisher<...>::SharedPtr ←
19         depth_camera_info_publisher;
20
21     libfreenect2::Freenect2 freenect2;
22
23     ...
24 };

```

[Halaman ini sengaja dikosongkan]

BAB IV

HASIL DAN PENGUJIAN

Pada bab ini akan dipaparkan hasil pengujian serta analisis dari desain dan implementasi sistem yang telah dibuat sebelumnya di bab 3. Setiap pengujian yang dilakukan pada penelitian ini diujikan menggunakan model robot yang ada di simulasi dan menggunakan prototipe robot yang diuji secara *real*.

Pengujian yang dilakukan menggunakan model robot yang ada di simulasi dilakukan di lingkungan *outdoor* dan di lingkungan *indoor* seperti yang telah dibuat di bagian 3.3. Pengujian tersebut dilakukan di simulator Gazebo dengan menggunakan komputer dengan spesifikasi seperti yang dapat dilihat pada tabel 4.1. Sedangkan pengujian yang dilakukan menggunakan prototipe robot secara *real* dilakukan di lingkungan laboratorium AJ403 Teknik Komputer ITS serta menggunakan komputer robot dengan spesifikasi seperti yang dapat dilihat pada tabel 4.2.

Tabel 4.1: Spesifikasi komputer untuk menjalankan simulator.

OS	Ubuntu 20.04.2 LTS
CPU	Intel i3-8100 (4) @ 3.600GH
GPU	NVIDIA GeForce GTX 1050 Ti
RAM	7901 MiB

Tabel 4.2: Spesifikasi komputer yang ada pada prototipe robot.

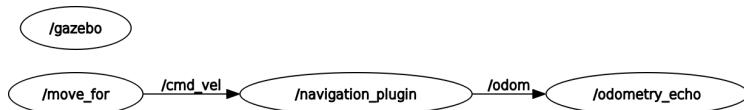
OS	Ubuntu 20.04.2 LTS
CPU	Intel i3-10110U (4) @ 4.100
GPU	Intel UHD Graphics
RAM	3648 MiB

4.1 Pengujian Gerakan

Pengujian gerakan terbagi menjadi dua bagian, yakni pengujian gerakan linier dan pengujian gerakan putar (*angular movement*). Pengujian gerakan ini dilakukan untuk menguji perintah gerakan yang dikirim melalui *topic* /cmd_vel serta untuk menguji estimasi posisi dan orientasi dari perhitungan odometri yang dapat diterima melalui *topic* /odom. Kedua pengujian ini melibatkan komponen navigasi yang ada pada robot dan *move for node* sebagai *behavior node* yang mengatur tingkah laku robot. Dalam hal ini, tingkah laku yang dilakukan adalah memberikan perintah gerakan pada robot selama selang waktu tertentu.

4.1.1 Pengujian Gerakan Linier dan Estimasi Posisi pada Robot di Simulasi

Pengujian gerakan linear dan estimasi posisi pada robot di simulasi dilakukan dengan cara menjalankan lingkungan *outdoor* pada simulator Gazebo, menjalankan *move for node* sebagai *behavior node* dari pengujian, dan menjalankan *odometry echo node* untuk menampilkan hasil estimasi posisi dari kalkulasi odometri robot. Seperti yang terlihat pada gambar 4.1, *node* /move_for akan mengirimkan *topic* /cmd_vel yang memerintahkan *node* /navigation_plugin untuk menggerakkan robot sesuai dengan data kecepatan yang ada di *topic* tersebut, setelah itu *node* /odometry_echo akan menerima *topic* /odom yang menunjukkan estimasi posisi dari perhitungan yang dilakukan oleh *node* /navigation_plugin.



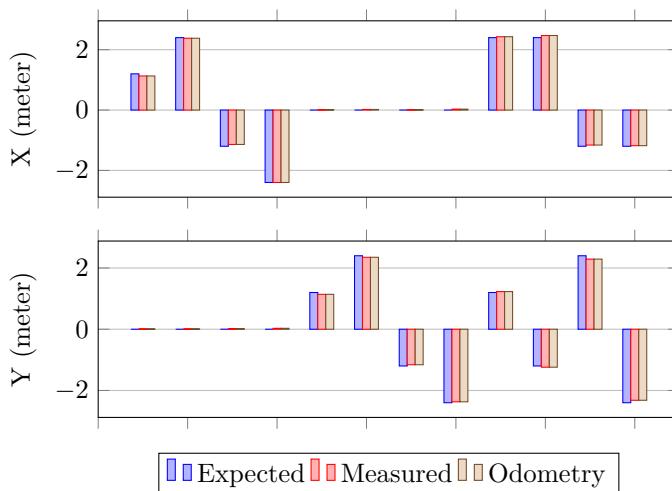
Gambar 4.1: Relasi antar-*node* dari pengujian gerakan linier dan estimasi posisi pada robot di simulasi.

Pengujian ini dilakukan dengan berbagai macam konfigurasi kecepatan X dan Y yang diperintahkan selama 3 detik. Hasil pengujian ini bisa dilihat pada tabel 4.3. Pada tabel tersebut, nilai yang ada di kolom *speed* adalah besar kecepatan yang diatur pada *topic* /cmd.vel, nilai yang ada di kolom *estimated* didapatkan

Tabel 4.3: Hasil estimasi posisi dari gerakan linier pada robot di simulasi selama 3 detik.

Speed		Expected			Measured		Odometry			
X (m/s)	Y (m/s)	X (m)	Y (m)	D (m)	X (m)	Y (m)	X (m)	Y (m)	E (m)	E (%)
0.4	0.0	1.20	0.00	1.20	1.13	0.01	1.13	0.01	0.04	3.1%
0.8	0.0	2.40	0.00	2.40	2.38	0.01	2.38	0.01	0.01	0.5%
-0.4	0.0	-1.20	0.00	1.20	-1.14	0.02	-1.14	0.02	0.03	2.8%
-0.8	0.0	-2.40	0.00	2.40	-2.40	0.03	-2.4	0.03	0.02	0.7%
0.0	0.4	0.00	1.20	1.20	0.01	1.14	0.01	1.14	0.03	2.7%
0.0	0.8	0.00	2.40	2.40	0.02	2.35	0.02	2.35	0.03	1.2%
0.0	-0.4	0.00	-1.20	1.20	0.01	-1.16	0.01	-1.16	0.02	1.9%
0.0	-0.8	0.00	-2.40	2.40	0.03	-2.37	0.03	-2.37	0.02	0.9%
0.8	0.4	2.40	1.20	2.68	2.43	1.23	2.43	1.23	0.02	0.9%
0.8	-0.4	2.40	-1.20	2.68	2.47	-1.24	2.47	-1.24	0.04	1.6%
-0.4	0.8	-1.20	2.40	2.68	-1.16	2.29	-1.16	2.29	0.06	2.2%
-0.4	-0.8	-1.20	-2.40	2.68	-1.18	-2.32	-1.18	-2.32	0.04	1.6%

dari perkalian besar kecepatan dengan durasi pengujian, nilai yang ada di kolom *measured* didapatkan dari koordinat model yang ada di simulasi, dan terakhir nilai yang ada di kolom *odometry* didapatkan dari data yang ada pada *topic /odom*. Subkolom D pada kolom *estimated* merupakan jarak tempuh dari posisi yang diharapkan, sedangkan subkolom E pada kolom *odometry* merupakan jarak rata-rata *error* pada nilai odometri terhadap nilai yang diharapkan (*estimated*) dan nilai pengukuran (*measured*) serta persentasenya jika dibandingkan dengan jarak tempuh yang diharapkan.



Gambar 4.2: Grafik estimasi posisi dari gerakan linier pada robot di simulasi.

Dari data yang dihasilkan oleh pengujian ini dapat diketahui bahwa gerakan yang diperintahkan kepada robot memiliki *error* jarak sebesar 1-6 cm dengan persentase rata-rata sebesar 1.7% dari jarak tempuh yang diharapkan. Lebih lanjut, ketika hasil tersebut ditampilkan sebagai grafik, seperti yang terlihat pada gambar 4.2, hasil yang didapatkan cenderung memiliki posisi dengan arah positif dan negatif yang sesuai dengan nilai yang diharapkan (*expected*) dan nilai pengukuran (*measured*).

4.1.2 Pengujian Gerakan Linier dan Estimasi Posisi pada *Real Robot*

Pengujian gerakan linier dan estimasi posisi pada *real robot* dilakukan dengan cara yang sama dengan yang dilakukan di simulasi seperti yang ada di bagian 4.1.1. Perbedaannya, pada pengujian ini komponen navigasi yang sebelumnya berasal dari simulasi digantikan dengan *navigation node* yang mengakses perangkat yang ada pada *real robot*. Seperti yang terlihat pada gambar 4.3, *node /move_for* akan mengirimkan *topic /cmd_vel* dengan fungsi yang sama ke *node /navigation*, setelah itu *node /odometry_echo* akan menerima *topic /odom* yang berasal dari *node /navigation*.



Gambar 4.3: Relasi antar-*node* dari pengujian gerakan linier dan estimasi posisi pada *real robot*.

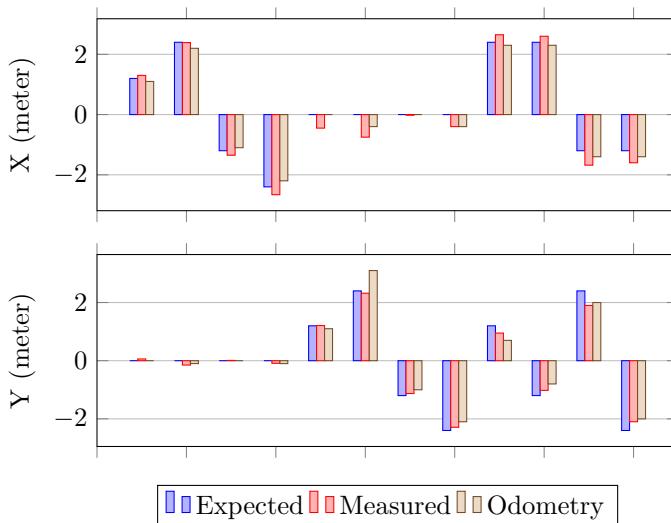
Pengujian pada *real robot* ini juga dilakukan dengan berbagai macam konfigurasi kecepatan X dan Y yang diperintahkan selama 3 detik. Hasil pengujian ini bisa dilihat pada tabel 4.4. Pada tabel tersebut, nilai yang ada di kolom *speed* adalah besar kecepatan yang diatur pada *topic /cmd_vel*, nilai yang ada di kolom *estimated* didapatkan dari perkalian besar kecepatan dengan durasi pengujian, nilai yang ada di kolom *measured* didapatkan dari pengukuran perpindahan *real robot* menggunakan meter ukur, dan terakhir nilai yang ada di kolom *odometry* didapatkan dari data yang ada di *topic /odom*. Sama seperti pada tabel 4.3, subkolom D pada kolom *estimated* merupakan jarak tempuh dari posisi yang diharapkan, sedangkan subkolom E pada kolom *odometry* merupakan jarak rata-rata *error* pada nilai odometri terhadap nilai yang diharapkan (*estimated*) dan nilai pengukuran (*measured*) serta persentasenya jika dibandingkan dengan jarak tempuh yang diharapkan.

Dari data yang dihasilkan oleh pengujian ini dapat diketahui bahwa gerakan yang diperintahkan kepada robot memiliki *error* jarak sebesar 15-83 cm dengan persentase rata-rata sebesar 16.3% dari jarak tempuh yang diharapkan. Nilai yang dihasilkan tersebut memiliki *error* yang relatif besar, hal ini disebabkan karena ketika

Tabel 4.4: Hasil estimasi posisi dari gerakan linier pada *real robot* selama 3 detik.

Speed		Expected			Measured		Odometry			
X (m/s)	Y (m/s)	X (m)	Y (m)	D (m)	X (m)	Y (m)	X (m)	Y (m)	E (m)	E (%)
0.4	0.0	1.20	0.00	1.20	1.30	0.06	1.10	0.00	0.15	12.9%
0.8	0.0	2.40	0.00	2.40	2.39	-0.15	2.20	-0.10	0.21	8.8%
-0.4	0.0	-1.20	0.00	1.20	-1.35	0.01	-1.10	0.00	0.18	14.6%
-0.8	0.0	-2.40	0.00	2.40	-2.66	-0.09	-2.20	-0.10	0.34	14.2%
0.0	0.4	0.00	1.20	1.20	-0.45	1.21	0.00	1.10	0.28	23.5%
0.0	0.8	0.00	2.40	2.40	-0.75	2.32	-0.40	3.10	0.83	34.6%
0.0	-0.4	0.00	-1.20	1.20	-0.02	-1.13	0.00	-1.00	0.17	13.8%
0.0	-0.8	0.00	-2.40	2.40	-0.40	-2.29	-0.40	-2.10	0.35	14.4%
0.8	0.4	2.40	1.20	2.68	2.65	0.95	2.30	0.70	0.47	17.5%
0.8	-0.4	2.40	-1.20	2.68	2.60	-1.02	2.30	-0.80	0.39	14.6%
-0.4	0.8	-1.20	2.40	2.68	-1.68	1.90	-1.40	2.00	0.37	13.9%
-0.4	-0.8	-1.20	-2.40	2.68	-1.60	-2.10	-1.40	-2.00	0.34	12.5%

robot bergerak dan berhenti, terdapat *slip* yang menyebabkan posisi robot bergeser jauh dari posisi yang diharapkan. Namun terlepas dari besarnya *error* tersebut, ketika hasil yang didapatkan ditampilkan sebagai grafik seperti yang terlihat pada gambar 4.4, seperti kesimpulan pada pengujian sebelumnya, hasil yang didapatkan juga cenderung memiliki posisi dengan arah positif dan negatif yang sesuai dengan nilai yang diharapkan (*expected*) dan nilai pengukuran (*measured*) walaupun dengan beberapa kesalahan yang disebabkan oleh *slip* seperti yang telah dijelaskan sebelumnya.



Gambar 4.4: Grafik estimasi posisi dari gerakan linier pada *real robot*.

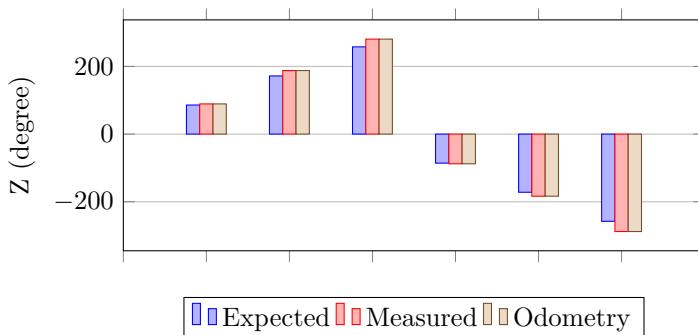
4.1.3 Pengujian Gerakan Putar dan Estimasi Orientasi pada Robot di Simulasi

Pengujian gerakan putar dan estimasi orientasi pada robot di simulasi dilakukan dengan menjalankan *node* yang sama seperti yang ada pada pengujian gerakan linier di bagian 4.1.1. Perbedaannya, pengujian ini dilakukan dengan menggunakan beberapa nilai kecepatan putar di sumbu Z selama 3 detik.

Tabel 4.5: Hasil estimasi orientasi dari gerakan putar pada robot di simulasi selama 3 detik.

Speed	Expected	Measured	Odometry		
			Z (deg)	E (deg)	E (%)
0.5	85.9	89.2	89.2	1.6	1.9%
1.0	171.9	187.8	187.8	8.0	4.6%
1.5	257.8	280.7	280.7	11.4	4.4%
-0.5	-85.9	-87.8	-87.8	0.9	1.1%
-1.0	-171.9	-183.6	-183.6	5.9	3.4%
-1.5	-257.8	-288.0	-288.0	15.1	5.9%

Hasil pengujian ini bisa dilihat pada tabel 4.5. Pada tabel tersebut, nilai yang ada di kolom *speed* adalah besar kecepatan yang diatur pada *topic /cmd/vel*, nilai yang ada di kolom *estimated* didapatkan dari perkalian besar kecepatan putar dengan durasi pengujian, nilai yang ada di kolom *measured* didapatkan dari orientasi model yang ada di simulasi, dan nilai yang ada di kolom *odometry* didapatkan dari data orientasi yang ada pada *topic /odom*. Subkolom E pada kolom *odometry* merupakan rata-rata *error* pada nilai odometri terhadap nilai yang diharapkan (*estimated*) dan nilai pengukuran (*measured*) serta persentasenya jika dibandingkan dengan nilai sudut yang diharapkan.



Gambar 4.5: Grafik estimasi orientasi dari gerakan putar pada robot di simulasi.

Dari data yang dihasilkan oleh pengujian ini dapat diketahui bahwa gerakan putar yang diperintahkan kepada robot memiliki *error* jarak sebesar 0.9-15.1 derajat dengan persentase rata-rata sebesar 3.5% dari nilai sudut yang diharapkan. Lebih lanjut, ketika hasil tersebut ditampilkan sebagai grafik, seperti yang terlihat pada gambar 4.5, hasil yang didapatkan cenderung memiliki sudut dengan arah positif dan negatif yang sesuai dengan nilai yang diharapkan (*expected*) dan nilai pengukuran (*measured*).

4.1.4 Pengujian Gerakan Putar dan Estimasi Orientasi pada *Real Robot*

Sama seperti pada pengujian simulasi di bagian sebelumnya, Pengujian gerakan putar dan estimasi orientasi pada *real robot* juga dilakukan dengan menjalankan *node* yang sama seperti yang ada pada pengujian gerakan linier di bagian 4.1.2. Perbedaannya, pengujian ini dilakukan dengan menggunakan beberapa nilai kecepatan putar di sumbu Z selama 3 detik.

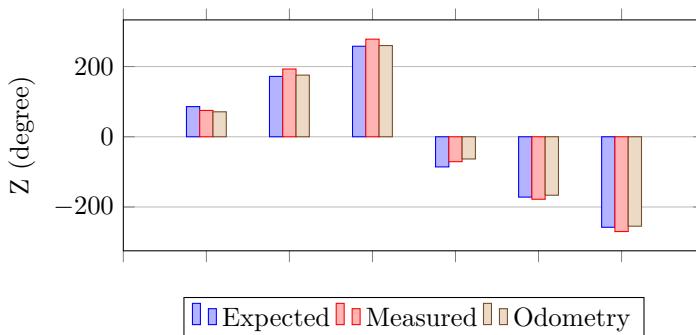
Tabel 4.6: Hasil estimasi orientasi dari gerakan putar pada *real robot* selama 3 detik.

Speed	Expected	Measured	Odometry		
			Z (deg)	E (deg)	E (%)
0.5	85.9	75.0	71.1	9.3	10.9%
1.0	171.9	193.0	175.6	10.6	6.1%
1.5	257.8	278.0	259.8	10.1	3.9%
-0.5	-85.9	-71.0	-63.3	15.2	17.7%
-1.0	-171.9	-178.0	-166.5	8.5	4.9%
-1.5	-257.8	-270.0	-254.7	9.2	3.6%

Hasil pengujian ini bisa dilihat pada tabel 4.6. Pada tabel tersebut, nilai yang ada di kolom *speed* adalah besar kecepatan yang diatur pada *topic /cmd/vel*, nilai yang ada di kolom *estimated* didapatkan dari perkalian besar kecepatan putar dengan durasi pengujian, nilai yang ada di kolom *measured* didapatkan dari pengukuran menggunakan kompas yang dipasang pada robot, dan terakhir nilai yang ada di kolom *odometry* didapatkan dari data orientasi yang

ada pada *topic* /odom. Sama seperti pada tabel 4.5, Subkolom E pada kolom *odometry* merupakan rata-rata *error* pada nilai odometri terhadap nilai yang diharapkan (*estimated*) dan nilai pengukuran (*measured*) serta persentasenya jika dibandingkan dengan nilai sudut yang diharapkan.

Dari data yang dihasilkan oleh pengujian ini dapat diketahui bahwa gerakan putar yang diperintahkan kepada robot memiliki *error* jarak sebesar 8.5-15.2 derajat dengan persentase rata-rata sebesar 7.8% dari nilai sudut yang diharapkan. Lebih lanjut, ketika hasil yang didapatkan ditampilkan sebagai grafik seperti yang terlihat pada gambar 4.5, seperti kesimpulan pada pengujian sebelumnya, hasil yang didapatkan cenderung memiliki sudut dengan arah positif dan negatif yang sesuai dengan nilai yang diharapkan (*expected*) dan nilai pengukuran (*measured*).



Gambar 4.6: Grafik estimasi orientasi dari gerakan putar pada *real robot*.

4.2 Pengujian Citra Kamera

Pengujian citra kamera terbagi menjadi dua bagian, yakni pengujian pengiriman citra kamera di perangkat yang sama dan di antara perangkat yang berbeda. Pengujian pengiriman citra ini dilakukan untuk menguji data citra gambar yang diterima melalui *topic* /raw_image serta untuk menguji kemampuan ROS 2 dalam mengirim data citra dengan ukuran besar secara *real-time*. Kedua

pengujian ini melibatkan komponen kamera yang ada pada robot, *image viewer node* untuk melihat tampilan data citra yang diterima, dan *command-line \$ ros2 topic* yang digunakan untuk mengukur estimasi *delay* dan frekuensi dari data citra yang diterima.

4.2.1 Pengujian Pengiriman Citra Kamera pada Robot di Simulasi



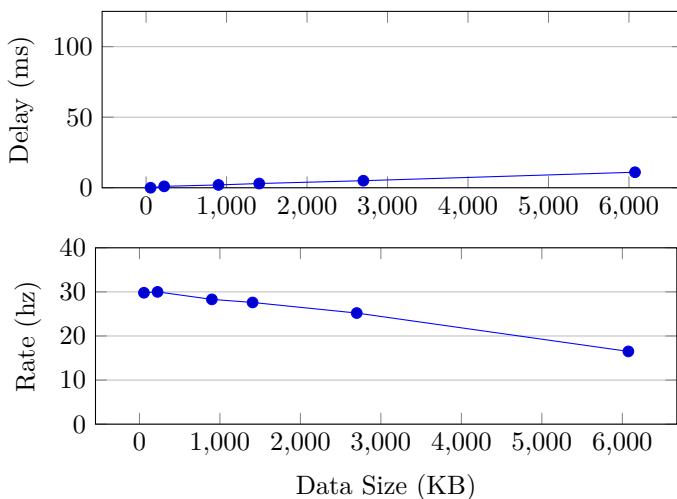
Gambar 4.7: Relasi antar-*node* dari pengujian pengiriman citra kamera di simulasi.

Pengujian pengiriman citra kamera pada robot di simulasi dilakukan dengan cara menjalankan lingkungan *indoor* pada simulator gazebo, menjalankan *image viewer node* untuk melihat hasil pengiriman gambar, serta menjalankan *command-line \$ ros2 topic* \leftarrow *delay* dan \leftarrow *ros2 topic hz* untuk mengukur *delay* serta frekuensi dari data yang dikirim. Seperti yang terlihat pada gambar 4.7, *node /camera_plugin* akan mengirimkan *topic /camera/image_raw* yang berisi citra kamera, setelah itu *node /image_viewer* akan menerima citra tersebut dan menampilkannya dalam bentuk GUI.

Tabel 4.7: Hasil *delay* dan frekuensi dari pengiriman citra kamera pada robot di simulasi.

Resolution			Delay	Rate	
Width	Height	Size (KB)	ms	hz	percent
160	120	56	0.0	29.8	99.5%
320	240	225	1.0	30.0	100.0%
640	480	900	2.0	28.3	94.4%
800	600	1406	3.0	27.6	92.0%
1280	720	2700	5.0	25.2	84.0%
1920	1080	6075	11.0	16.5	55.1%

Pengujian ini dilakukan dengan berbagai macam konfigurasi resolusi citra yang dikirim. Hasil pengujian ini bisa dilihat pada tabel 4.7. Pada tabel tersebut *width* dan *height* merupakan resolusi citra yang dikirimkan, *size* merupakan hasil perkalian resolusi dengan jumlah *channel* (dalam hal ini 4 untuk citra RGBA), *delay* merupakan selang waktu yang dibutuhkan sebelum citra sampai ke penerima, dan *rate* merupakan frekuensi pengiriman citra.



Gambar 4.8: Grafik *delay* dan frekuensi dari pengiriman citra pada robot di simulasi.

Dari data yang dihasilkan oleh pengujian ini dapat diketahui bahwa data yang dikirimkan memiliki *delay* sebesar 0-11 ms serta frekuensi sebesar 16.5-29.8 hz. Lebih lanjut, ketika hasil yang didapatkan ditampilkan sebagai grafik seperti yang terlihat pada gambar 4.8, hasil yang didapatkan menunjukkan bahwa semakin besar ukuran data yang dikirimkan, maka *delay* yang dihasilkan akan cenderung naik dan frekuensi yang dihasilkan akan cenderung turun. Walaupun cenderung naik, pada pengiriman dengan data

ukuran terbesar, nilai *delay* yang dihasilkan tersebut terhitung rendah, hanya sebesar 11 ms. Sedangkan untuk frekuensi, hingga batas resolusi 800 x 600, pengiriman citra yang dilakukan bisa menghasilkan frekuensi diatas 90% dari nilai FPS yang dimiliki kamera yang ada di simulasi.

4.2.2 Pengujian Pengiriman Citra Kamera pada *Real Robot*



Gambar 4.9: Relasi antar-*node* dari pengujian pengiriman citra kamera pada *real robot*.

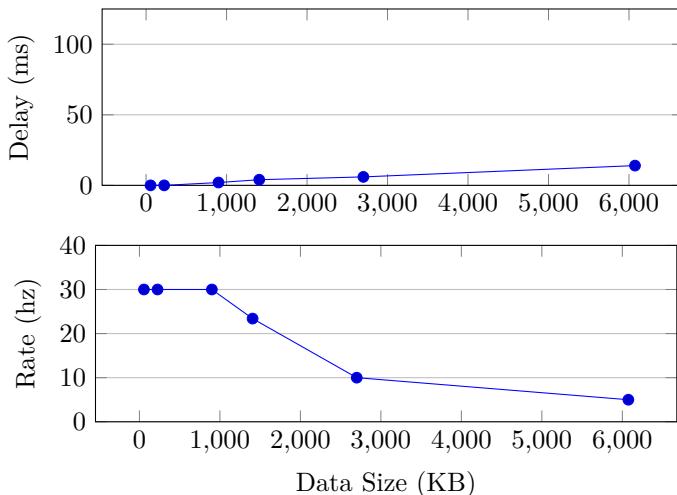
Sama seperti pengujian sebelumnya yang ada di bagian 4.2.1, pengujian ini juga dilakukan dengan menjalankan *image viewer node* dan *command-line \$ ros2 topic*. Hanya saja, sebagai ganti dari *node /camera.plugin* yang ada di simulasi, data citra kamera yang dikirim akan berasal dari *V4L2 camera node*. Seperti yang dapat dilihat pada gambar 4.9, *node /v4l2_camera* akan mengirimkan *topic /image_raw* yang berisi citra kamera, setelah itu *node /image_viewer* akan menerima citra tersebut dan menampilkannya dalam bentuk GUI.

Tabel 4.8: Hasil *delay* dan frekuensi dari pengiriman citra kamera pada *real robot*.

Resolution			Delay	Rate	
Width	Height	Size (KB)	ms	hz	percent
160	120	56	0.0	30.0	100.0%
320	240	225	0.0	30.0	100.1%
640	480	900	2.0	30.0	100.1%
800	600	1406	4.0	23.4	77.9%
1280	720	2700	6.0	10.0	33.3%
1920	1080	6075	14.0	5.0	16.7%

Pengujian ini dilakukan dengan berbagai macam konfigurasi re-

solusi citra yang dikirim. Hasil pengujian ini bisa dilihat pada tabel 4.7. Sama seperti pengujian yang ada di simulasi, pada tabel tersebut *width* dan *height* merupakan resolusi citra yang dikirimkan, *size* merupakan hasil perkalian resolusi dengan jumlah *channel*, *delay* merupakan selang waktu yang dibutuhkan sebelum citra sampai ke penerima, dan *rate* merupakan frekuensi pengiriman citra.



Gambar 4.10: Grafik *delay* dan frekuensi dari pengiriman citra pada *real robot*.

Dari data yang dihasilkan oleh pengujian ini dapat diketahui bahwa data yang dikirimkan memiliki *delay* sebesar 0-14 ms serta frekuensi sebesar 5.0-30.0 hz. Lebih lanjut, ketika hasil yang didapatkan ditampilkan sebagai grafik seperti yang terlihat pada gambar 4.10, seperti kesimpulan pada pengujian sebelumnya, hasil yang didapatkan menunjukkan bahwa semakin besar ukuran data yang dikirimkan, maka *delay* yang dihasilkan akan cenderung naik dan frekuensi yang dihasilkan akan cenderung turun. Walaupun cenderung naik, pada pengiriman dengan data ukuran terbesar, nilai *delay* yang dihasilkan tersebut terhitung rendah, hanya sebesar 14 ms. Sedangkan untuk frekuensi, hingga batas resolusi 640 x 480,

pengiriman citra yang dilakukan bisa menghasilkan frekuensi diatas 90% dari nilai FPS yang dimiliki kamera pada *real robot*.

4.2.3 Pengujian Pengiriman Citra Kamera Antar-perangkat pada Robot di Simulasi

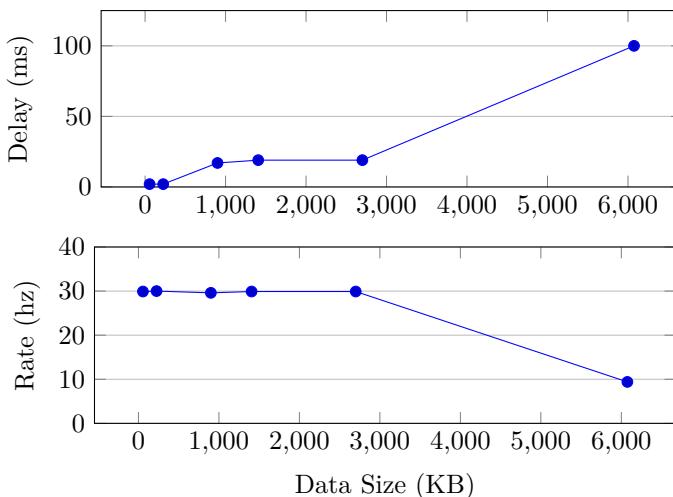
Pengujian pengiriman citra kamera antar-perangkat pada robot di simulasi memiliki kesamaan proses dengan yang dilakukan di pengujian pada sesama perangkat yang ada di bagian 4.2.1. Hanya saja, di pengujian ini, *image viewer node* dan *command-line \$ \leftarrow ros2 topic* dijalankan di perangkat berbeda yang terhubung dengan perangkat yang menjalankan lingkungan simulasi menggunakan jaringan *ethernet*. Perangkat lain yang digunakan di pengujian ini adalah sebuah komputer Intel NUC dengan spesifikasi yang sama seperti yang dimiliki prototipe robot di tabel 4.2.

Tabel 4.9: Hasil *delay* dan frekuensi dari pengiriman citra kamera antar-perangkat pada robot di simulasi.

Resolution			Delay	Rate	
Width	Height	Size (KB)	ms	hz	percent
160	120	56	2.0	29.9	99.7%
320	240	225	2.0	30.0	100.0%
640	480	900	17.0	29.6	98.6%
800	600	1406	19.0	29.9	99.8%
1280	720	2700	19.0	29.9	99.6%
1920	1080	6075	100.0	9.4	31.5%

Sama seperti pengujian pada sesama perangkat, pengujian ini juga dilakukan dengan berbagai macam konfigurasi resolusi citra yang dikirim. Hasil pengujian ini bisa dilihat pada tabel 4.9. Pada tabel tersebut *width* dan *height* merupakan resolusi citra yang dikirimkan, *size* merupakan ukuran data yang dikirim, *delay* merupakan selang waktu yang dibutuhkan untuk sampai ke perangkat penerima, dan *rate* merupakan frekuensi pengiriman citra.

Dari data yang dihasilkan oleh pengujian ini dapat diketahui bahwa data yang dikirimkan memiliki *delay* sebesar 2-100 ms serta frekuensi sebesar 9.4-29.9 hz. Lebih lanjut, ketika hasil yang



Gambar 4.11: Grafik *delay* dan frekuensi dari pengiriman citra antar-perangkat pada robot di simulasi.

didapatkan ditampilkan sebagai grafik seperti yang terlihat pada gambar 4.11, hasil yang didapatkan menunjukkan bahwa semakin besar ukuran data yang dikirimkan, maka *delay* yang dihasilkan akan cenderung naik dan frekuensi yang dihasilkan akan cenderung turun. Berbeda dengan yang dihasilkan di pengujian pada sesama perangkat, hingga batas resolusi 1280 x 720, pengiriman citra yang dilakukan bisa menghasilkan *delay* di bawah 20 ms dan frekuensi di atas 90%. Namun, diatas resolusi tersebut, *delay* yang dihasilkan naik drastis hingga 100 ms, sedangkan frekuensi yang dihasilkan turun drastis hingga 9.4 hz.

4.2.4 Pengujian Pengiriman Citra Kamera Antar-perangkat pada *Real Robot*

Pengujian pengiriman citra kamera antar-perangkat pada *real robot* memiliki kesamaan proses dengan yang dilakukan di pengujian sebelumnya yang ada di bagian 4.2.3, dimana di pengujian ini, *image viewer node* dan *command-line \$ ros2 topic* dijalankan di perangkat

berbeda yang terhubung menggunakan jaringan *ethernet* dengan perangkat yang menjalankan *V4L2 camera node*.

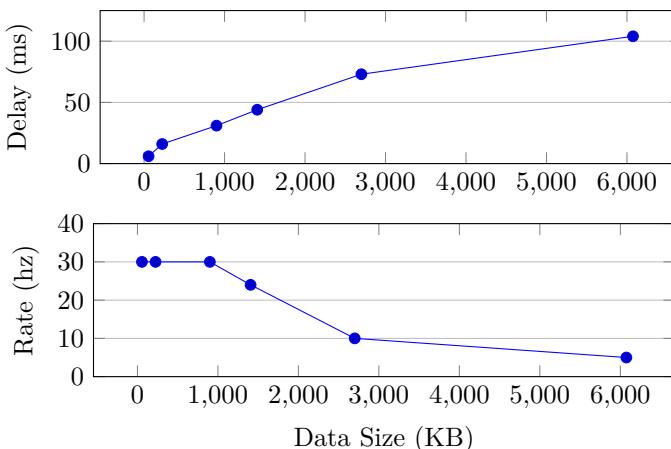
Tabel 4.10: Hasil *delay* dan frekuensi dari pengiriman citra kamera antar-perangkat pada *real robot*.

Resolution			Delay	Rate	
Width	Height	Size (KB)	ms	hz	percent
160	120	56	6.0	30.0	100.0%
320	240	225	16.0	30.0	100.1%
640	480	900	31.0	30.0	100.0%
800	600	1406	44.0	24.0	79.9%
1280	720	2700	73.0	10.0	33.3%
1920	1080	6075	104.0	5.0	16.7%

Sama seperti pengujian-pengujian sebelumnya, pengujian ini juga dilakukan dengan berbagai macam konfigurasi resolusi citra yang dikirim. Hasil pengujian ini bisa dilihat pada tabel 4.10. Pada tabel tersebut *width* dan *height* merupakan resolusi citra yang dikirimkan, *size* merupakan ukuran data yang dikirim, *delay* merupakan selang waktu yang dibutuhkan untuk sampai ke perangkat penerima, dan *rate* merupakan frekuensi pengiriman citra.

Dari data yang dihasilkan oleh pengujian ini dapat diketahui bahwa data yang dikirimkan memiliki *delay* sebesar 6-104 ms serta frekuensi sebesar 5-30 hz. Lebih lanjut, ketika hasil yang didapatkan ditampilkan sebagai grafik seperti yang terlihat pada gambar 4.12, seperti kesimpulan pada pengujian sebelumnya, hasil yang didapatkan menunjukkan bahwa semakin besar ukuran data yang dikirimkan, maka *delay* yang dihasilkan akan cenderung naik dan frekuensi yang dihasilkan akan cenderung turun.

Berbeda dengan yang dihasilkan di pengujian-pengujian sebelumnya, *delay* yang dihasilkan selalu naik secara drastis, dimana pada resolusi 640 x 480, *delay* yang dihasilkan bernilai 31.0 ms. Sedangkan untuk frekuensi, hingga batas resolusi 640 x 480, pengiriman citra yang dilakukan bisa menghasilkan frekuensi di atas 90% dari nilai FPS yang dimiliki kamera pada *real robot*. Namun, diatas resolusi tersebut, frekuensi yang dihasilkan akan turun drastis



Gambar 4.12: Grafik *delay* dan frekuensi dari pengiriman citra antar-perangkat pada *real robot*.

hingga 5.0 hz.

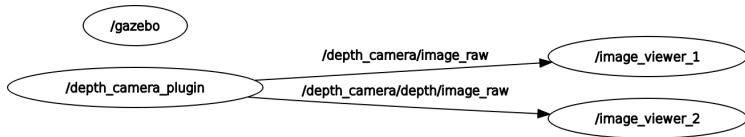
4.3 Pengujian *Depth Camera*

Pengujian *depth camera* dilakukan untuk menguji kemampuan sistem untuk menerima data citra berwarna maupun citra kedalaman (*depth image*) dari *depth camera* yang ada di simulasi maupun yang ada pada *real robot*. Pengujian ini melibatkan dua buah *image viewer node* yang digunakan untuk menampilkan tangkapan citra berwarna yang ada pada *topic /raw_image* dan citra kedalaman yang ada pada *topic /depth/raw_image*.

4.3.1 Pengujian Citra *Depth Camera* pada Robot di Simulasi

Pengujian citra *depth camera* pada robot di simulasi dilakukan dengan cara menjalankan lingkungan *indoor* di simulator dan menjalankan dua buah *image viewer node* untuk melihat hasil penerimaan data citra berwarna dan citra kedalaman. Seperti yang dapat dilihat pada gambar 4.13, *node /depth_camera_plugin* akan

mengirimkan *topic* `/depth_camera/image_raw` yang berisi citra berwarna ke *node* `/image_viewer_1`, serta *topic* `/depth_camera/depth<→/image_raw` yang berisi citra kedalaman ke *node* `/image_viewer_2`.



Gambar 4.13: Relasi antar-*node* dari pengujian citra *depth camera* pada robot di simulasi.

Hasilnya, kedua *image viewer node* tersebut akan menampilkan citra yang dikirim oleh *depth camera plugin*. Seperti yang dapat dilihat pada gambar 4.14, gambar pertama menampilkan citra dari sebuah ruangan *indoor* secara berwarna, sedangkan gambar kedua menampilkan citra dari sebuah ruangan *indoor* secara hitam putih. Citra kedua merupakan citra kedalaman, dimana semakin terang warna pada suatu pixel, maka jarak dari titik tersebut lebih jauh dari jangkauan kamera.



Gambar 4.14: Perbandingan hasil tangkapan citra berwarna dan citra kedalaman di simulasi.

4.3.2 Pengujian Citra *Depth Camera* pada *Real Robot*

Sama seperti pada pengujian sebelumnya di bagian 4.3.1, pengujian citra *depth camera* pada *real robot* dilakukan dengan men-

jalankan dua buah *image viewer node* untuk melihat hasil penerimaan data citra berwarna dan citra kedalaman. Hanya saja, sebagai ganti dari *node /depth_camera_plugin* yang ada di simulasi, data citra berwarna dan citra kedalaman yang digunakan akan berasal dari *Kinect2 node*. Seperti yang dapat dilihat pada gambar 4.15, *node /kinect2* akan mengirimkan *topic /kinect2/depth↔/image_raw* yang berisi citra kedalaman ke *node /image_viewer_2*, serta *topic /kinect2/image_raw* yang berisi citra berwarna ke *node /image_viewer_1*.



Gambar 4.15: Relasi antar-*node* dari pengujian citra *depth camera* pada *real robot*.

Hasilnya, kedua *image viewer node* tersebut akan menampilkan citra yang dikirim oleh *Kinect2 node*. Seperti yang dapat dilihat pada gambar 4.16, gambar pertama menampilkan citra dari ruang lab komputer secara berwarna, sedangkan gambar kedua menampilkan citra dari ruang lab komputer secara hitam putih. Berbeda dengan hasil yang didapatkan di simulasi, pada pengujian ini, citra kedalaman yang didapatkan memiliki hasil yang sangat terang. Hal ini terjadi karena pendeknya jangkauan yang dimiliki Kinect V2, yakni hanya sejauh 4.5 meter.

4.4 Pengujian Deteksi Pose

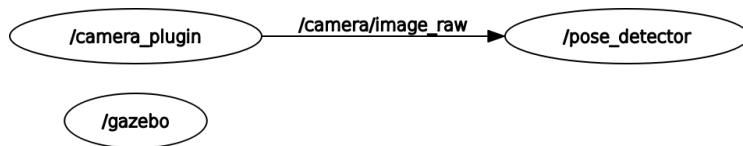
Pengujian deteksi pose dilakukan untuk menguji kemampuan visi komputer dari citra yang berasal dari sistem yang dibuat. Selain itu pengujian ini juga dilakukan untuk menguji kemampuan model pengguna dalam mensimulasikan pengguna real. Pengujian ini melibatkan komponen kamera yang ada pada robot dan sebuah *pose detector node* sebagai *node behavior* yang akan melakukan proses deteksi pose dari tangkapan citra pengguna.



Gambar 4.16: Perbandingan hasil tangkapan citra berwarna dan citra kedalaman pada *real robot*.

4.4.1 Pengujian Deteksi Pose Pengguna pada Robot di Simulasi

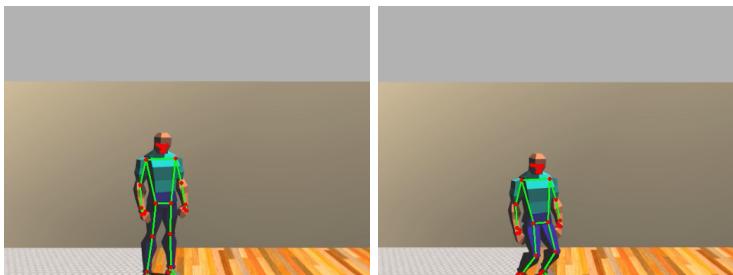
Pengujian deteksi pose pengguna pada robot di simulasi dilakukan dengan cara menjalankan lingkungan simulasi yang berisi model pengguna dan menjalankan *pose detector node* yang melakukan proses visi komputer untuk mendeteksi pose dari tangkapan citra model pengguna. Seperti yang terlihat pada gambar 4.17, *camera plugin* yang ada pada model robot akan mengirimkan tangkapan citra melalui *topic /camera/image_raw* yang nantinya akan diterima oleh *pose detector node*.



Gambar 4.17: Relasi antar-*node* dari pengujian deteksi pose pengguna pada robot di simulasi.

Oleh *pose detector node*, data citra yang diterima tersebut kemudian akan diproses sehingga menghasilkan titik-titik dari setiap bagian pose dari model pengguna. Titik-titik tersebut kemudian akan digambarkan pada citra yang diproses dan ditampilkan dalam bentuk GUI. Seperti yang terlihat pada gambar 4.18, pose yang

dendeteksi oleh *pose detector node* mampu mendeteksi keseluruhan pose yang membentuk model pengguna. Pose tersebut juga dapat terdeteksi ketika posisi kaki model pengguna berada dalam kondisi berdiri maupun ditekuk.



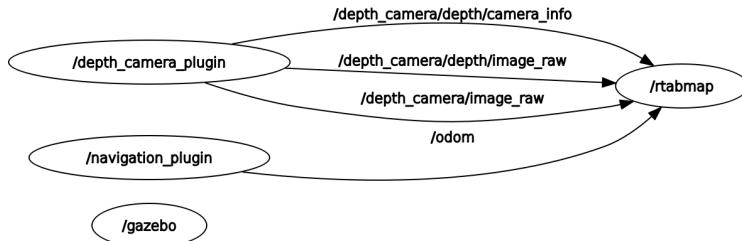
Gambar 4.18: Hasil deteksi pose pengguna pada robot di simulasi.

4.5 Pengujian SLAM

Pengujian *simultaneous localization and mapping* (SLAM) dilakukan untuk menguji kemampuan sistem ketika banyak komponen yang ada pada robot digunakan untuk suatu proses secara bersamaan. Selain itu pengujian ini juga dilakukan untuk menguji kemampuan lingkungan dalam mensimulasikan ruangan yang dapat digunakan pada *real robot*. Pengujian ini melibatkan *RTABMap node* yang berasal dari *package rtabmap_ros* [43] untuk melakukan pemetaan ruangan sekaligus lokalisasi menggunakan metode SLAM.

4.5.1 Pengujian Pemetaan Ruangan pada Robot di Simulasi

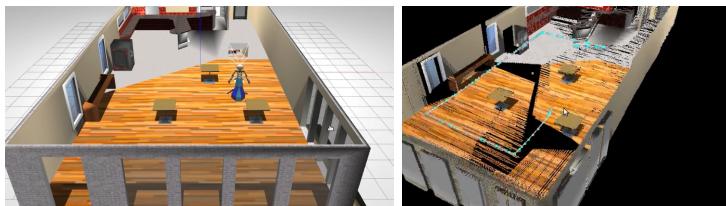
Pengujian pemetaan ruangan di simulasi dilakukan dengan cara menjalankan ruangan *indoor* pada simulator Gazebo dan menjalankan *RTAB-Map node* yang melakukan pemetaan ruangan dari data yang dikirimkan model robot yang ada di simulasi. Seperti yang dapat dilihat pada gambar 4.19, *node /depth_camera_plugin* akan mengirimkan data *depth camera* melalui tiga buah *topic* ke *node /rtabmap*, sedangkan *node /navigation_plugin* akan mengirimkan data melalui *topic /odom* ke *node /rtabmap*.



Gambar 4.19: Relasi antar-*node* dari pengujian pemetaan ruangan pada robot di simulasi.

RTAB-Map node kemudian akan memproses keseluruhan data yang didapat dari komponen *depth camera* dan navigasi untuk melakukan pemetaan pada ruangan yang ada di simulasi. Seperti yang dapat dilihat pada gambar 4.20, gambar pertama menunjukkan tangkapan layar yang ada di simulasi ketika robot sedang bergerak sambil melakukan pemetaan ruangan, sedangkan gambar kedua menunjukkan tangkapan layar dari hasil pemetaan yang ditampilkan secara visual oleh RTAB-Map.

Dari hasil pemetaan tersebut, dapat dilihat bahwa citra berwarna dan citra kedalaman yang diterima diubah kedalam bentuk *point cloud* oleh RTABMap sehingga menghasilkan bentuk peta seperti yang ada di visualisasi tersebut. Di dalam visualisasi tersebut juga terlihat bentuk lintasan yang telah dilalui robot selama melakukan pemetaan sebagai garis dengan titik berwarna biru.



Gambar 4.20: Proses dan hasil pemetaan ruangan pada robot di simulasi.

[Halaman ini sengaja dikosongkan]

BAB V

PENUTUP

Pada bab ini akan dipaparkan kesimpulan dari hasil pengujian yang akan menjadi jawaban dari permasalahan yang diangkat oleh penelitian ini. Selain itu akan dipaparkan juga saran mengenai hal yang bisa dilakukan untuk mengembangkan penelitian ini ke arah yang lebih lanjut.

5.1 Kesimpulan

Berdasarkan hasil pengujian yang telah dilakukan, lingkungan simulasi yang dibuat menggunakan simulator Gazebo mampu digunakan untuk melakukan pengujian SARs secara virtual. Hasil pembacaan data sensor pada robot, seperti kamera dan *depth camera*, dapat disimulasikan pada lingkungan yang dibuat sehingga memungkinkan pengambilan data pengujian untuk dilakukan secara virtual. Kemudian, dengan adanya abstraksi dari sistem kontroler yang dibuat menggunakan ROS 2, program *behavior* yang diujikan pada model robot di simulasi menghasilkan tindakan yang sama ketika diujikan pada robot fisik. Seperti hasil ketika robot diperintahkan untuk bergerak, dimana hasil posisi dan orientasi akhir tersebut mendekati hasil yang diharapkan dengan perbedaan *error* sebesar 2.6% di simulasi dan 12.5% pada kondisi *real*.

Dengan performa yang dimiliki oleh ROS 2, pertukaran data yang terjadi antara program dapat dilakukan secara *real-time* dengan *delay* yang rendah. Hal ini sesuai dengan hasil pengiriman data citra yang hingga resolusi 640 x 480 mampu menghasilkan *delay* kurang dari 50 ms dan frekuensi di atas 90% pada sesama perangkat maupun antar-perangkat. Pada simulator tersebut, model pengguna mampu mensimulasikan pengguna *real* melalui hasil deteksi pose yang dilakukan terhadap model tersebut, sedangkan lingkungan simulasi mampu mensimulasikan sebuah ruangan melalui hasil pemetaan yang dilakukan oleh robot secara virtual.

5.2 Saran

Pengembangan lebih lanjut dari penelitian ini bisa dilakukan dengan meningkatkan keakuratan komponen yang ada serta penambahan komponen yang bisa digunakan seperti lengan *manipulator* dan *lidar scanner* pada model robot yang ada di simulasi. Pilihan jenis perangkat yang digunakan pada robot fisik juga bisa ditambahkan, seperti Intel RealSense dan ZED *stereo camera* sebagai pengganti Kinect V2 untuk komponen *depth camera*.

Untuk simulator, sebagai alternatif dari Gazebo, beberapa pilihan simulator lain juga bisa digunakan untuk melakukan pengujian seperti Webots dan OpenAI Gym. Selain itu, untuk mendapatkan hasil visual yang lebih baik, simulator juga bisa dikembangkan menggunakan *game engine* terkini yang mampu mensimulasikan lingkungan 3D beserta *physics* yang ada di dalamnya seperti pada *game engine* Unity dan Unreal.

Terkait dengan *assistive robotics*, pengembangan lebih lanjut juga bisa dilakukan dengan meningkatkan interaksi antara manusia dengan robot, seperti membawa pengguna untuk berinteraksi dengan robot di dunia simulasi menggunakan perangkat *virtual reality* (VR) maupun mensimulasikan masukan suara dari pengguna yang dapat ditangkap oleh robot yang ada di simulasi. Lebih lanjut, pengembangan juga bisa dilakukan dengan meningkatkan model pengguna yang ada di simulasi, seperti memberikan kemampuan untuk mengubah ekspresi maupun meningkatkan animasi dari gerakan yang dilakukan.

DAFTAR PUSTAKA

- [1] M. Heerink, B. Kröse, V. Evers, and B. Wielinga, “Assessing acceptance of assistive social agent technology by older adults: the almere model,” in *International Journal of Social Robotics*, 2010, p. 361–375.
- [2] “Understanding ros 2 nodes — ros 2 documentation: Foxy documentation,” 2021. [Online]. Available: <https://index.ros.org/doc/ros2/Tutorials/Understanding-ROS2-Nodes/>
- [3] “Understanding ros 2 actions — ros 2 documentation: Foxy documentation,” 2021. [Online]. Available: <https://docs.ros.org/en/foxy/Tutorials/Understanding-ROS2-Actions.html>
- [4] “About internal ros 2 interfaces — ros 2 documentation: Foxy documentation,” 2021. [Online]. Available: <https://docs.ros.org/en/foxy/Concepts/About-Internal-Interfaces.html>
- [5] “rqt graph - visualize and debug your ros graph - the robotics back-end,” 2021. [Online]. Available: <https://roboticsbackend.com/rqt-graph-visualize-and-debug-your-ros-graph/>
- [6] “Simulating moose — moose tutorials 0.0.2 documentation,” 2021. [Online]. Available: <http://www.clearpathrobotics.com/assets/guides/kinetic/moose/MooseSimulation.html>
- [7] H. Oliveira, A. Sousa, A. Moreira, and P. Costa, “Dynamical models for omni-directional robots with 3 and 4 wheels,” *ICINCO 2008 - Proceedings of the 5th International Conference on Informatics in Control, Automation and Robotics*, vol. 1, 2008.
- [8] H. Choi, Y. Lee, M. Lee, J. Kim, and Y. Shim, “A wearable virtual chair with the passive stability assist,” in *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2015, pp. 3897–3900.

- [9] “20200501 time of flight - time-of-flight camera - wikipedia,” 2021. [Online]. Available: https://en.wikipedia.org/wiki/Time-of-flight_camera#/media/File:20200501_Time_of_flight.svg
- [10] “Kinect 2.0 for windows available july 15 — time,” 2021. [Online]. Available: <https://time.com/2962269/kinect-v2-0-for-windows/>
- [11] “Home - mediapipe,” 2021. [Online]. Available: <https://google.github.io/mediapipe/>
- [12] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann, “Blazepose: On-device real-time body pose tracking,” 2020.
- [13] P. Gonçalves, P. Torres, C. Alves, F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J. Zufferey, D. Floreano, and A. Martinoli, “The e-puck, a robot designed for education in engineering,” *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, vol. 1, 2009.
- [14] M. Blatnický, J. Dižo, J. Gerlici, M. Sága, T. Lack, and E. Kučba, “Design of a robotic manipulator for handling products of automotive industry,” *International Journal of Advanced Robotic Systems*, vol. 17, no. 1, 2020.
- [15] D. Feil-Seifer and M. J. Mataric, “Defining socially assistive robotics,” in *9th International Conference on Rehabilitation Robotics, 2005. ICORR 2005.*, 2005, pp. 465–468.
- [16] Z. Erickson, V. Gangaram, A. Kapusta, C. K. Liu, and C. C. Kemp, “Assistive gym: A physics simulation framework for assistive robotics,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 10169–10176.
- [17] K. Takaya, T. Asai, V. Kroumov, and F. Smarandache, “Simulation environment for mobile robots testing using ros and gazebo,” in *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*, 2016, pp. 96–101.

- [18] O. Michel, “Cyberbotics ltd. webotsTM: Professional mobile robot simulation,” *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004.
- [19] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.
- [20] E. Rohmer, S. P. N. Singh, and M. Freese, “V-rep: A versatile and scalable robot simulation framework,” in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1321–1326.
- [21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.
- [22] W. Qian, Z. Xia, J. Xiong, Y. Gan, Y. Guo, S. Weng, H. Deng, Y. Hu, and J. Zhang, “Manipulation task simulation using ros and gazebo,” in *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, 2014, pp. 2594–2598.
- [23] M. Zhang, M. Lan, J. Lin, S. Wang, K. Liu, F. Lin, and B. M. Chen, “A high fidelity simulator for a quadrotor uav using ros and gazebo,” in *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, 2015, pp. 002 846–002 851.
- [24] Y. Maruyama, S. Kato, and T. Azumi, “Exploring the performance of ros2,” in *2016 International Conference on Embedded Software (EMSOFT)*, 2016, pp. 1–10.
- [25] A. Clegg, Z. Erickson, P. Grady, G. Turk, C. C. Kemp, and C. K. Liu, “Learning to collaborate from simulation for robot-assisted dressing,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2746–2753, 2020.
- [26] C. Rich and C. L. Sidner, “Robots and avatars as hosts, advisors, companions, and jesters,” *AI Magazine*, vol. 30, no. 1, p. 29, 2009.

- [27] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, “Ros: an open-source robot operating system,” *ICRA Workshop on Open Source Software*, vol. 3, 2009.
- [28] G. Pardo-Castellote, “Omg data-distribution service: architectural overview,” in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, 2003, pp. 200–206.
- [29] J. M. Schlesselman, G. Pardo-Castellote, and B. Farabaugh, “Omg data-distribution service (dds): architectural update,” in *IEEE MILCOM 2004. Military Communications Conference, 2004.*, vol. 2, 2004, pp. 961–967 Vol. 2.
- [30] “Ros 2 middleware interface,” 2021. [Online]. Available: https://design.ros2.org/articles/ros_middleware_interface.html
- [31] “eprosima fast dds,” 2021. [Online]. Available: <https://www.eprosima.com/index.php/products-all/eprosima-fast-dds>
- [32] “Eclipse cyclone dds — projects.eclipse.org,” 2021. [Online]. Available: <https://projects.eclipse.org/projects/iot.cyclonedds>
- [33] “Ros 2 client interfaces (client libraries) — ros 2 documentation: Foxy documentation,” 2021. [Online]. Available: <https://docs.ros.org/en/foxy/Concepts/About-Client-Interfaces.html>
- [34] “rclnodejs - npm,” 2021. [Online]. Available: <https://www.npmjs.com/package/rclnodejs>
- [35] “Introspection with command line tools — ros 2 documentation: Foxy documentation,” 2021. [Online]. Available: <https://docs.ros.org/en/foxy/Concepts/About-Command-Line-Tools.html>
- [36] “Qt, cross-platform software development for embedded and desktop,” 2021. [Online]. Available: <https://www.qt.io/>
- [37] “Overview and usage of rqt — ros 2 documentation: Foxy documentation,” 2021. [Online]. Available: <https://docs.ros.org/en/foxy/Concepts/About-RQt.html>

- [38] B. Gerkey, R. Vaughan, and A. Howard, “The player/stage project: Tools for multi-robot and distributed sensor systems,” *Proceedings of the International Conference on Advanced Robotics*, 2003.
- [39] “Gazebo : Tutorial : Plugins 101,” 2021. [Online]. Available: http://gazebosim.org/tutorials/?tut=plugins_hello_world
- [40] A. Kim and M. Golnaraghi, “A quaternion-based orientation estimation algorithm using an inertial measurement unit,” in *PLANS 2004. Position Location and Navigation Symposium (IEEE Cat. No.04CH37556)*, 2004, pp. 268–272.
- [41] G. J. Iddan and G. Yahav, “Three-dimensional imaging in the studio and elsewhere,” in *Three-Dimensional Image Capture and Applications IV*, B. D. Corner, J. H. Nurre, and R. P. Pargas, Eds., vol. 4298, International Society for Optics and Photonics. SPIE, 2001, pp. 48 – 55.
- [42] L. Xiang, F. Echtler, C. Kerl, T. Wiedemeyer, Lars, hanyazou, R. Gordon, F. Facioni, laborer2008, R. Wareham, M. Goldhoorn, alberth, gaborpapp, S. Fuchs, jmtatsch, J. Blake, Federico, H. Jungkurth, Y. Mingze, vinouz, D. Coleman, B. Burns, R. Rawat, S. Mokhov, P. Reynolds, P. Viau, M. Fraissinet-Tachet, Ludique, J. Billingham, and Alistair, “libfreenect2: Release 0.2,” Apr. 2016. [Online]. Available: <https://doi.org/10.5281/zenodo.50641>
- [43] M. Labb   and F. Michaud, “Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation,” *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21831>
- [44] R. Dikairono, Setiawardhana, D. Purwanto, and T. A. Sardjono, “Cnn-based self localization using visual modelling of a gyrocompass line mark and omni-vision image for a wheeled soccer robot application,” *International Journal of Intelligent Engineering and Systems*, 2020.

- [45] Muhtadin, R. M. Zanuar, I. K. E. Purnama, and M. H. Purnomo, “Autonomous navigation and obstacle avoidance for service robot,” in *2019 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM)*, 2019, pp. 1–8.
- [46] M. Muhtadin, M. Arrazi, S. Ali, T. Pratama, D. Wicaksono, A. Putra, I. Ari, A. Maulana, O. Bramastyo, S. Asshakina, M. Attamimi, M. Arifin, M. Hery Purnomo, and D. Purwanto, “Ichiro robots winning robocup 2018 humanoid teensize soccer competitions,” in *RoboCup 2018: Robot World Cup XXII*, 2019, pp. 425–435.
- [47] Z. Mohamed and G. Capi, “Development of a new mobile humanoid robot for assisting elderly people,” *Procedia Engineering*, vol. 41, pp. 345 – 351, 2012.
- [48] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann, “Mediapipe: A framework for building perception pipelines,” 2019.
- [49] “Linux kernel media documentation — the linux kernel documentation,” 2021. [Online]. Available: <https://linuxtv.org/downloads/v4l-dvb-apis-new/index.html>