

## SKRIPSI

RANCANG BANGUN SISTEM PEMETAAN DAN LOKALISASI BERBASIS ALGORITMA SLAM MENGGUNAKAN *DEPTH SENSOR KINECT* PADA *MOBIL E ROBOT*

Disusun dan diajukan oleh:

FARIZ ACHMAD FAIZAL  
D041191091



PROGRAM STUDI SARJANA TEKNIK ELEKTRO  
FAKULTAS TEKNIK  
UNIVERSITAS HASANUDDIN  
GOWA  
2023

## LEMBAR PENGESAHAN SKRIPSI

RANCANG BANGUN SISTEM PEMETAAN DAN LOKALISASI BERBASIS ALGORITMA SLAM MENGGUNAKAN *DEPTH SENSOR KINECT* PADA *MOBIL E ROBOT*

Disusun dan diajukan oleh

FARIZ ACHMAD FAIZAL

D041191091

Telah dipertahankan di hadapan Panitia Ujian yang dibentuk dalam rangka Penyelesaian Studi Program Sarja Program Studi Teknik Elektronika

O

Fakultas Teknik Universitas Hasanuddin

Pada tanggal

Dan dinyatakan telah memenuhi syarat kelulusan

Menyetuui,

Pembimbing Utama,

Pembimbing Pendamping,

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

Dr. A. Ejah Umraeni Salam, S.T, M.T.

NIP. 197708172005011003

NIP. 197209081997022001

Ketua Program Studi,

Dr. Eng. Ir. Dewiani, MT.

NIP. 196910261994122001

## PERNYATAAN KEASLIAN

Yang bersignature dibawah ini:

Nama : Fariz Achmad Faizal

NIM : D041191091

Program Studi : Teknik Elektro

Jenjang : S1

Menyatakan dengan ini bahwa karya tulisan saya berjudul

### RANCANG BANGUN SISTEM PEMETAAN DAN LOKALISASI BERBASIS ALGORITMA SLAM MENGGUNAKAN DEPTH SENSOR KINECT PADA MOBILE ROBOT

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

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

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

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

Gowa, 11 September 2023

Yang Menyatakan

Fariz Achmad Faizal

## ABSTRAK

**FARIZ ACHMAD FAIZAL.** *Rancang Bangun Sistem Pemetaan Dan Lokalisasi Berbasis Algoritma Slam Menggunakan Depth Sensor Kinect Pada Mobile Robot* (dibimbing oleh Muh Anshar dan Ejah Umraeni Salam)

Kemampuan pokok yang diperlukan oleh sebuah robot yang bergerak adalah kemampuan untuk bergerak dari satu lokasi ke lokasi lain dengan aman saat menjalankan tugas yang ditugaskan. Untuk memenuhi kebutuhan ini, robot harus mampu mengenali lingkungan sekitarnya dan menentukan posisinya terhadap lingkungan tersebut. Pengenalan lingkungan dilakukan dengan menggunakan sensor Microsoft Kinect untuk menghasilkan peta dua dimensi (2D). Sementara untuk menentukan posisi relatif robot dalam peta (lokalisasi), metode particle filter digunakan. Untuk menerapkan pemetaan dan lokalisasi secara bersamaan, algoritma visual Simultaneous Localization and Mapping (SLAM) RTABmap, yang berbasis Robot Operating System (ROS), digunakan pada robot yang telah dirancang. Hasil yang diperoleh adalah peta probabilitas 2D occupancy grid map. Berdasarkan penelitian ini, akurasi odometri, yang melibatkan penggabungan data dari encoder, inertial measurement unit (IMU), dan Kinect, mampu mengestimasi pose dan posisi robot dengan deviasi sekitar 0.016 meter. Sementara pengujian hasil pemetaan menunjukkan hasil terbaik ketika memetakan r

uangan berukuran kecil tanpa adanya pengaruh sinar matahari, dengan kesalahan Root Mean Square Error (RMSE) sekitar 4.90% terhadap posisi asli.

Kata Kunci : Automation, SLAM, Kinect, RTABMAP, AMCL, ROS, Robotics, Odometry

## ABSTRACT

**FARIZ ACHMAD FAIZAL.** *Design of Slam-Based Mapping and Localization System Using Kinect Depth Sensor on Mobile Robot* (supervised by Muham Anshar and Ejah Umraeni Salam)

The fundamental capability required by a mobile robot is the ability to move safely from one location to another while performing assigned tasks. To fulfill this need, a robot must be capable of recognizing its surrounding environment and determining its relative position within that environment. Environment recognition is achieved using the Microsoft Kinect sensor to generate a two-dimensional (2D) map. Meanwhile, for estimating the robot's relative position within the map (localization), the particle filter method is employed. To implement mapping and localization simultaneously, the visual Simultaneous Localization and Mapping (SLAM) algorithm RTABmap, based on the Robot Operating System (ROS), is utilized on the designed robot. The obtained result is a 2D probabilistic occupancy grid map. From the conducted research, it was found that the

odometry accuracy, involving the fusion of data from encoders, inertial measurement unit (IMU), and Kinect, was able to estimate the robot's pose and position with a deviation of approximately 0.016 meters. In the mapping experiments, the best results were obtained when mapping small-sized rooms without the influence of sunlight, with a Root Mean Square Error (RMSE) of around 4.90% compared to the original map.

**Keywords:** Automation, SLAM, Kinect, RTABMAP, AMCL, ROS, Robotics, Odometry

## DAFTAR ISI

LEMBAR PENGESAHAN SKRIPSI	ii
PERNYATAAN KEASLIAN	iii
ABSTRAK	iv
ABSTRACT	v
DAFTAR ISI	vi
DAFTAR GAMBAR	viii
DAFTAR TABEL	x
DAFTAR LAMPIRAN	xi
KATA PENGANTAR	xii
BAB I PENDAHULUAN	1
I. 1 Latar Belakang	1
I. 2 Rumusan Masalah	3

I. 3 Tujuan Penelitian	3
1. 4 Manfaat Penelitian	3
1. 5 Ruang Lingkup	4
BAB II TINJAUAN PUSTAKA	5
2. 1 Penelitian Terkait	5
2. 2 Robot	7
2. 3 <i>Mobile Robot</i>	8
2. 4 <i>Robot Operating System</i>	10
2. 4. 1 <i>ROS Node</i>	11
2. 4. 2 <i>ROS Topic</i>	12
2. 4. 3 <i>ROS Package</i>	12
2. 5 <i>Telerobotic</i>	12
2. 6 Robot Preseption	14
2. 7 Odometry	15
2. 10 SLAM	16
2. 11 Real-Time Appearance-Based Mapping	16
2. 11. 1 Node Odometry RTAB-Map	18
2. 11. 2 Graph creation and data storage	19
2. 11. 4 Deteksi Loop Closure	21
2. 11. 5 Optimization	22
2. 12 Adaptive Monte Carlo Localization	22
BAB III METODE PENELITIAN	24
3. 1 Waktu dan Lokasi Penelitian	24
3. 2 Rancangan Umum Penelitian	24
3. 3 Rancangan Perangkat Keras	28
3. 3. 1 Mekanisasi Robot	28
3. 3. 2 Desain Elektronika	30
3. 4 Rancangan Perangkat Lunak	43
3. 4. 1 Instalasi ROS	44
3. 4. 2 Pemodelan Robot dengan URDF	46
3. 4. 3 Pembuatan node program pada ROS	47
3. 5 Rancangan Pengujian	55
3. 5. 1 Evaluasi Hasil Pemetaan dengan Metode RMSE	56

3. 5. 2 Absolute trajectory error (ATE)	57
3. 5. 3 Relative pose error (RPE)	59
BAB IV ANALISIS DAN PEMBAHASAN	60
4. 1 Pengujian odometry dengan position tracking	60
4. 2 pengujian hasil pemetaan	62
4. 2. 1 Pemetaan pada ruang uji berbeda	62
4. 2. 2 Pemetaan dengan tingkat <i>Luminansi berbeda</i>	66
4. 3 pengujian kemampuan lokalisasi posisi robot dalam peta	68
4. 3. 1 Hasil Pengujian ATE dan RPE	68
4. 3. 3 Pengujian Penentuan posisi dengan partikel filter	71
BAB V KESIMPULAN DAN SARAN	73
5. 1 Kesimpulan	73
5. 2 Saran	74
DAFTAR PUSTAKA	75
LAMPIRAN	79

## DAFTAR GAMBAR

Gambar 1	Mobile Robot Menggunakan Caterpillar Track	8
Gambar 2	Mobile Robot Berkaki	8
Gambar 3	Mobile Robot Beroda	9
Gambar 4	Prinsip Kerja Kerangka Perangkat Lunak Berbasis ROS untuk Mengontrol Operasi Semua Sistem Robot Multi-Node	11
Gambar 5	<i>Man-Machine Interface</i>	14
Gambar 6	Blok Diagram RTAB-Map pada node ROS	17
Gambar 7	Alur Kerja AMCL	23
Gambar 8	Blok Diagram Arsitektur AMCL	23
Gambar 9	Diagram Alir Rancangan Penelitian	25
Gambar 10	Sistem Pemetaan dan Lokalisasi	26
Gambar 11	Diagram Blok Sistem	26
Gambar 12	Diagram Masukan	27
Gambar 13	Diagram proses	27
Gambar 14	Digaram keluaran	28
Gambar 15	Gambar Desain Robot	29
Gambar 16	Blok Diagram Umum hubungan antar komponen	31
Gambar 17	Subsistem Sensor Robot	31
Gambar 18	Gambar Data Kedalaman Sensor Kinect	32
Gambar 19	Komponen-Komponen Sensor Kinect	33
Gambar 20	Data Depth Kinect, (a) IR Kinect, (b) Depth Map Kinect	34
Gambar 21	Prinsip Kerja Kinect	34
Gambar 22	Data Depth Kinect (a) Bentuk 2D Array, (b) Dalam Plot 3D	35
Gambar 23	Kinect memberikan tiga output: Gambar IR, gambar RGB	35

Gambar	24 Invers Kinect pada fungsi kedalaman asli	36
Gambar	25 Blok diagram modul sensor IMU MPU-60X0	37
Gambar	26 Sistem kerja encoder	38
Gambar	27 Board Arduino Mega 2560	39
Gambar	28 Laptop HP Victus 15	40
Gambar	29 Prosesor Intel core i7	40
Gambar	30 GPU Nvidia RTX3050	40
Gambar	31 Motor DC <i>Power Window</i> 12 V	42
Gambar	32 skematik witres-bot	43
Gambar	33 Visualisasi Model Robot dengan Rviz	46
Gambar	34 Komunikasi Komputer-Arduino Ros Serial	47
Gambar	35 Kinematics of Differential Drive Robots	49
Gambar	36 Blok diagram pemetaan dan lokalisasi dengan RTABMAP	50
Gambar	37 Flowchart sistem pemetaan	51
Gambar	38 Flowchart lokalisasi dengan AMCL	53
Gambar	39 Visualisasi Lokalisasi dengan AMCL Menggunakan Rviz	55
Gambar	40 Pengujian Pembacaan Odometry	61
Gambar	41 Perbandingan trajectory dari sumber odometry berbeda terhadap ground truth	61
Gambar	42 Denah Lab Sistem kendali dan Instrumentasi	63
Gambar	43 Denah Gedung COT Lantai 2	64
Gambar	44 Denah Gedung Elektro Lantai 3	65
Gambar	45 Pengujian pada skenario 1	66
Gambar	46 Data lux pada skenario 1	66
Gambar	47 Pengujian pada skenario 2	67
Gambar	48 Data lux pada skenario 2	67
Gambar	49 Visualisasi Rviz Robot AMCL Estimasi Pose	69
Gambar	50 Perbandingan Trajectory Lokalisasi AMCL Terhadap Ground Truth Gambar	69

## DAFTAR TABEL

Tabel 1 Jadwal kegiatan penelitian	24
Tabel 2 Komponen utama rancang bangun robot otonom Waiter-Bot	29
Tabel 3 Spesifikasi Arduino Mega 2560	39
Tabel 4 Spesifikasi Komputer yang Digunakan	41
Tabel 5 Simpangan odometry terhadap ground truth	62
Tabel 6 Hasil pemetaan ruang uji 1	63
Tabel 7 Hasil pemetaan ruang uji 2	64
Tabel 8 Hasil pemetaan ruang uji 3	65
Tabel 9 Hasil pemetaan pada kondisi lux skenario 1	67
Tabel 10 Hasil pemetaan pada kondisi lux skenario 3	68
Tabel 11 Analisis ATE dan RPE terhadap hasil estimasi lokalisasi	70
Tabel 12 visualisasi lokalisasi dengan jumlah partikel berbeda	71
Tabel 13 Perbandingan waktu untuk mencapai konvergensi pada jumlah partikel berbeda	72

## DAFTAR LAMPIRAN

Lampiran 1 Dokumentasi Pelaksanaan Penelitaian	79
Lampiran 2 Kode Pemrograman Low Level	83
Lampiran 3 Pemodelan Robot dengan URDF	99
Lampiran 4 Kode Pemrograman ROS	104
Lampiran 5 Launch File SLAM	116
Lampiran 6 Data Hasil Pemetaan Ruang Uji 1 (Laboratorium Sistem Kendali dan instrumentasi)	119
Lampiran 7 Data Hasil Pemetaan Ruang Uji 2 (Gedung Center Of Technology Lantai 2)	126
Lampiran 8 Data Hasil Pemetaan Ruang Uji 3 (Gedung Elektro Lantai 3)	133
Lampiran 9 Data Hasil Pemetaan pada kondisi luminensi skenario 1	
	140
Lampiran 10 Data Hasil Pemetaan pada kondisi luminensi skenario 2	147

## KATA PENGANTAR

Puji syukur Penulis panjatkan ke hadirat Allah SWT yang telah memberikanrahmat dan hidayah-Nya kepada Penulis sehingga dapat menyelesaikan penelitiandan skripsi ini yang berjudul “PERANCANGAN SISTEM PEMETAAN DAN LOKALISASI BERBASIS ALGORITMA SLAM MENGGUNAKAN DEPTH SENSOR KINECT PADA MOBILE ROBOT” . Sertaselawat dan salam penulis sampaikan kepada junjungan Nabi Muhammad SAWyang telah membawa kita dari alam jahiliah menuju alam kemajuan sepertisekarang ini. Penyelesaian skripsi ini merupakan salah satu upaya Penulis untukmemenuhi syarat memperoleh gelar Sarjana Teknik di Departemen TeknikElektro Fakultas Teknik Universitas Hasanuddin.

Penulis menyadari bahwa untuk menyelesaikan skripsi serta penelitian ini tidaklah mudah, banyak hambatan dan masalah yang penulis hadapi hingga sampai ke penyelesaian skripsi dan penelitian ini. Namun berkat doa dan dukungan dari berbagai pihak akhirnya skripsi dan penelitian ini Alhamdulillah Penulis telah berhasil menyelesaiannya. Oleh sebab itu pada kesempatan kali ini perkenankan Penulis menyampaikan ucapan terima kasih kepada :

- 1) Bapak Muh Anshar, ST. M.Sc(Research), Ph. D. selaku Dosen Pembimbing I dan Ibu Dr. A. Ejah Umraeni Salam, S.T., M.T. selaku Dosen Pembimbing II, yang telah memberikan bimbingan, saran dan dukungan kepada penulis dalam penggerjaan skripsi ini.
- 2) Bapak Prof. Dr.-Ing. Ir. Faizal Arya Samman, IPU., ACPE. selaku Dosen Penguji I dan Ibu Ida Rachmaniar Sahali, S.T., M.T. selaku Dosen Penguji II, yang telah memberikan kritik dan saran dalam penyusunan skripsi ini.
- 3) Kedua orang tua dan saudara penulis, Bapak Faizal Muis, Ibu Firna Sofianti Thamrin, Adik Fadli Fathurrahman Faizal, dan Adik Filidzah Muthiah Faizal yang telah memberikan do'a dan dukungan selama penyelesaian skripsi ini.
- 4) Ibu Dr. Eng. Ir. Dewiani, M.T., IPM. selaku Kepala Departemen Teknik Elektro Fakultas Teknik Universitas Hasanuddin.
- 5) Bapak Prof. Dr. Eng. Ir. Muhammad Isran Ramli, S.T., M.T. selaku Dekan Fakultas Teknik Universitas Hasanuddin.
- 6) Bapak Prof. Dr. Ir. Jamaluddin Jompa, M.Sc. selaku Rektor Universitas Hasanuddin.
- 7) Bapak/Ibu Dosen dan Staff Departemen Teknik Elektro Fakultas Teknik Universitas Hasanuddin yang telah banyak memberikan ilmu dan bantuan selama menempuh proses perkuliahan.

- 8) Seluruh teman - teman Lab Riset IASCR terkhususnya Salam, Hasbih dan Arya yang selalu menemani dalam proses penyelesaian skripsi ini.
- 9) Saudari Nur Iqrima Fitrah Qalby yang telah memberikan dukungan dan motivasi dalam penyusunan skripsi ini.
- 10) Seluruh teman - teman Bani TR19GER yang turut memberikan dukungan dalam penyelesaian skripsi ini.
- 11) Seluruh pihak yang tidak dapat disebutkan satu persatu dalam membantu dan mendukung penyelesaian skripsi ini.

Semoga Allah SWT membalas kebaikan semua pihak yang telah membantu Penulis menyelesaikan skripsi ini. Penulis menyadari skripsi ini masih jauh dari kata sempurna, sehingga penulis dengan sangat terbuka menerima kritikan dan saran yang membangun untuk memperbaiki skripsi dan penelitian ini ke depannya.

Gowa, 11 September 202

3

Penulis

## BAB I

### PENDAHULUAN

#### I. 1 Latar Belakang

Perkembangan teknologi robot di berbagai bidang industri telah memberikan dampak yang signifikan pada kemajuan industri tersebut. Salah satu bidang industri yang telah menggunakan teknologi robot adalah industri manufaktur. Dalam industri manufaktur, robot digunakan untuk memindahkan objek atau barang dari satu tempat ke tempat yang lain, serta untuk perakitan antar komponen (Shobirin, 2016). Penggunaan robot dalam industri manufaktur ini dilakukan untuk meningkatkan efisiensi dan efektivitas produksi, serta untuk mengurangi biaya produksi.

Selain di industri manufaktur, penggunaan robot juga telah di terapkan di berbagai bidang industri lainnya, seperti di bidang kesehatan, pertanian, dan pertambangan. Di bidang kesehatan, robot digunakan untuk membantu dokter dalam melakukan operasi, serta untuk membantu pasien dalam melakukan terapi fisik[5]. Di bidang pertanian, robot digunakan untuk membantu petani dalam melakukan panen, serta untuk memantau kondisi tanaman(Pranoto dkk, 2021). Di bidang pertambangan, robot digunakan untuk melakukan eksplorasi dan pengambilan sampel di area tambang yang sulit dijangkau oleh manusia (Riyanti dkk, 2021).

Semakin banyaknya aplikasi robot yang digunakan di berbagai bidang industri menuntut kemampuan robot untuk dapat bekerja secara efektif di lingkungan yang semakin kompleks. Hal ini membutuhkan kemampuan robot untuk dapat memahami lingkungan sekitarnya secara menyeluruh yang tidak bisa dicapai dengan cara konvensional. Oleh karena itu, teknik pemetaan dan lokalisasi pada robot menjadi semakin

in penting. Salah satu sensor yang digunakan untuk memperoleh info rmasi tentang lingkungan sekitar robot adalah Kinect depth sensor (Zamisyak dkk, 2016). Kinect depth sensor memberikan informasi 3D tentang lingkungan sekitar robot, yang dapat digunakan untuk memb aungun peta digital dan membantu robot dalam melakukan path planning

.

Untuk membangun sistem mapping dan localization pada robot, d igunakanlah algoritma SLAM (Simultaneous Localization and Mapping) . Algoritma ini merupakan pendekatan yang populer dalam navigasi d an pemetaan robot mobile (Zamisyak dkk, 2016). Dalam penggunaannya , algoritma SLAM menggunakan informasi dari sensor untuk memperkir akan posisi dan orientasi robot dalam peta, serta memperbarui peta saat robot bergerak.

Penelitian mengenai sistem mapping dan localization pada robo t menggunakan Kinect depth sensor dan algoritma SLAM sangat pentin g untuk mengembangkan robot yang dapat bergerak secara otonom dan menghindari rintangan di lapangan. Selain itu, penelitian ini juga dapat memberikan manfaat dalam berbagai aplikasi, seperti di bidan g surveilans, eksplorasi, dan transportasi.

Mobile robot adalah salah satu jenis robot yang membutuhkan k emampuan pemetaan dan lokalisasi yang baik. Mobile robot digunakan untuk bergerak di lingkungan yang berbeda-beda, seperti di dalam ruangan, di luar ruangan, atau di lingkungan yang tidak teratur. Oleh karena itu, mobile robot membutuhkan kemampuan untuk menavigasi lingkungan sekitarnya dengan akurat dan efisien. Teknik pemetaan d an lokalisasi pada mobile robot dapat dilakukan dengan menggunakan sensor signal processing atau image analysis, seperti yang dijelas kan sebelumnya.

Dalam penelitian mengenai sistem mapping dan localization pada robot menggunakan Kinect depth sensor dan algoritma SLAM, diharapkan dapat memberikan kontribusi dalam pengembangan robot yang dapat bergerak secara otonom dan menghindari rintangan di lapangan. Selain itu, penelitian ini juga dapat memberikan manfaat dalam berbagai aplikasi, seperti di bidang surveilans, eksplorasi, dan transportasi. Dengan semakin berkembangnya teknologi robot, diharapkan teknik pemetaan dan lokalisasi pada robot dapat terus ditingkatkan untuk mendukung penggunaan robot di berbagai bidang industri.

## I. 2 Rumusan Masalah

Adapun rumusan masalah penelitian ini sebagai berikut:

- a. Bagaimana perancangan sistem pemetaan dan lokalisasi berbasis algoritma SLAM (*Simultaneous Localization And Mapping*) menggunakan depth sensor Kinect?
- b. Bagaimana perancangan model odometry robot dengan metode sensor fusion berbasis EKF (*Extended Kalman Filter*) dalam menunjang sistem pemetaan dan lokalisasi?
- c. Bagaimana performa sistem pemetaan yang dibangun dengan metode sensor fusion dan AMCL (*Adaptive Monte Carlo Localization*) dapat dipengaruhi dengan parameter lingkungan yang berbeda?

## I. 3 Tujuan Penelitian

Adapun tujuan penelitian ini sebagai berikut:

- a. Merancang sistem pemetaan dan lokalisasi robot berbasis SLAM (*Simultaneous Localization And Mapping*) menggunakan depth sensor Kinect
- b. Melakukan perancangan model odometry robot dengan metode sensor fusion berbasis EKF (*Extended Kalman Filter*) dalam menunjang sistem pemetaan dan lokalisasi
- c. Mengetahui performa sistem pemetaan dan lokalisasi robot yang telah dirancang dalam menghadapi parameter lingkungan yang berbeda

#### 1.4 Manfaat Penelitian

Dari penelitian ini, diharapkan dapat memperoleh manfaat sebagai berikut:

1. **Kontribusi pada Pengembangan Teknologi Robotika:** Penelitian ini dapat menjadi kontribusi penting dalam pengembangan teknologi robotika dengan fokus pada pemetaan dan lokalisasi berbasis algoritma SLAM. Ini dapat menjadi fondasi untuk perkembangan lebih lanjut dalam bidang robotika dan otomasi.
2. **Penggunaan ROS (Robot Operating System):** Penelitian ini akan menunjukkan cara menggunakan ROS sebagai platform pengembangan untuk robot otonomus. Hal ini akan memberikan panduan bagi peneliti dan pengembang lainnya yang tertarik untuk menggunakan ROS
3. **Penerapan Algoritma SLAM:** Dengan mendemonstrasikan implementasi algoritma SLAM dalam konteks pemetaan dan lokalisasi, penelitian ini dapat membantu mengedukasi komunitas robotika tentang kegunaan dan aplikasi algoritma ini dalam situasinya.

4. **Penerapan di Berbagai Sektor Otomasi Industri:** Hasil penelitian ini dapat diadopsi dalam berbagai sektor otomasi industri.
5. **Peningkatan Pendidikan dan Pelatihan:** Penelitian ini dapat digunakan sebagai sumber pembelajaran untuk mahasiswa dan para profesional yang tertarik dalam bidang robotika, ROS, dan SLAM.
6. **Efisiensi dalam Berbagai Aplikasi:** Penggunaan sistem pemetaan dan lokalisasi yang dikembangkan dalam penelitian ini dapat meningkatkan efisiensi dalam berbagai aplikasi seperti logistik, survei, pemantauan, dan banyak lagi.
7. **Kemajuan dalam Riset Kontrol dan Robotika Modern:** Penelitian ini dapat menjadi tonggak penting dalam kemajuan riset kontrol dan robotika modern.

### 1.5 Ruang Lingkup

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

- a. Perancangan sistem pemetaan dan lokalisasi untuk aplikasi indoor pada mobile robot dengan objek yang akan dipetakan adalah objek yang dapat memantulkan sinar inframerah dari sensor.
- b. Pengujian performa sistem SLAM dalam melakukan pemetaan di dasarkan pada perbedaan faktor luminensi dan luas area untuk setiap lingkungan uji.
- c. Perancangan model odometry robot menggunakan 3 source odometry yaitu wheel odometry, IMU dan visual odometry.

## BAB II

### TINJAUAN PUSTAKA

#### 2.1 Penelitian Terkait

1. Sistem Pemetaan Menggunakan Fitur Depth Sensor Kinect pada Mobile Robot untuk Proses Evakuasi Kebakaran Gedung

---

Nama Penulis : ALI UROIDHI  
Tahun Terbit : 2017

Penelitian ini bertujuan untuk membuat sebuah sistem kontrol robot mobile menggunakan Kinect, dengan sistem pemetaan jalan sekaligus. Hasil dari penelitian ini adalah sistem kontrol robot mobile yang dapat melakukan pemetaan jalan dan dapat digunakan untuk proses evakuasi kebakaran gedung.

---

2. Mapping dan Lokalisasi System pada Mobile Robot Menggunakan Metode SLAM Berbasis LiDAR

---

Nama Penulis : Achmad Akmal Fikri  
Lilik Anifah  
Tahun Terbit : 2021

Penelitian ini mengusulkan sistem mapping dan lokalisasi untuk mengenali lingkungan sekitar dengan membuat peta lingkungan menggunakan algoritma SLAM berbasis LiDAR. Hasil dari penelitian ini adalah sistem mapping dan lokalisasi yang mampu mengenali kondisi lingkungan sekitar robot meskipun masih terdapat gangguan pada peta.

---

### 3. Pemetaan Lingkungan Multi Robot dengan Algoritma ORB SLAM-2

---

Nama Penulis :	Penelitian ini bertujuan untuk membuat rancangan sistem pemetaan lingkungan multi-robot dengan menggunakan algoritma ORB SLAM-2. Hasil dari penelitian ini adalah sistem desentralisasi sehingga algoritma dijalankan pada kedua robot. Kemudian setiap robot melakukan pemetaan lingkungannya dan mengirimkannya ke komputer agar dapat divisualisasi.
Tahun Terbit :	2020

---

### 4. Simultaneous Localization and Mapping for Autonomous Robot Navigation

---

Nama Penulis :	Penelitian ini memperkenalkan teknik baru berbasis visi komputer untuk SLAM (Simultaneous Localization and Mapping) menggunakan Extended Kalman Filter untuk menghasilkan peta lingkungan dan navigasi secara efisien dalam lingkungan secara otomatis. Algoritma yang diusulkan dalam artikel ini secara efektif menghasilkan jalur terpendek yang bebas tabrakan untuk me
Tahun Terbit :	2021

---

---

ncapai tujuan. Sensor yang digunakan dalam proyek ini adalah sensor Kinect, yang merupakan sensor yang ekonomis dan dilengkapi dengan kamera dan sensor kedalaman yang menggunakan sinar inframerah. Kinerja navigasi otomatis juga disajikan dalam hal akurasi mengikuti jalur dan menghindari rintangan.

---

## 5. Sistem Navigasi Pada Prototipe Robot Kursi Beroda Untuk Penyandang Disabilitas

---

Nama Penulis :	Penelitian ini bertujuan untuk membuat prototipe sebuah robot kursi roda yang mampu melakukan navigasi dengan memanfaatkan depth kamera Kinect untuk melakukan skeleton tracking dalam sistem pandu arahnya. Hasil dari penelitian ini menunjukkan kemampuan sensor Kinect untuk di terapkan dalam dunia robotika
Tahun Terbit :	

---

### 2.2 Robot

Istilah robot berasal dari bahasa Cekoslowakia, “Robota” yang berarti “Kerja Cepat”. Istilah ini muncul pada tahun 1920 oleh seorang pengarang sandiwara bernama Karel Capek (Indrawan, 2008). Robot merupakan suatu sistem atau alat yang dapat berperilaku at-

au meniru perilaku manusia dengan tujuan untuk mengantikan dan mempermudah kerja/aktivitas manusia.

Untuk dapat diklasifikasikan sebagai robot, maka robot harus memiliki dua macam kemampuan, yaitu: (1) Bisa mendapatkan informasi dari sekelilingnya; dan (2) Bisa melakukan sesuatu secara fisik, seperti bergerak atau memanipulasi objek. Supriyanto juga menambahkan bahwa sebuah robot dapat saja dibuat untuk berbagai macam aktivitas, namun sebuah robot harus dibuat dengan tujuan untuk kebaikan manusia. Ada beberapa fungsi robot, sehingga manusia memerlukan kehadirannya yaitu: (1) Meningkatkan produksi, akurasi, dan daya tahan. Robot banyak digunakan di industri; (2) Untuk tugas-tugas yang berbahaya, kotor, dan beresiko. Robot digunakan ketika manusia tidak mampu masuk ke daerah yang beresiko, seperti robot untuk menjelajah planet, robot untuk mendeteksi limbah nuklir, robot militer, dll; (3) Untuk pendidikan. Banyak robot yang digunakan untuk menarik pelajar agar tertarik mempelajari teknologi. Seperti pengenalan dunia teknologi di bidang robotika, misal mengajar anak dalam pembuatan robot lego, dll; (4) Untuk membantu manusia, khususnya dalam aktivitas sehari-hari. Robot yang berperan dalam hal ini biasanya dikenal sebagai Robot Sosial (SocialRobot). Seperti robot pelayan (Service Robot) yang banyak dikembangkan tamanya di rumah sakit (Supriyanto dkk, 2010).

Berdasarkan dari penjelasan di atas, maka sudah jelas persoalan mengapa manusia memerlukan kehadiran dari robot. Karena keterbatasan kemampuan yang dimiliki oleh manusia sehingga kehadiran robot sangat dibutuhkan dunia saat ini.

### 2.3 *Mobile Robot*

Mobile Robot adalah jenis robot yang memiliki kemampuan untuk berpindah tempat dari satu tempat ketempat yang lain. Menurut Anshar (2010, h. 2) jika ditinjau dari jenis aktuatorrrnya, robot dibagi atas dua jenis, yaitu robot berkaki dan robot beroda.



Gambar 1 Mobile Robot Menggunakan Caterpillar Track



Gambar 2 Mobile Robot Berkaki



Gambar 3 Mobile Robot Beroda

Pada robot beroda memungkinkan semua aktuator menyentuh permukaan tanah, hal ini memberikan keunggulan pada robot beroda dalam menjaga keseimbangan agar tidak terjatuh pada saat melakukan pergerakan. Berbeda dengan robot berkaki yang mengharuskan aktuatornya mampu mendukung beban dari robot agar tetap berada dalam kondisi yang seimbang, hal ini dikarenakan pada saat melakukan pergerakan, salah satu aktuator (kaki) dari robot harus bertumpu pada permukaan, sehingga dalam desain mekaniknya diperlukan kecermatan dan perhitungan yang lebih untuk menghasilkan pergerakan robot yang seimbang, berbeda dengan robot beroda yang cukup mudah dalam menentukan titik seimbangannya (Anshar, 2010).

Dalam klasifikasinya, konfigurasi tipe roda pada mobile robot dibedakan oleh beberapa jenis, hal ini tergantung dari pemilihan jenis, model, dan fungsi dari robot itu sendiri, ditinjau dari tiga hal yakni: (1) Kemampuan manuver dari robot; (2) Kemampuan dari robot dalam mengontrol pergerakannya, dan (3) Kemampuan robot dalam menjaga kestabilannya (Goris dkk, 2005)

Untuk dapat membuat sebuah robot mobile minimal diperlukan pengetahuan tentang mikrokontroler dan sensor-sensor elektronik. Sensor digunakan untuk mengenali lingkungan pada robot sehingga robot dapat menghindari halangan yang ada di sekitar. Untuk robot bersensor biasanya berperan sebagai AMR atau autonomous mobile robot, selain itu juga ada robot menggunakan sensor tapi tanpa perlu kontrol cerdas untuk menentukan jalur jalannya yaitu AGV atau autonomous guided vehicle dimana robot ini menggunakan sensor untuk mengikuti jalur yang telah ditentukan berupa garis terang gelap (line follower robot), dinding (wallfollower robot), ataupun cahaya. Ada juga robot yang tanpa menggunakan sensor, namun digerakkan menggunakan kont

roller yang dijalankan oleh manusia, robot ini sekarang sudah banyak dijual dipasarkan berupa mainan menggunakan remot kontrol.

Sensor yang digunakan pada robot juga ada banyak macam, baik laser, sonar, kamera, dan juga gps. Pada robot-robot canggih juga terdapat sensor kemiringan tanah agar tidak terjatuh dalam melewati jalur-jalur tertentu.

#### ***2.4 Robot Operating System***

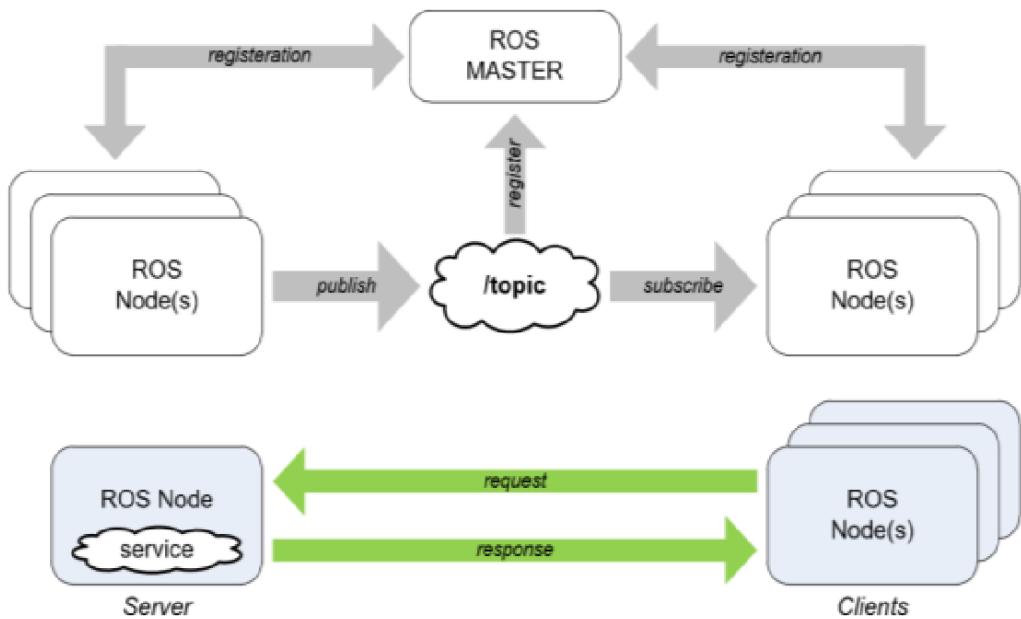
ROS adalah framework untuk membuat program robot yang fleksibel dan mudah digunakan. ROS berisi beragam peralatan (tool), library, driver, dan konvensi yang bertujuan untuk memudahkan pembuatan program yang kompleks dan handal pada berbagai platform robot. Seharusnya, dengan adanya ROS, setiap programmer menggunakan style yang sama dalam membuat sebuah program untuk robot jenis apapun.

Robot Operating System sebenarnya adalah meta-operating system atau framework yang bersifat opensource yang dapat digunakan untuk robot. ROS mempunyai sebuah service seperti halnya sebuah sistem operasi pada umumnya, termasuk abstraksi perangkat keras, kendali perangkat tingkat bawah, implementasi dari fungsi-fungsi yang biasa digunakan, penyampaian pesan/data diantara proses serta management package. ROS juga menyediakan alat dan library yang memungkinkan kita untuk dapat membangun, memprogram hingga menjalankan program melalui banyak computer.

Software pada ROS dapat dibagi menjadi tiga grup: tools untuk mendistribusikan perangkat lunak berbasis ROS, implementasi ROS client library seperti rosCPP, rospy, roslisp, dan yang terakhir package yang berisi kode yang berhubungan dengan aplikasi fungsional robot (Wicaksono, 2019).

Client libraries ROS utamanya diperuntukkan untuk sistem UNIX . Untuk hal ini Ubuntu Linux terdaftar sebagai “supported” sedangkan varian lain seperti Fedora Linux, macOS, serta Windows terdaftar sebagai “experimental” .

Kemampuan ROS multi-node dapat mengontrol banyak fungsi operasional robot. Robot yang memiliki tugas besar menjalankan banyak fungsi dan canggih, penerapan cara tradisional akan menyita waktu dan energi para peneliti. Dengan multi-node, properti ROS seperti yang ditunjukkan pada Gambar 4 akan memberikan kontrol aliran robot yang efisien. Selanjutnya, proses membangun perangkat lunak robot tidak butuh waktu lama karena dapat dilakukan secara tim.



Gambar 4 Prinsip Kerja Kerangka Perangkat Lunak Berbasis ROS untuk Mengontrol Operasi Semua Sistem Robot Multi-Node

(Sumber : Nurdin & Hendriyawan 2019)

#### 2. 4. 1 *ROS Node*

Sebuah node dalam ROS adalah sebuah proses yang melakukan suatu komputasi. berbagai node ini terkombinasi bersama menjadi suatu

graph dan berkomunikasi dari satu node ke node lain menggunakan aliran topics, services, dan parameter server. Node ini diperuntukkan untuk beroperasi pada level yang sangat sempit. Sebagai contoh sebuah sistem kendali robot biasanya terdiri dari banyak node: satu node mengendalikan laser range finder, satu node mengendalikan motor, satu node lokalisasi, satu node untuk perencanaan jalur (path planning), satu node untuk monitoring, dan seterusnya.

Penggunaan node ini memberi manfaat antara lain: toleransi kegagalan yang terisolasi pada satu fungsionalitas tertentu pada individu node, kompleksitas dari kode berkangurang dibanding sebuah sistem monolithic, reusability serta maintainability tinggi dengan penyederhanaan berdasarkan fungsionalitas-fungsionalitas tersebut.

Selain itu sebuah node dapat dikembangkan menjadi sebuah nodelet, yaitu sebuah node yang dapat dipanggil dari dalam kode program lainnya menggunakan modul nodelet manager.

#### 2.4.2 *ROS Topic*

Topic adalah komponen yang berfungsi sebagai media pertukaran pesan (disebut message dalam sistem ROS) antar node yang dinamai sesuai isi dari pesan tersebut. Sistemnya menggunakan anonymous publish/subscribe semantics. Nodes yang membutuhkan informasi dari suatu topic akan melakukan subscribe dan node yang menyediakan informasi akan melakukan publish. Jumlah node yang melakukan subscribe ataupun publish pada suatu topic tidak terbatas jumlahnya.

Setiap topic memiliki tipe datanya masing-masing sesuai tipe ROS message yang di bawanya. Kita dapat membuat tipe message sendiri sesuai kebutuhan kita, atau menggunakan tipe message yang disedi-

akan oleh ROS client library. ROS topic juga mendukung komunikasi via TCP/IP atau UDP.

#### **2. 4. 3 *ROS Package***

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

#### **2. 5 *Telerobotic***

Bruno Siciliano dan Oussama Khatib dalam buku yang berjudul *Springer Handbook Of Robotic*, 2008 menuliskan Guenter Niemeyer dkk (2008), menyatakan. Telerobotika (tele-robotics) adalah cabang dari robotika yang melibatkan pengendalian atau manipulasi robot dari jarak jauh, sering kali menggunakan teknologi telekomunikasi, seperti jaringan komputer atau koneksi internet. Ini berbeda dengan sistem robot biasa yang biasanya beroperasi dalam jarak dekat atau bahkan dalam satu lokasi fisik dengan operatornya, tiga kategori utama, yaitu :

- *Direct Control*
- *Shared Control*
- *Supervisory Control*

##### *a. Direct control*

Kontrol langsung adalah mode operasi telerobotika di mana seorang operator manusia secara langsung mengendalikan robot dengan menggunakan perangkat input seperti joystick, pengend

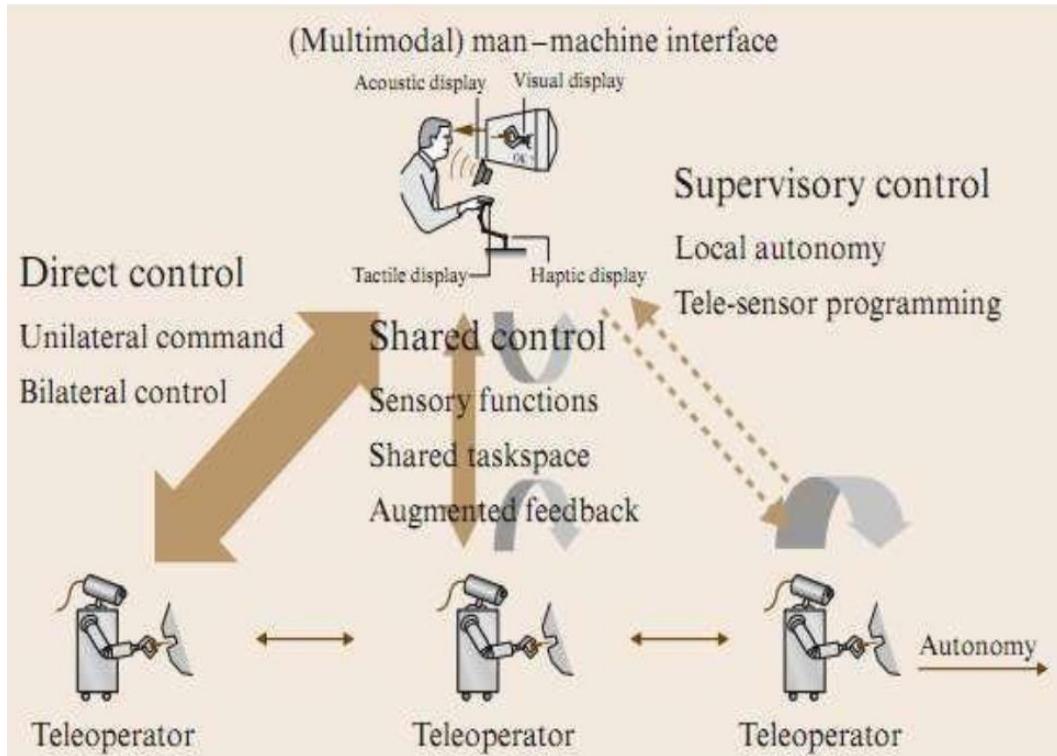
ali jarak jauh, atau perangkat lainnya. Dalam mode ini, setiap gerakan yang dilakukan oleh operator manusia segera direplikasi oleh robot. Kontrol langsung berguna dalam situasi di mana operator membutuhkan tingkat kontrol yang tinggi, seperti dalam operasi bedah telerobotik atau dalam situasi berbahaya yang memerlukan reaksi instan.

*b. Shared Control*

Dalam mode kontrol bersama, kontrol robot dibagi antara operator manusia dan sistem otomatis. Operator manusia memberikan arahan dan input, tetapi sistem otomatis juga memainkan peran dalam mengarahkan robot. Tujuannya adalah mencapai kesimbangan antara kontrol manusia dan otomatis untuk mengoptimalkan kinerja robot. Kontrol bersama sangat berguna dalam situasi di mana operator manusia mungkin tidak memiliki pengetahuan atau keahlian teknis yang cukup untuk mengendalikan robot sepenuhnya, atau di mana tugasnya sangat kompleks.

*c. Supervisory Control*

Mode kontrol pengawasan adalah mode di mana operator manusia bertindak sebagai pengawas atau supervisor atas tugas yang dilakukan oleh robot. Operator mengendalikan robot secara lebih tinggi, memberikan instruksi tingkat tinggi atau tujuan, sementara sebagian besar kontrol tugas harian diberikan kepada sistem otomatis. Kontrol pengawasan sering digunakan dalam aplikasi industri dan militer di mana robot harus bekerja secara mandiri dengan sedikit intervensi manusia, tetapi manusia tetap bertanggung jawab atas pengawasan umum dan pengambilan keputusan.



Gambar 5 *Man-Machine Interface*

(Sumber : Springer Handbook of Robotics, 2008)

## 2.6 Robot Preseption

Persepsi robot adalah kemampuan robot untuk mengambil informasi dari lingkungan sekitarnya melalui sensor atau kamera dan mengolah informasi tersebut untuk mengambil keputusan atau melakukan tindakan tertentu. Persepsi robot dapat berupa sistem persepsi yang berfungsi untuk mendeteksi objek tertentu pada lingkungan sekitarnya, seperti bola pada lapangan sepak bola (Mahandi, 2021), atau sensor yang mendukung pergerakan robot, seperti Ultrasonic PING sensor yang digunakan sebagai Wall Following pada mobile robot (Prasetyo dkk, 2021). Selain itu, persepsi robot juga dapat berupa perlakuan hewani yang menanggapi cahaya, suhu, suara, sentuhan dan postur, seperti pada robot terapi bagi lanjut usia yang mengalami demensia (Firmansyah, 2015). Dalam pengembangannya, sistem persepsi robo

t dapat menggunakan berbagai teknologi, seperti metode filtering, metode Fuzzy, atau algoritma greedy.

Sistem persepsi robot dapat memberikan informasi tentang lingkungan sekitarnya, seperti posisi objek atau landmark tertentu, yang dapat digunakan sebagai input untuk algoritma SLAM (Utomo, 2015). Dalam penelitian yang menggunakan HBE Robocar, robot menggunakan landmark berupa bola untuk menentukan posisi dan diproses menggunakan algoritma particle filter dengan variasi jumlah partikel.

Algoritma SLAM dapat digunakan untuk membuat peta lingkungan: Algoritma SLAM dapat digunakan untuk membuat peta lingkungan yang akurat dan detail. Dalam penelitian yang menggunakan mobile robot dengan sensor LiDAR, algoritma SLAM digunakan untuk membuat peta lingkungan menggunakan metode google cartographer yang dikombinasikan dengan metode eulerdometry. Peta lingkungan ini dapat digunakan sebagai referensi untuk navigasi robot.

Sistem persepsi robot dan algoritma SLAM dapat digunakan untuk navigasi robot, seperti pada mobile robot dengan penggerak mekanum yang menggunakan Ultrasonic PING sensor sebagai Wall Following. Dalam penggunaannya, Ultrasonic PING sensor dapat digunakan dengan metode Fuzzy untuk mendapatkan jarak sesuai dengan setpoint yang ditentukan dan meminimalkan kesalahan. Algoritma SLAM juga dapat digunakan untuk perencanaan rute gerak robot dari titik awal menuju titik akhir.

## 2.7 Odometry

Odometry terdiri dari kata Yunani yaitu odos yang berarti rute dan metron yang berarti ukuran. Odometry adalah penggunaan data dari sensor gerak yang digunakan untuk memperkirakan perubahan posisi

i dari waktu ke waktu. Sistem ini digunakan pada beberapa robot ber kaki maupun beroda untuk memperkirakan posisi mereka yang relative terhadap posisi awal. Metode ini sangat peka terhadap kesalahan karena terintegrasi terhadap variabel kecepatan dari waktu ke waktu untuk memberikan perkiraan posisi. Hasil analisis menghasilkan persamaan berikut.

$$\begin{bmatrix} X & New\ Robot \\ Y & New\ Robot \end{bmatrix} = \begin{bmatrix} X & Previous\ Robot \\ Y & Previous\ Robot \end{bmatrix} + \begin{bmatrix} \cos\theta_{robot} & -\sin\theta_{robot} \\ \sin\theta_{robot} & \cos\theta_{robot} \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} \dots (1)$$

Odometry membutuhkan sistem elektronik penunjang, bagian penting dari sistem elektronik penunjang odometry adalah sensor pergerakan. Sensor pergerakan yang bisa digunakan untuk sistem odometry antara lain kamera, sensor inersia, maupun rotary encoder.

Odometry dapat dikatakan sebagai suatu sistem pengukuran yang menggunakan data-data dari aktuator untuk memperkirakan pergerakan dari robot (Dudek, 2008). Pada robot dalam penelitian ini, odometry gerakan translasi robot dapat dilakukan menggunakan putaran dari roda yang posisinya sudah diketahui secara pasti. Sedangkan untuk odometry rotasi robot dapat dilakukan menggunakan sensor gyroscope yang posisi awalnya sudah diketahui.

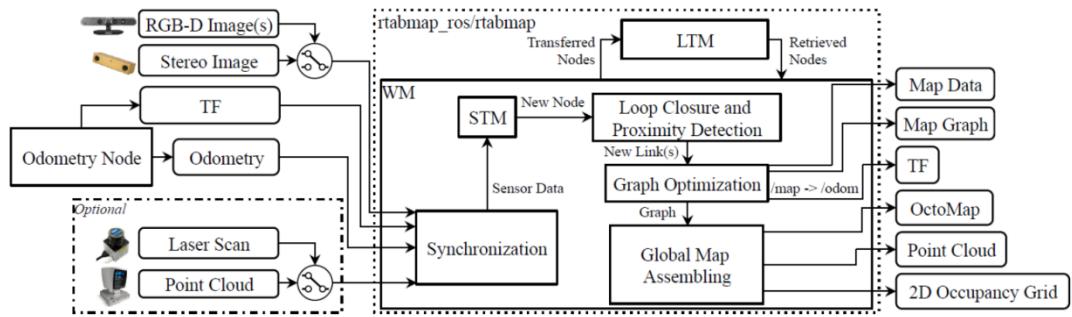
## 2.10 SLAM

SLAM (Simultaneous Localization And Mapping) adalah suatu metode yang digunakan sebuah robot untuk membuat suatu peta sekaligus bisa melokalisasikan dirinya di peta yang dibuatnya. Metode ini memungkinkan robot untuk mengerjakan dua tugas secara bersamaan. yaitu memetakan lokasi serta mengetahui posisinya di dalam peta yang dibuatnya (Prayoga et al., 2010). Metode SLAM ini banyak sekali digunakan pada pengembangan mobile robot dengan kasus pembacaan suatu tempat yang memiliki bidang yang rata. Agar dapat membaca dengan b

aik, robot harus dilengkapi dengan bermacam sensor pendekripsi sebagaimana indra pembacanya. Bisa memakai LIDAR maupun SONAR (Sidharta, 2019). Terdapat berbagai cara yang bisa digunakan untuk mengaplikasikan algoritma SLAM secara 2D, yaitu Hector SLAM, GMapping, Karto SLAM, Core SLAM, dan Lago SLAM. Algoritma SLAM dapat dikelompokan menjadi 3 bagian, yang pertama berdasarkan sensor yang digunakan untuk mengukur jarak, bisa menggunakan laser, ultrasound atau odometry, kedua berdasarkan metode perhitungan, bisa menggunakan algoritma kalman filter atau partikel filter, dan yang ketiga berdasarkan strukturnya (Utomo, 2015). Algoritma SLAM ini semuanya terdapat pada package Robot Operating System (Rahman, 2020).

## 2.11 Real-Time Appearance-Based Mapping

RTAB-Map adalah pendekatan SLAM berbasis grafik yang populer yang mengimplementasikan deteksi *loop closure* visual menggunakan BoW (Bag of Words) inkremental. Algoritma ini dapat mengambil data LiDAR, RGB-D, dan kamera stereo sebagai input, memungkinkan berbagai mode operasi seperti pemetaan 6 Dof dan 3 Dof. Salah satu kontribusi utama dari RTAB-Map adalah implementasi sistem pengelolaan memori yang memungkinkan operasi pada lingkungan skala besar sambil mempertahankan pemetaan jangka panjang. RTAB-Map juga kompatibel dengan ROS (Robot Operating System) dan mendapat dukungan besar dari komunitas ilmiah dalam satu dekade terakhir ini.



Gambar 6 Blok Diagram RTAB-Map pada node ROS

(Sumber : Labb   and Michaud, 2019)

Pada Gambar diatas blok fungsional berbeda dari RTAB-Map dapat diamati, seperti yang disajikan dalam (Labb   & Michaud, 2019). Di sini jelas bahwa algoritma ini memerlukan setidaknya sumber odometri dan input gambar RGB-D atau stereo. Input sensor Laser berfungsi sebagai input data jarak opsional yang dapat berupa laser scanner 2D atau pointclouds 3D dan dapat digunakan untuk membentuk grid okupansi 2D atau 3D sesuai kebutuhan. RTAB-Map juga mencakup node odometri jika tidak ada odometri eksternal yang disediakan. Blok utama RTAB-Map berisi sistem pengelolaan memori dan blok deteksi loop closure, fixture detection, optimisasi grafik, dan modul perakitan peta global. RTAB-Map mampu menghasilkan beberapa output secara *realtime*: peta yang dihasilkan dalam berbagai format (OctoMap, grid okupansi, dan pointclouds 3D), koreksi odometri dalam bentuk tf (format ROS untuk transformasi), serta grafik yang dihasilkan dengan atau tanpa informasi sensor.

### 2. 11. 1 Node Odometry RTAB-Map

RTAB-Map mengimplementasikan sebuah node odometri yang mampu menghitung odometri visual (VO) atau odometri LiDAR (LO) jika tidak ada sumber eksternal odometri, seperti odometri roda atau sumber V0 atau LO lainnya. Node ini menghitung odometri berdasarkan metoda

logi yang disajikan dalam Scaramuzza dan Fraundorfer (2011), menghitung F2F dan F2M dalam kasus V0, serta S2S dan S2M dalam kasus L0

#### A. Odometry Visual

Untuk pendekatan odometri visual, diperlukan transformasi antara kerangka koordinat kamera dan kerangka robot serta su mber gambar stereo atau RGB-D. Gambar input digunakan untuk mendapatkan fitur visual. Kemudian, fitur-fitur ini akan melalui langkah pencocokan fitur yang membandingkan fitur yang diperoleh dalam bingkai gambar saat ini dengan bingkai kunci sebelumnya atau peta fitur yang tersimpan, yaitu F2F dan F2M secara berturut-turut. Setelah fitur-fitur dipasangkan, dilakukan perhitungan prediksi gerakan menggunakan PnP RANSAC, yang kemudian dioptimalkan dengan penyesuaian bundel lokal. Akhirnya, posisi yang telah dioptimalkan beserta ketidakpastiannya diberikan sebagai output dari node odometri. Jika posisi baru cukup signifikan (ditentukan oleh ambang batas pada jumlah inlier dalam estimasi gerakan), fitur-fitur yang diekstrak ditambahkan ke peta fitur untuk pencocokan fitur selanjutnya. Perlu diperhatikan bahwa peta fitur memiliki jumlah maksimum fitur yang tersimpan setelah itu fitur-fitur yang lebih lama akan dibuang, jadi ini bukanlah algoritma pemetaan jangka panjang.

#### B. Odometry LiDAR

Odometri LiDAR sangat mirip dengan yang disajikan dalam odometri visual. Dalam kasus ini, data sensor laser harus diberikan sebagai input bersama dengan transformasi antara kerangka koordinat LiDAR dan kerangka dasar robot. Selain itu, o

dometri eksternal seperti odometri roda dapat ditambahkan untuk meningkatkan prediksi gerakan, karena odometri LiDAR dapat kehilangan jejaknya jika tidak cukup fitur terdeteksi. Langkah pertama adalah menyaring data awan titik, setelah itu melanjutkan dengan registrasi ICP (Iterative Closest Point) dari awan titik saat ini dalam bingkai kunci sebelumnya atau dalam peta, yaitu S2S atau S2M secara berturut-turut. Dalam hal ini, peta adalah awan titik yang terbentuk dari semua pemindai yang terdaftar dari setiap bingkai kunci. Setelah registrasi ICP, posisi odometri berikutnya diperoleh dan diberikan sebagai output dalam bentuk transformasi yang diperbarui dan posisi dengan ketidakpastian. Sama seperti pendekatan odometri visual, jika rasio korespondensi bingkai saat ini berada di bawah ambang batas yang telah ditentukan, bingkai baru dianggap sebagai bingkai kunci.

### **2. 11. 2 Graph creation and data storage**

Seperti yang disebutkan di atas, RTAB-Map adalah sistem SLAM berbasis grafik. Ini berarti bahwa sistem ini membuat ulang *environment* menggunakan node dan *link*.

#### **Node dan Link dalam RTAB-Map**

Node adalah titik tertentu dalam waktu di mana data sensor diterima. Frekuensi sensor yang dipasang pada robot menentukan frekuensi pembuatan node baru. Dalam RTAB-Map, semua data masukan disinkronkan dan disimpan dalam node yang sama, yang sekarang akan dibutuhkan sebagai (signatures). Menurut Labb   dan Michaud (2018), setiap signature dapat menyimpan (tergantung pada data sensor yang tersedia) :

- ID : Unique timestamp
- Weight : tingkatan signature
- BoW : bag of words, penanda visual yang digunakan untuk deteksi loop closure dan pembaruan weight, yang dikenal dengan istilah image signature
- Occupancy Grid
- Sensor data :
  - Pose : input dari odometry
  - Gambar RGB : digunakan untuk menangkap fitur lingkungan
  - Gambar Depth : digunakan untuk menangkap posisi 3D
  - laser scanner : Digunakan untuk transformasi loop closure dan perbaikan odometri, serta oleh modul Deteksi Proksimitas

Semua signature saling terhubung oleh yang disebut "link," yang menggambarkan transformasi antara signature dan ketidakpastian pengukuran sebagai matriks informasi. *Link-link* ini dapat disimpan dalam 3 Derajat Kebebasan ( $x, y, \theta$ ) atau 6 Derajat Kebebasan ( $x, y, z, \text{pitch}, \text{yaw}, \text{roll}$ ), tergantung apakah ruangnya digambarkan dalam 2D atau 3D, menggunakan koordinat Euklides. Semua *link* mengandung informasi yang sama, tetapi RTAB-Map memiliki konvensi penamaan tergantung pada modul yang menyediakan link-link ini.

- *Link* tetangga (Neighbour link): Dibuat antara Signature baru dan yang sebelumnya.
- *Link* kedekatan (Proximity link): Ditambahkan ketika dua signature yang dekat diselaraskan bersama menggunakan data LiDAR dan penyelarasannya pemindaian.

- *Link loop closure* (*Loop closure link*): Ditambahkan ketika loop closure terdeteksi antara signature baru dan satu dalam peta.
- Kendala landmark (*Landmark constraint*): *Link* ini tidak didefinisikan secara eksplisit dalam dokumentasi resmi RTAB-Map, tetapi hadir dalam fitur baru algoritma ini. Ini mungkin bukan link dalam definisi formal yang kita miliki, tetapi merupakan constrain yang menghubungkan sebuah signature dengan landmark yang dapat diidentifikasi. Hal ini dapat dilakukan dengan menggunakan penanda Aruco atau tag April, yang deteksinya dimungkinkan dalam versi terbaru RTAB-Map sejak tahun 2019
- .

## **Memory Management**

Sistem pengelolaan memori dirun di atas core RTAB-Map. Sistem ini bertanggung jawab atas pengelolaan cara penyimpanan posisi dan data sensor serta pemilihan mana yang digunakan untuk melakukan deteksi kedekatan dan loop closure. Sistem pengelolaan memori dibagi menjadi tiga memori utama: Memori Jangka Pendek (STM), Memori Kerja (WM), dan Memori Jangka Panjang (LTM). Seperti yang dijelaskan di Mark te Brake (2021), dua tujuan utama dari perbedaan ini adalah: untuk memisahkan data yang baru saja diperoleh dari yang digunakan dalam deteksi loop closure dan untuk menjaga jumlah signature yang rendah selama deteksi loop closure agar tetap sesuai dengan batasan *realtime*.

- STM (Memori Jangka Pendek): Ini dapat dianggap sebagai buffer berukuran tetap yang memproses dan menyimpan signature yang paling baru sehingga mereka tidak memengaruhi deteksi *loop closure*. Menurut Labb   dan Michaud (2019), di sini lah grid okupansi

dihitung dan semua informasi signature dirangkum. Menurut Labb  e dan Michaud (2013), ada memori sebelumnya yang disebut Memori Sensorik (SM) di mana ekstraksi fitur dan reduksi fitur dilakukan sebelum masuk ke STM. Fitur-fitur yang diekstraksi diubah menjadi kata-kata visual menggunakan metode BoW inkremental (BoW).

- WM (Memori Kerja): Memori ini berisi semua signature yang merupakan kandidat untuk *loop closure*. Memori ini biasanya disimpan di dalam RAM, sehingga hanya signature yang bukan kandidat untuk transfer yang tetap ada di dalam WM untuk mengurangi penggunaan memori dan waktu pemrosesan.
- LTM (Memori Jangka Panjang): Memori ini menyimpan semua signature yang bukan signature baru maupun kandidat untuk *loop closure*. Menurut Labb  e dan Michaud (2013), signature disimpan dalam database yang berisi *link*, ID signature, dan signature itu sendiri.

#### 2.11.4 Deteksi Loop Closure

Deteksi *loop closure* dalam RTAB-Map didasarkan pada pendekatan yang umumnya dikenal sebagai Bag of Visual Words (BoVW). Dalam metodologi ini, fitur dari setiap bingkai kunci yang terdeteksi digabungkan menjadi BoW, dan setiap bingkai kunci diubah menjadi representasi menggunakan kata-kata visual yang dihasilkan. Filter Bayesian diskrit digunakan untuk melacak *loop closure* dengan mengestimasi probabilitas bahwa lokasi saat ini berkaitan dengan *signature* yang tersedia di WM. Jika probabilitasnya lebih tinggi dari ambang batas tertentu, hipotesis ini dianggap sebagai kandidat *loop closure* dan grafik posisi diperbarui sebagai konsekuensinya.

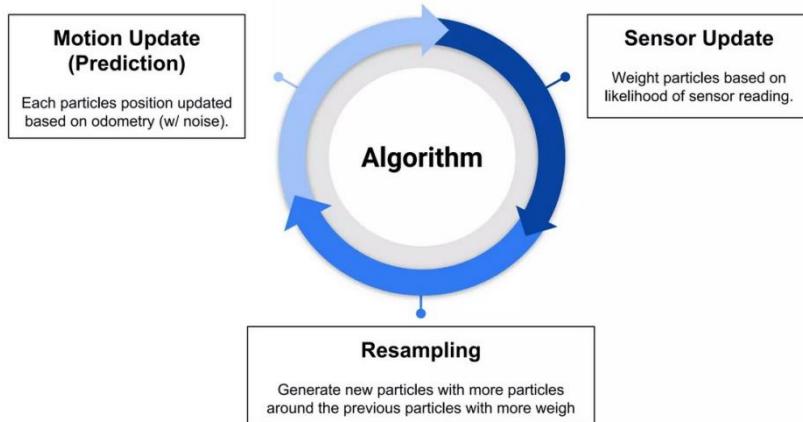
## 2.11.5 Optimization

Dalam RTAB-Map, model word di dalam Memori Kerja (WM) dioptimalkan untuk secara akurat merepresentasikan lintasan yang dilakukan oleh robot. Untuk melakukan optimasi, informasi posisi diekstraksi dari *signature* dan dimasukkan ke dalam pemoptimalan nonlinear. Saat ini, RTAB-Map mengimplementasikan beberapa optimizer yang paling populer seperti g2o, gtsam, dan TORO.

## 2.12 Adaptive Monte Carlo Localization

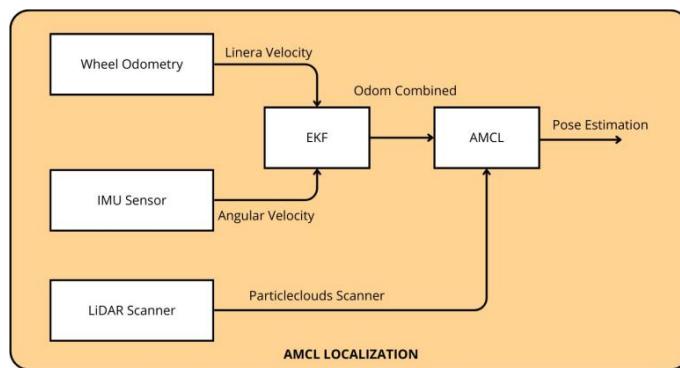
Adaptive Monte Carlo Localization (AMCL) adalah sistem lokalisasi probabilistik untuk robot yang bergerak di bidang 2D (Thrun dkk, 2005). AMCL merupakan varian dari Monte Carlo Localization (MCL) yang menggunakan filter partikel untuk memperkirakan lokasi dan orientasi robot. Algoritma AMCL dimulai dengan keyakinan awal di distribusi probabilitas pose robot, yang direpresentasikan oleh partikel yang didistribusikan sesuai dengan keyakinan tersebut. Algoritma kemudian menggunakan langkah-langkah berikut untuk memperkirakan pose robot:

- Gerakan: Robot bergerak dan memperbarui perkiraan pose berdasarkan data odometer.
- Sensor: Robot merasakan lingkungannya menggunakan sensor seperti pemindai laser dan memperbarui perkiraan pose berdasarkan kemungkinan data sensor.
- Resampling: Partikel diresampling berdasarkan bobot mereka untuk menghasilkan set partikel baru untuk iterasi berikutnya
- .
- Filter partikel: Partikel diberi bobot berdasarkan kemungkinan data sensor dan data odometer, dan perkiraan pose diperbaui berdasarkan partikel yang diberi bobot.



Gambar 7 Alur Kerja AMCL

AMCL digunakan dalam ROS untuk lokalisasi dalam SLAM[1][5]. AMCL adalah bagian dari tumpukan navigasi ROS dan digunakan untuk memperkirakan pose robot dalam peta yang diketahui dari lingkungan. Algoritma AMCL menyesuaikan jumlah partikel berdasarkan jarak KL untuk memastikan bahwa distribusi partikel konvergen ke distribusi yang benar dari keadaan robot berdasarkan semua pengukuran sensor dan gerakan masa lalu dengan probabilitas tinggi.



Gambar 8 Blok Diagram Arsitektur AMCL

## BAB III

### METODE PENELITIAN

#### 3.1 Waktu dan Lokasi Penelitian

Pelaksanaan penelitian tugas akhir ini terhitungan sembilan bulan tepatnya pada bulan Januari–September 2023. Dengan rincian kegiatan sebagai berikut:

Tabel 1 Jadwal kegiatan penelitian

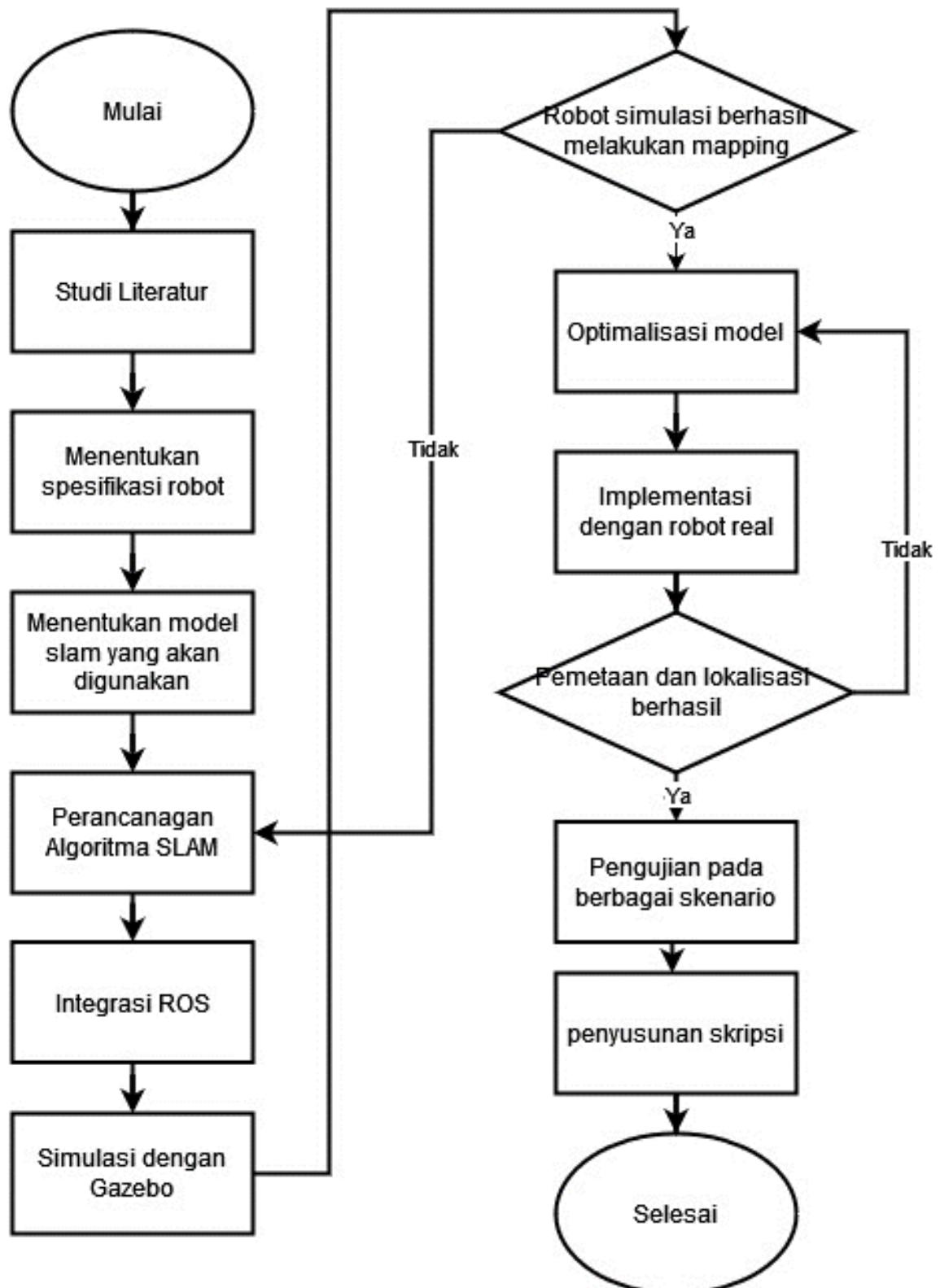
No	Jenis Kegiatan	Bulan								
		1	2	3	4	5	6	7	8	9
1	Studi literatur									
2	Persiapan alat dan bahan									
3	Desain produk									
4	Rancang bangun <i>low level</i> robot									
5	Rancang bangun <i>high level</i> robot									
6	Pengujian pemetaan dan lokalisasi									
7	Pengambilan dan olah data									
8	Penyusunan skripsi									

Adapun lokasi pelaksanaan penelitian ini bertempat di Laboratorium Sistem Kendali dan Instrumentasi Departemen Teknik Elektro Fakultas Teknik Universitas Hasanuddin.

#### 3.2 Rancangan Umum Penelitian

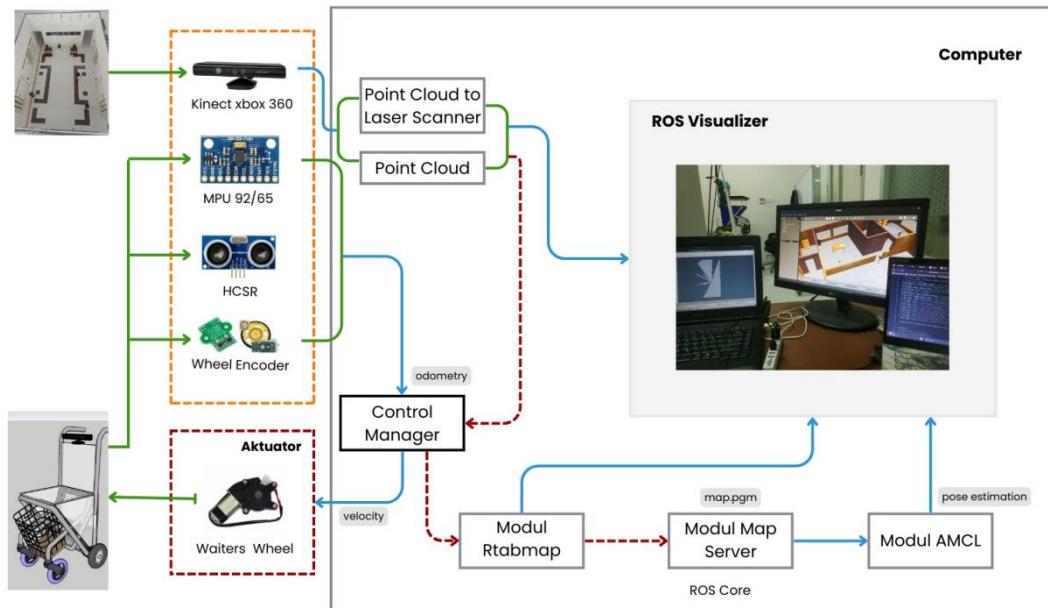
Pada tahapan perancangan umum sistem pemetaan dan lokalisasi menggunakan algoritma slam yang di implementasikan melalui modul *framework Robot Operating System* (ROS) sebagai robot *middleware*. Sec

ara keseluruhan alur pelaksanaan penelitian ini dapat dilihat pada Gambar 9.



Gambar 9 Diagram Alir Rancangan Penelitian

Tahapan perancangan sistem navigasi robot otonom Waiter-Bot m  
eliputi aspek perangkat keras dan perangkat lunak di mana blok diag  
ram secara keseluruhan dapat dilihat pada Gambar 10.



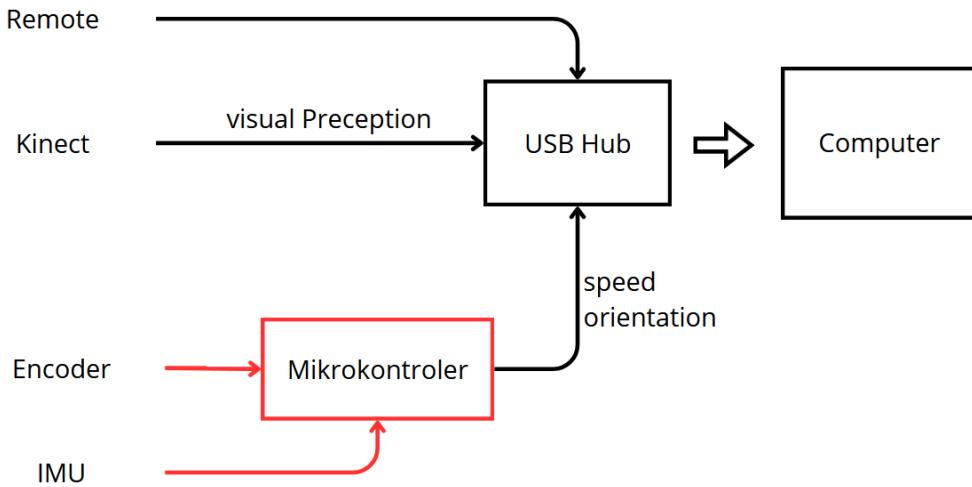
Gambar 10 Sistem Pemetaan dan Lokalisasi

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



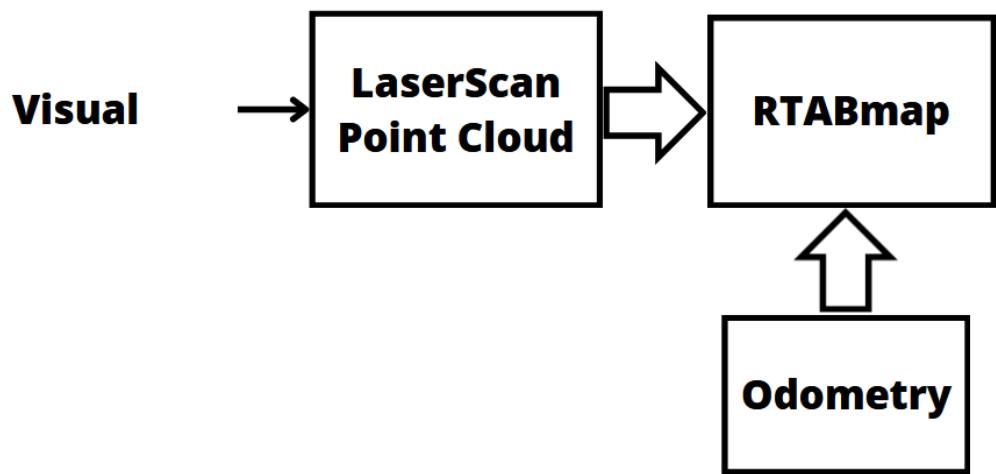
Gambar 11 Diagram Blok Sistem

Secara umum pada diagram blok sistem menerima masukan berupa  
persepsi robot yang dibangun berdasarkan data inputan dari berbagai  
sensor pendukung, data visual yang ditangkap robot kemudian di  
proses untuk menghasilkan data peta dan estimasi posisi serta orien  
tasi keberadaan robot dalam sistem SLAM



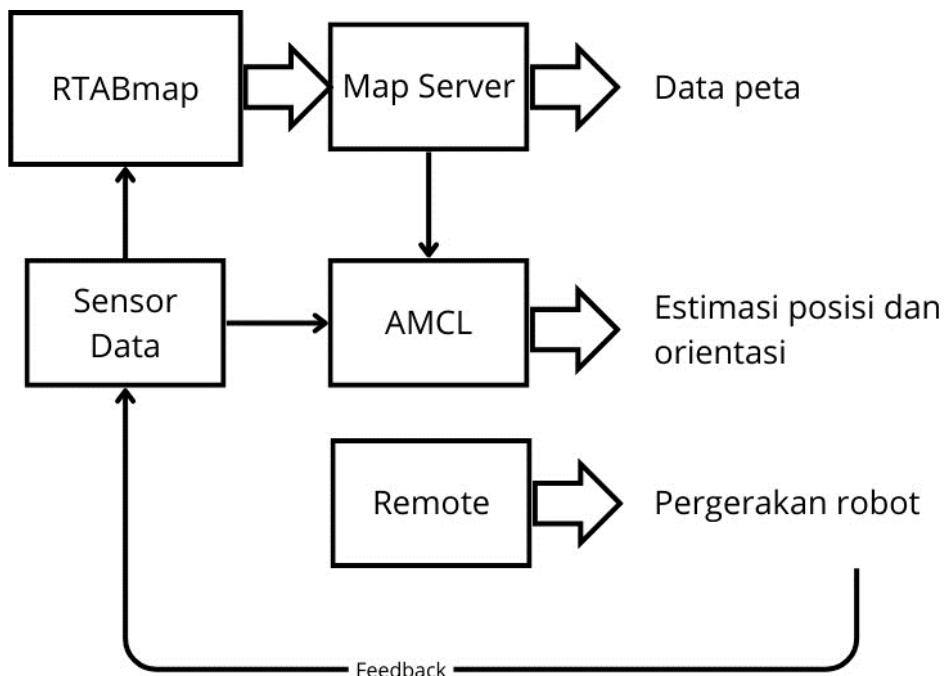
Gambar 12 Diagram Masukan

Pada blok masukan robot menerima inputan dari berbagai sensor. pada low level robot menerima masukan berupa encoder dan IMU yang dibaca oleh Arduino kemudian dikirimkan ke high level. Kemudain pada high level akan menggabungkan data sensor dari Arduino dan informasi visual dari Kinect untuk menghasilkan data odometry gabungan menggunakan modul EKF robot pose yang berupa estimasi posisi dan orientasi robot. Robot juga akan menerima data visual dari Kinect berupa persepsi bentuk lingkungan yang akan diolah bersamaan dengan data odometry untuk menjadi inputan bagi modul RTABmap guna menghasilkan peta



Gambar 13 Diagram proses

Pada tahap pemrosesan modul RTABmap pada robot akan mengolah data yang diterima menjadi peta representasi bentuk lingkungan disekitar robot dalam bentuk occupancy grid map. Peta ini kemudian akan diterima oleh modul map server sehingga dapat disimpan menjadi peta yang siap digunakan



Gambar 14 Digaram keluaran

Data pada map server akan divisualisasikan pada rviz dan menjadi inputan oleh modul AMCL untuk bersamaan dengan data sensor untuk menghasilkan partikel probabilitik keberadaan robot pada peta yang telah dibangun oleh RTABmap.

### 3.3 Rancangan Perangkat Keras

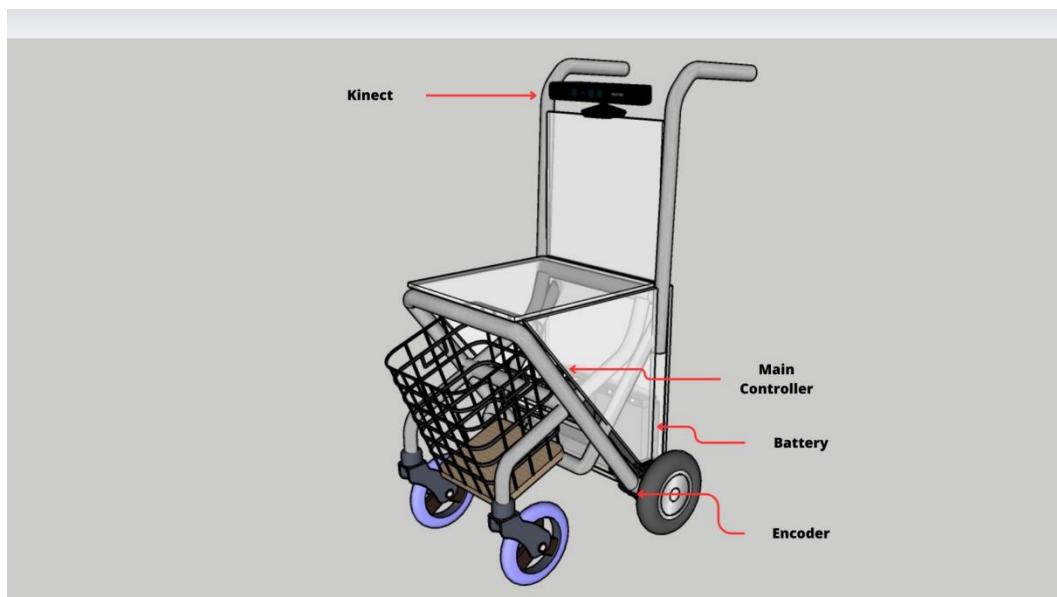
Dalam perancangan perangkat keras, terbagi atas beberapa tahapan perancangan sebagai berikut.

#### 3.3.1 Mekanisasi Robot

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

##### 3.3.1.1 Desain Robot

Dalam mekanisasi robot dimulai dengan merancangan chasis atau badan robot agar komponen sensor dan daya dapat diletakan bagian *control board*. Chasis yang digunakan pada penelitian ini merupakan modifikasi dari penelitian Muhammad Takbir (2019) yang di modifikasi dengan menempatkan sensor, kontroler, aktuator serta komponen pendukung lainnya untuk menunjang fungsi robot dalam melakukan pemetaan lingkungan yang dapat kita lihat pada Gambar 4.



Gambar 15 Gambar Desain Robot

Keterangan informasi terkait komponen yang digunakan dalam mekanisasi mobile robot yang digunakan sebagai berikut.

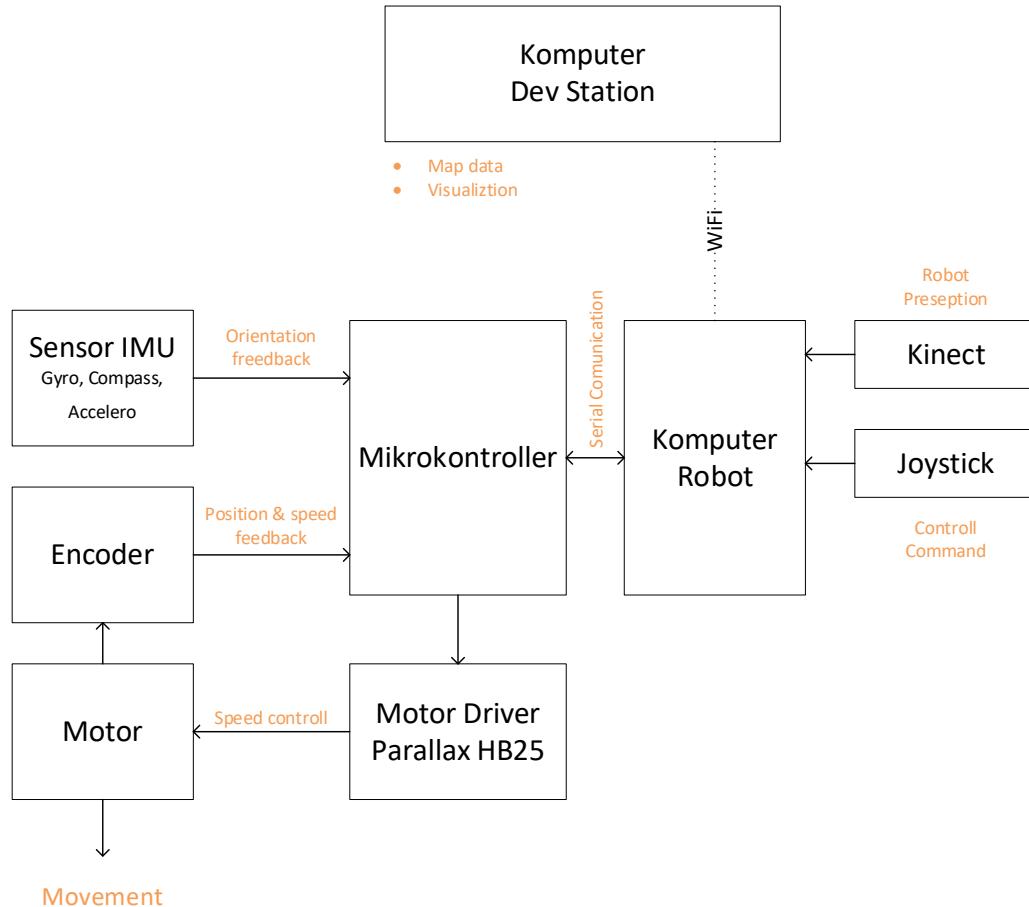
Tabel 2 Komponen utama rancang bangun robot otomotif Waiter-Bot

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

Jenis	Komponen	Kegunaan
	PC	Melakukan komputasi ROS dan mengontrol sistem bagian <i>high level</i>

### 3.3.2 Desain Elektronika

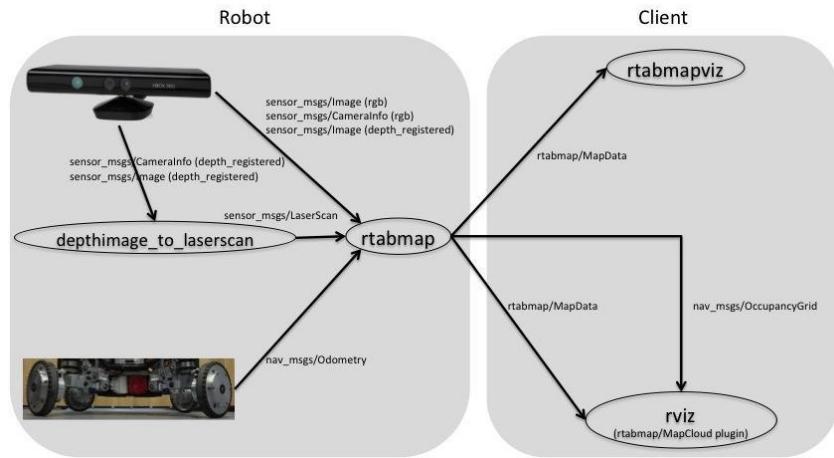
Pada perancangan kelistrikan mobile robot dilakukan dengan mempertimbangkan penambahan sensor dan modul kontroler sesuai dengan kebutuhan robot dalam pemenuhan misi, yaitu untuk melakukan pemetaan. Selain itu, bagian daya juga harus dirancang untuk memenuhi kebutuhan seluruh komponen elektronik agar dapat beroperasi. Pada robot, kebanyakan komponen membutuhkan sumber catu daya 5v termasuk kontroler dan sensor, sedangkan motor membutuhkan supply 12v sehingga robot ini menggunakan dua buah aki 12v yang dihubungkan dengan buck converter 5v. hubungan antara komponen dapat dilihat pada blok diagram dibawah.



Gambar 16 Blok Diagram Umum hubungan antar komponen

### 3.3.2.1 Perancangan Susbsistem Sensor

Sistem sensor pada robot ini terbagi menjadi 2 bagian besar yaitu odometry dan presepsi lingkungan yang digunakan dalam tahap pemetaan dan lokalisasi. Dapat dilihat blok diagram subsistem sensor rancang bangun robot pada Gambar 5.



Gambar 17 Subsistem Sensor Robot

Komponen pada subsistem diatas berupa sensor antara lain:

### 3.3.2.2 Sensor Kinect Xbox 360

Kinect adalah "controller-free gaming" oleh Microsoft dan Xbox 360 video game platform. Sensor Kinect adalah batang horizontal yang terhubung dengan alas kecil yang memiliki poros yang dapat berputar. Perangkat ini memiliki kamera RGB, sensor kedalaman, dan mikrofon, yang menyediakan kemampuan untuk menangkap gerak secara 3D, mengenali wajah, dan mengenali suara

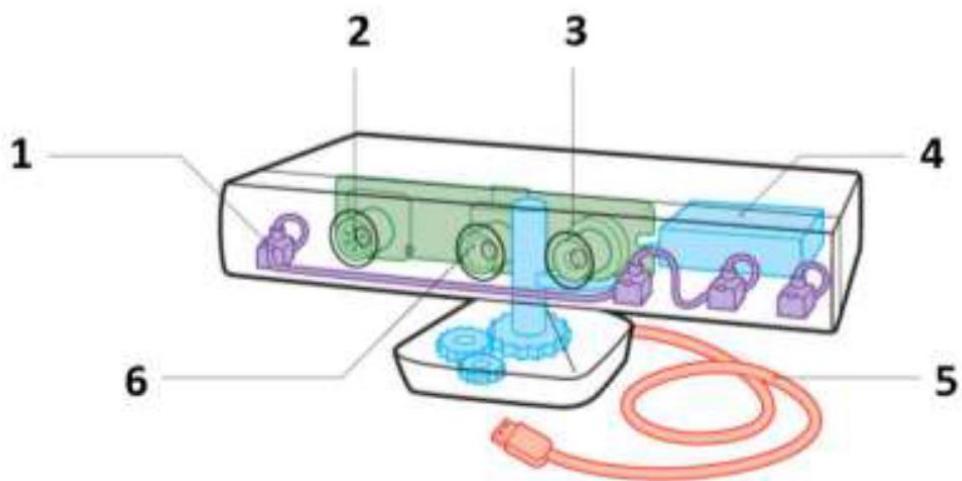
Sensor kedalaman terdiri dari proyektor laser infrared dikombinasikan dengan sensor CMOS monokromatik, yang merekam data video 3D dalam kondisi pencahayaan apa pun. Pengiriman video RGB menggunakan resolusi VGA (640 x 480 piksel) dengan kedalaman warna 8-bit dengan filter warna Bayer, sedangkan pengiriman video monokrom untuk deteksi kedalaman menggunakan resolusi VGA dengan kedalaman warna 11-bit. Sensor Kinect memiliki area penggunaan 1.2–35m (3.9–11 kaki), memiliki lebar pandangan angular 57 derajat horizontal dan 43 derajat vertikal, dan poros yang digerakkan oleh motor sampai dengan 27 derajat atas dan bawah. Bidang horizontal dari sensor Kinect pada jarak minimum ~0.8m (2.6 kaki) adalah sekitar ~87 cm (34 inch), dan bidang vertikal adalah ~63 cm (25 inch), menghasilkan resolusi sekitar 1.3 mm (0.051 inch) setiap pikselnya. Fitur mikrofon pada Kinect m

emiliki empat mikrofon kapsul dan setiap jalur dioperasikan dengan kedalam suara 16-bit pada kecepatan cuplik 16 kHz



Gambar 18 Gambar Data Kedalaman Sensor Kinect

(Sumber : Zaenuddin dkk, 2014)



Gambar 19 Komponen-Komponen Sensor Kinect

(Sumber : [https://www.wired.com/2011/06/mf\\_kinect/](https://www.wired.com/2011/06/mf_kinect/))

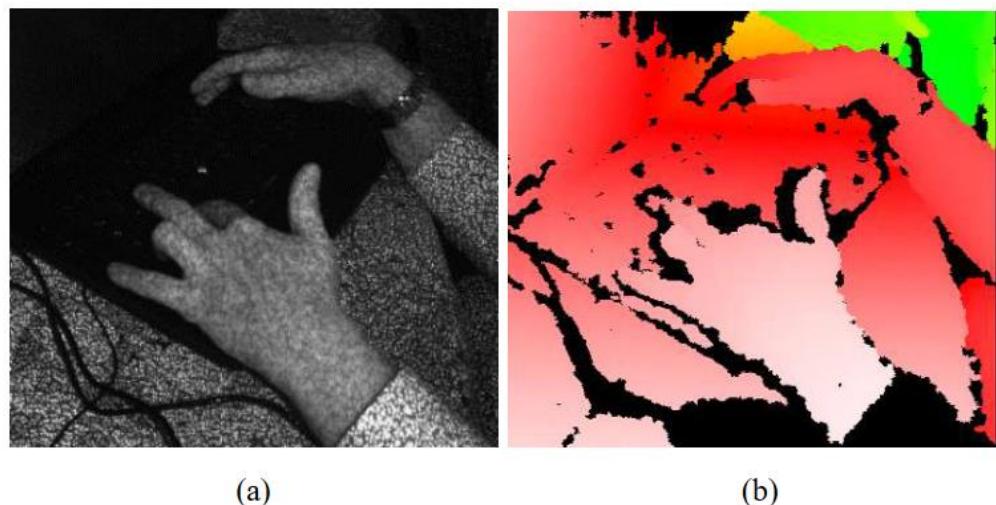
Data kedalaman Kinect dengan warna cerah menunjukkan bahwa objek dekat ke kamera. Namun, karena terlalu dekat akan menyebabkan objek tidak terdeteksi dan ditandai sebagai warna hitam dan objek terjauh akan berwarna

arna gelap, hal ini ditunjukkan pada Gambar 20. Output berupa histogram dari akumulatif frekuensi kemunculan setiap nilai. Histogram dibangun berdasarkan nilai-nilai kedalaman setiap pixel (Zaenuddin dkk, 2014)

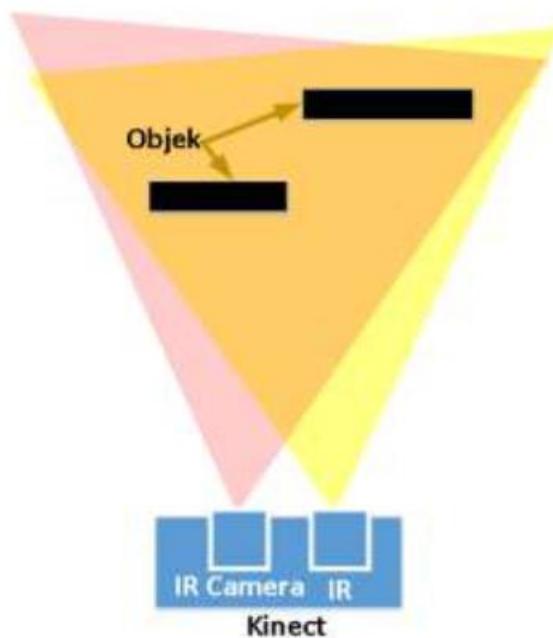
Gambar 19 menunjukkan isi dari komponen yang ada pada sensor kinect , komponen tersebut adalah

- |                             |                 |
|-----------------------------|-----------------|
| 1. Microphone Array         | 4. Tilt motor   |
| 2. IR emitter               | 5. USB cable    |
| 3. Depth camera / IR Camera | 6. Color camera |

Data depth kinect di dapatkan dari paduan IR Emitter dan IR Camera, prinsip kerjadari kedua perangkat tersebut di ilustrasikan pada Gambar 2 1, IR Emitter dengan IR Camera memiliki jarak yang konstan. Perbedaan jarak pada kedua perangkat mengakibatkan perbedaan penangkapan objek, sehingga terdapat beberapa titik yang terkena IR Emitter tetapi tidak terbaca oleh kamera, begitu juga sebaliknya. Cahaya yang di pancarkan IR Emitter memiliki pola-pola titik. Ketika jarak bendase makin jauh maka pola-pola titik akan semakin berjauhan dan semakin kecil. Ketika jarak benda semakin dekat maka pola-pola titik akan semakin berdekatan dan semakin besar. Berdasarkan pola-pola tersebut IR Camera akan membaca intensitas dari cahaya IR yang di pancarkan. Pancaran sinar IR dan data Depth di tunjukkan pada Gambar 20



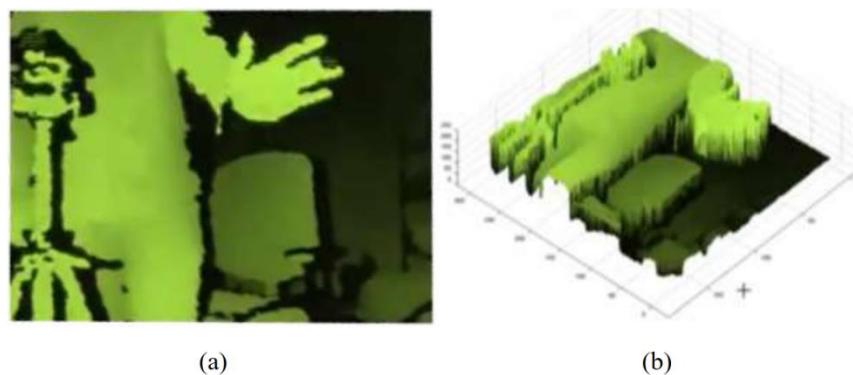
Gambar 20 Data Depth Kinect, (a) IR Kinect, (b) Depth Map Kinect  
(Sumber : <https://en.wikipedia.org/wiki/Kinect>)



Gambar 21 Prinsip Kerja Kinect

Berdasarkan prinsip di atas data depth kinect mempresentasikan jarak yang di dapat dari intensitas cahaya yang di tangkap oleh IR Camera. Semakin jauh suatu benda intensitas cahaya akan semakin kecil, kecilnya intensitas juga di pengaruhidari jauhnya pola-pola titik IR pada benda tersebut . Hasil jarak di presentasikan dalam 2D array, nilai dari 2d array dapat

mewakili semua jarak, hasil 2d arraybeserta plot diagram 3d di tunjukkan pada Gambar 22. berdasarkan Gambar 22, dapat terlihat perbedaan jarak antara orang dengan kursi, dengan dinding, dan dengan tripod kamera malalui plot 3d

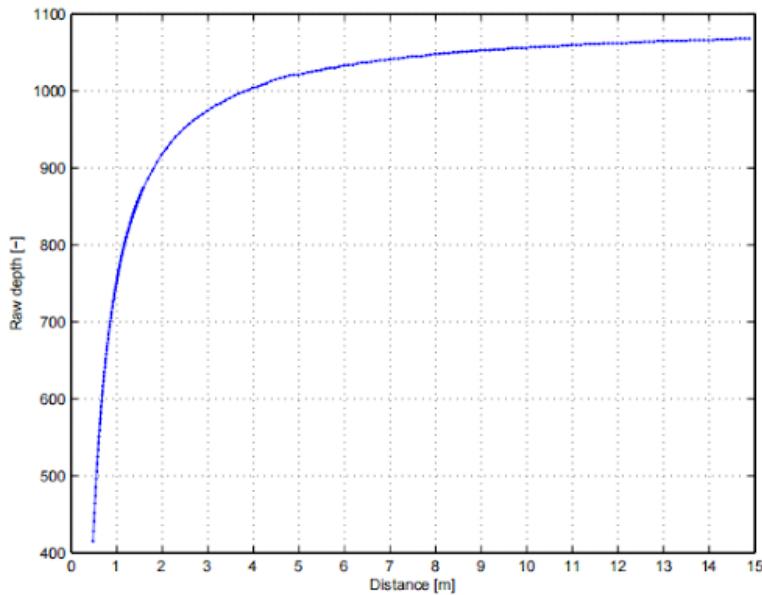


Gambar 22 Data Depth Kinect (a) Bentuk 2D Array, (b) Dalam Plot 3D

(Sumber : <https://www.youtube.com/watch?v=uq9SEJxZiUg>)



Gambar 23 Kinect memberikan tiga output: Gambar IR, gambar RGB Kamera IR ( $1280 \times 1024$  piksel untuk FOV  $57 \times 45$  derajat, panjang fokus 6,1 mm, ukuran piksel 5,2  $\mu\text{m}$ ) digunakan untuk mengamati dan menerjemahkan pola proyeksi IR untuk melakukan triangulasi pemandangan 3D.

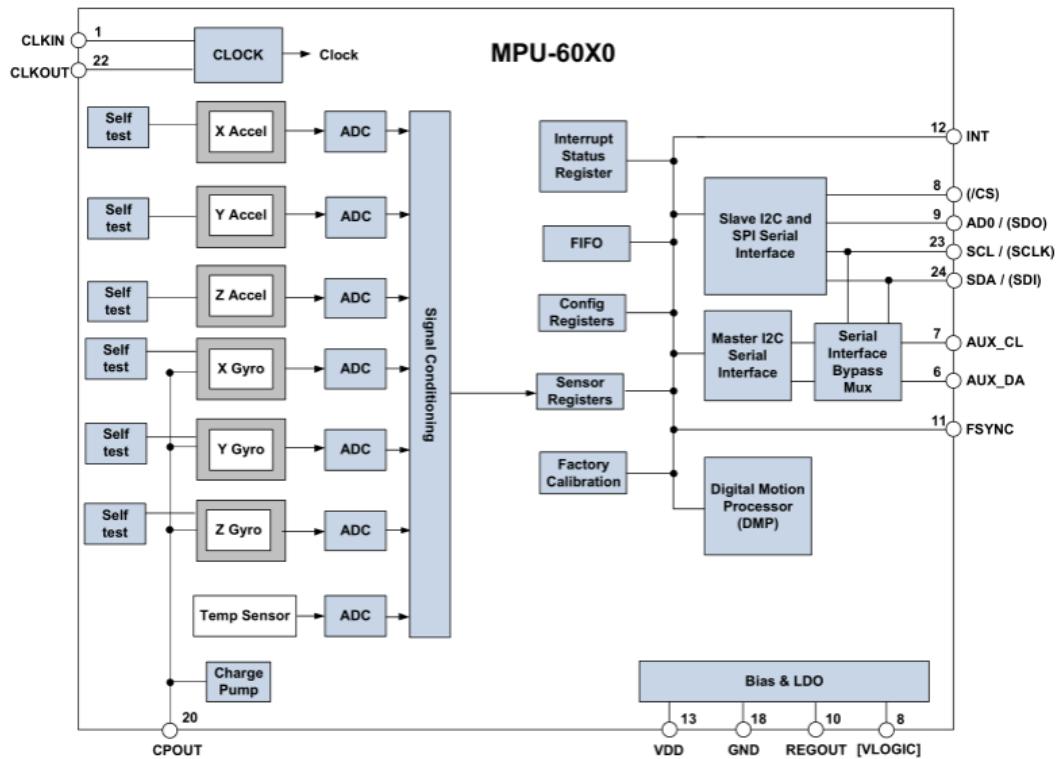


Gambar 24 Invers Kinect pada fungsi kedalaman asli

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

### 3.3.2.3 Sensor IMU

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



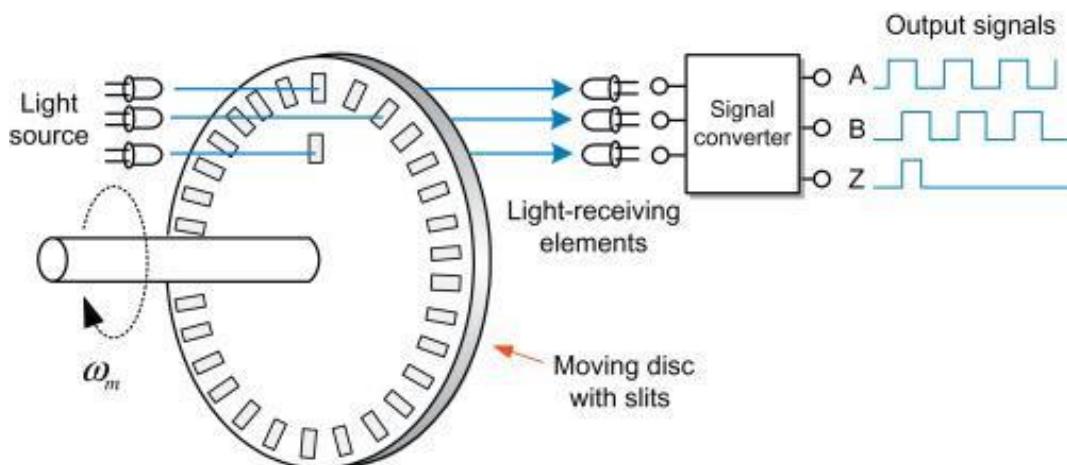
Gambar 25 Blok diagram modul sensor IMU MPU-60X0

Sensor IMU giroskop pada modul MPU-6050 ini terdiri dari tiga giroskop MEMS vibrasi, yang mendeteksi rotasi pada sumbu x, y, dan z. Ketika gyroskop diputar pada salah satu sumbu, Efek Coriolis menyebabkan getaran yang dideteksi oleh pickoff kapasitif yang menghasilkan sinyal tegangan. Sinyal yang ini dihasilkan diperkuat, dimeodulasi, dan difilter untuk menghasilkan tegangan yang sebanding dengan tingkat sudut. Tegangan ini didigitalkan menggunakan *on-chip* 16-bit *Analog-to-Digital Converters* (ADC) untuk sampel setiap sumbu. Rentang skala penuh sensor gyro dapat diprogram secara digital hingga  $\pm 250$ ,  $\pm 500$ ,  $\pm 1000$ , atau  $\pm 2000$  derajat per detik (dps) (Bagoes Prawira, 2018).

### 3.3.2.4 Sensor Encoder

Rotary Encoder merupakan suatu komponen elektro mekanis yang memiliki fungsi untuk memonitoring posisi anguler pada suatu poros yang berputar. Dari perputaran benda tersebut data yang termonitor

ing diubah oleh y ke dalam bentuk data digital berupa lebar pulsa kemudian dikirim ke kontroler (Mikrokontroler). Berdasarkan data yang di dapat berupa posisi anguler (sudut) kemudian dapat diolah oleh kontroler sehingga mendapatkan data berupa kecepatan, arah, dan posisi dari perputaran porosnya[8].



Gambar 26 Sistem kerja encoder

(Sumber : Kevin M. Lynch, 2016)

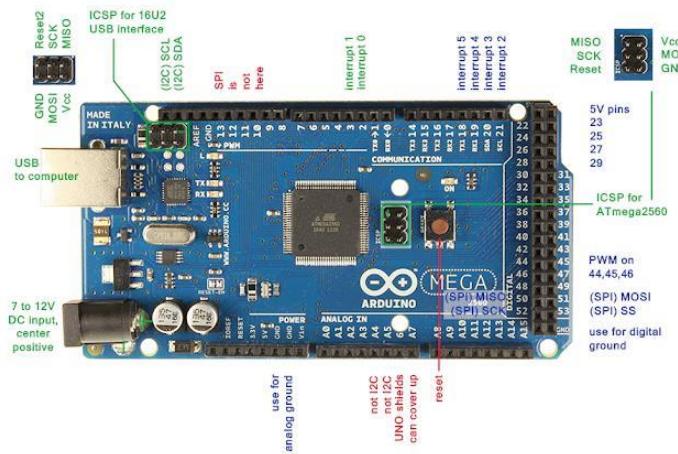
### 3.3.2.5 Perancangan Subsistem Proses

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

### 3.3.2.6 Arduino Mega 2560

Arduino Mega 2560 merupakan papan mikrokontroller yang berbasis ATmega 2560 dimana memiliki 54 pin digital input / output (15 pin diantaranya adalah PWM), 16 pin analog input, 4 pin UART (serial port hardware). Arduino Mega 2560 juga dilengkapi oscillator 16

Mhz, sebuah port USB, power jack DC, ICSP header, dan tombol reset . Itu semua dibutuhkan untuk mendukung mikrikontroler, untuk mulai mengktifkan cukup dengan menghubungkan power dari USB ke computer atau dengan adaptor AC - DC ke jack DC. Arduino Mega 2560 juga kom patibel dengan sebagian besar shield yang di rancang untuk Arduino Deumilanove atau Diecimila (Utami, 2015). Arduino Mega 2560 ditunj ukkan pada gambar 27 dibawah ini.



Gambar 27 Board Arduino Mega 2560

(Sumber : Arduino, 2018)

Spesifikasi dari Arduino Mega 2560 dapat di lihat di tabel 3 dibaw ah ini.

Tabel 3 Spesifikasi Arduino Mega 2560

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

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

Sumber : (Arduino, 2018)

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

### 3.3.2.7 Laptop

Peracangan sistem pemetaan pada robot mobile ini menggunakan 1 laptop sebagai komputer utama untuk mengolah semua data yang diterima dari berbagai sensor maupun Arduino. Laptop ini digunakan untuk menjalankan framework ros yang berjalan pada sistem operasi berbasis linux yaitu ubuntu 20.04. Untuk menjalankan proses pemetaan dan lokalisasi yang menitikberatkan pada pengolahan citra, laptop ini ditenagai dengan prosesor intel core i7 generasi ke 12 dan dikombinasikan dengan kartu grafis nvidia rtx 3050 menjadikan proses pemetaan dan lokalisasi dapat berjalan dengan lancar.



Gambar 28 Laptop HP Victus 15



Gambar 29 Prosesor Intel core i7



Gambar 30 GPU Nvidia RTX3050

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

Tabel 4 Spesifikasi Komputer yang Digunakan

Processor	Up to an Intel® Core™ i7-12700H processor (up to 4.7 GHz with Intel® Turbo Boost Technology(2g), 24 MB L3 cache, 14 cores, 20 threads)
Graphics	Up to 75W TGP WITH NVIDIA® GeForce RTX™ 3050 Ti Laptop graphics (4 GB GDDR6 dedicated) with DLSS, Ray Tracing, and Max-Q technologies.

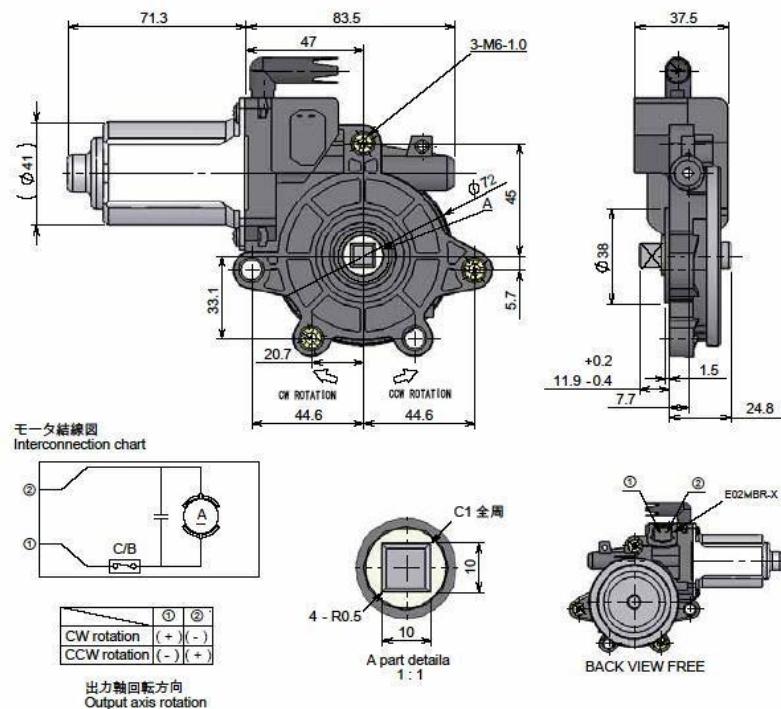
	<p>Includes the following NVIDIA Max-Q technology:</p> <ul style="list-style-type: none"> <li>Dynamic Boost</li> <li>Battery Boost</li> <li>Rapid Core Scaling</li> <li>CPU Optimizer</li> <li>Resizable BAR</li> <li>Optimal Playable Settings</li> </ul>
Memory	Up to 16 GB DDR4-3200 MHz RAM (2 x 8 GB)
Memory Slots	2
Storage	Up to a 1 TB PCIe® Gen4 NVMe™ TLC M.2 SSD
Wireless Connectivity	Up to Intel® Wi-Fi 6E3 AX211 (2x2) and Bluetooth® 5.2 combo (Supporting Gigabit data rate) with MU-MIMO supported
External I/O Ports	<ul style="list-style-type: none"> <li>SuperSpeed USB Type-C® 5Gbps signaling rate (DisplayPort™ 1.4, HP Sleep and Charge)</li> <li>SuperSpeed USB Type-A 5Gbps signaling rate (HP Sleep and Charge)</li> <li>SuperSpeed USB Type-A 5Gbps signaling rate</li> <li>Combo Audio Jack</li> <li>Ethernet Port</li> <li>HDMI 2.1</li> </ul>

Sumber: <https://www.omen.com/us/en/laptops/2022-victus-15-intel>

### 3.3.2.8 Perancangan Subsistem Keluaran

Keluaran dari sistem ini berupa data map hasil pemetaan dan juga pergerakan roda robot dalam melakukan pemetaan ruangan. Komponen yang bertanggung jawab dalam menghasilkan data peta adalah komputer sedangkan untuk melakukan pergerakan adalah motor DC power window.

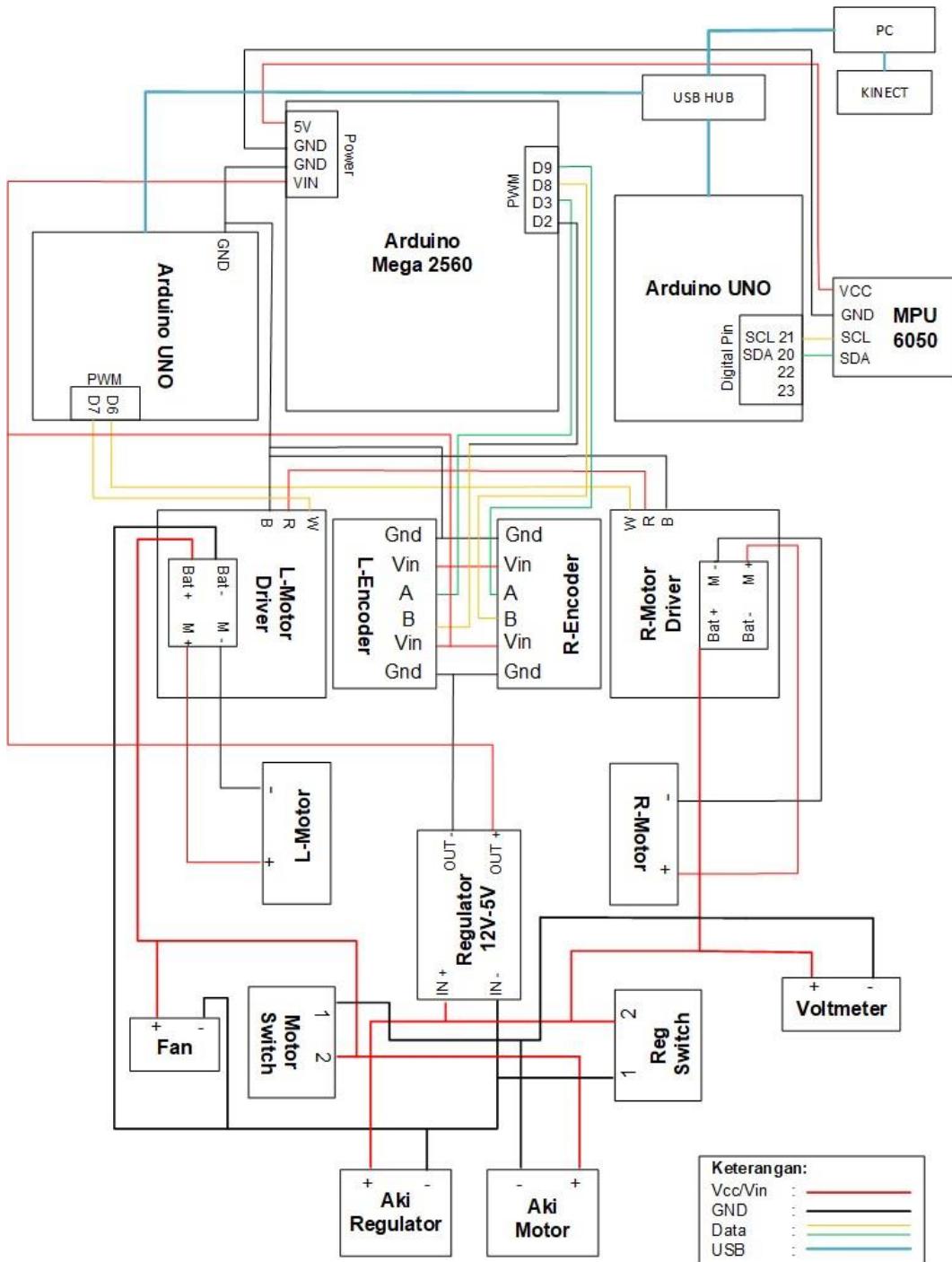
*Motor DC Power Window* adalah suatu motor yang mengubah energi listrik searah menjadi mekanis yang berupa tenaga penggerak torsi. (Christanto dkk., 2014).



Gambar 31 Motor DC Power Window 12 V

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

Secara lengkap hubungan antar komponen elektronika robot dapat dilihat pada skematik robot gambar 32



Gambar 32 skematik witres-bot

### 3.4 Rancangan Perangkat Lunak

Pada penelitian ini rancangan perangkat lunak merupakan bagian utama dari sistem pemetaan dan lokalisasi yang di rancang. Bagian

n ini dibagi menjadi beberapa tahapan proses mulai dari akusisi data sensor oleh mikrokontroler low level, proses komunikasi dengan komputer utama, integrasi dengan ros, filterisasi data sensor, pemodelan odometry serta hubungan antar link dan join, hingga pemrosesan peta dan lokalisasi

### 3.4.1 Instalasi ROS

ROS Noetic adalah salah satu versi dari Robot Operating System (ROS) yang dirilis oleh Open Robotics. ROS Noetic merupakan versi ke-12 dari ROS dan secara khusus dirancang untuk kompatibilitas dengan OS Ubuntu 20.04. ROS adalah sebuah framework perangkat lunak yang digunakan dalam pengembangan robot dan sistem kendali robotik.

proses instalasi ROS Noetic pada OS Ubuntu 20.04 dapat dilakukan dengan beberapa langkah berikut:

1. Tahap Pertama: Menyiapkan Repository ROS

- Buka terminal pada OS Ubuntu 20.04.
- Tambahkan repository ROS ke daftar sumber perangkat lunak dengan menjalankan perintah berikut:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu focal main" > /etc/apt/sources.list.d/ros-latest.list'
```

2. Tahap Kedua: Menyiapkan Kunci Verifikasi Untuk memastikan keamanan dan integritas paket yang diunduh, unduh dan import kunci verifikasi dengan menjalankan perintah berikut:

```
curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

3. Tahap Ketiga: Melakukan Instalasi ROS Noetic Update daftar paket pada sistem dengan menjalankan perintah berikut:

```
sudo apt update
```

Selanjutnya, instal ROS Noetic beserta paket-paket yang diperlukan dengan menjalankan perintah berikut:

```
sudo apt install ros-noetic-desktop
```

4. Tahap Keempat: Konfigurasi Lingkungan ROS

Setelah proses instalasi selesai, konfigurasikan lingkungan ROS dengan menjalankan perintah berikut:

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

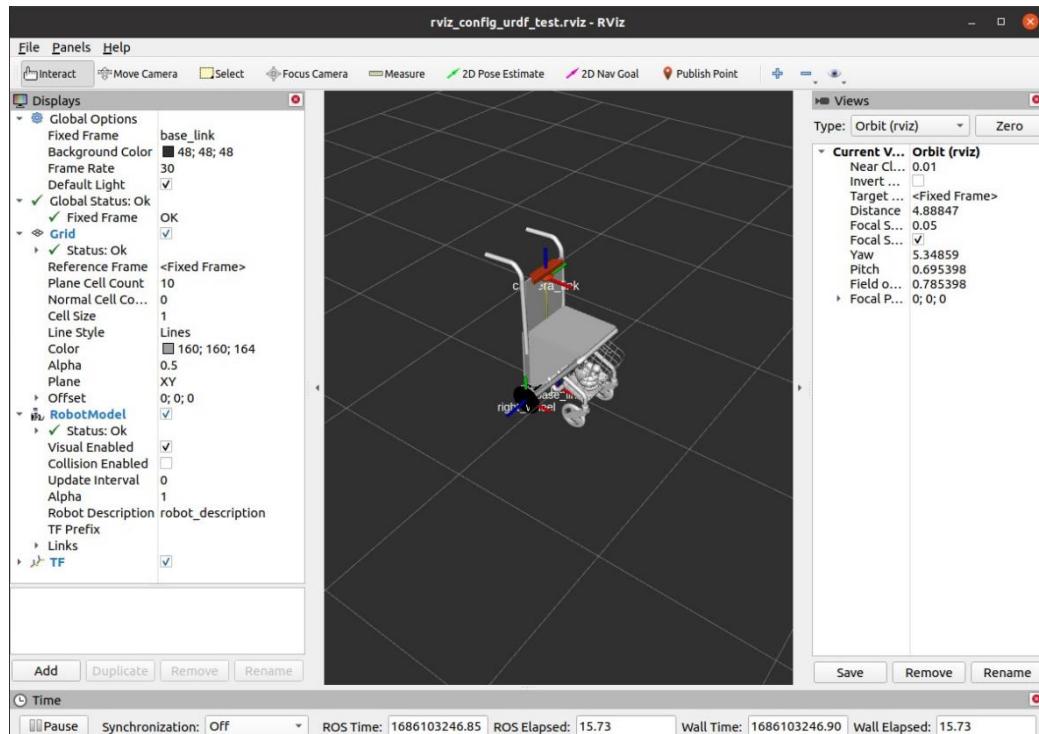
5. Tahap Kelima: Verifikasi Instalasi ROS

Untuk memastikan instalasi berhasil, jalankan perintah berikut untuk melihat daftar perintah ROS yang tersedia:

```
Roscore
```

Jika tidak ada kesalahan, maka ROS sudah terinstal dan berjalan dengan baik pada OS Ubuntu 20.04.

### 3.4.2 Pemodelan Robot dengan URDF



Gambar 33 Visualisasi Model Robot dengan Rviz

Pemodelan robot dalam format URDF (Unified Robot Description Format) pada ROS (Robot Operating System) melibatkan beberapa tahapan penting. Tahapan-tahapan ini mencakup penulisan deskripsi robot, penambahan file mesh, dan integrasi dengan sistem kontrol robot.

Tahap pertama adalah menulis deskripsi robot menggunakan URDF. Deskripsi ini mencakup informasi mengenai link (komponen) dan joint (hubungan antara komponen) dalam robot. Deskripsi yang akurat dan rinci membantu dalam merepresentasikan struktur dan geometri robot dengan tepat.

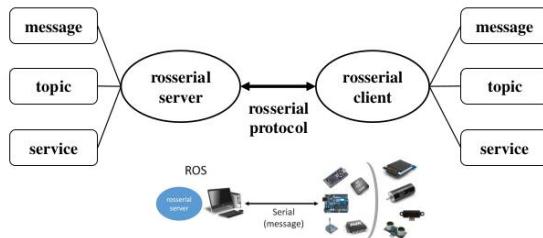
Selanjutnya, file mesh, yang mewakili visualisasi dari setiap link robot, dimasukkan ke dalam URDF. File mesh ini berisi informasi tentang bentuk dan tampilan visual dari masing-masing komponen robot. Memasukkan file mesh yang sesuai memungkinkan simulasi dan visualisasi yang lebih realistik dari robot.

Proses ini penting karena URDF memungkinkan simulasi yang akurat dan visualisasi yang lebih baik dari robot. Dengan menggunakan URDF, sistem kontrol robot otonom dapat mengakses informasi mengenai geometri robot, batas pergerakan sendi, dan hubungan antar-komponen. Hal ini memungkinkan sistem kontrol untuk merencanakan dan melaksanakan gerakan robot dengan presisi yang tinggi.

Selain itu, penggunaan URDF dalam sistem kendali robot otonom juga penting dalam pembuatan peta dan lokalisasi. URDF menyediakan informasi geometri yang diperlukan untuk pembuatan peta, seperti pengukuran jarak dan visualisasi persepsi sensor. Dengan menggunakan URDF, sistem kendali dapat mengintegrasikan data sensor dari robot dengan tepat untuk membangun peta lingkungan yang akurat. Selanjutnya, URDF juga membantu dalam proses lokalisasi, dengan memungkinkan sistem kendali untuk menentukan posisi dan orientasi robot dalam ruang dengan menggunakan informasi sensor yang terkait.

### 3.4.3 Pembuatan node program pada ROS

#### 3.4.3.1 Node Ros Serial Arduino



Gambar 34 Komunikasi Komputer-Arduino Ros Serial

Penggunaan ROS Serial untuk Komunikasi antara Arduino dan Komputer ROS Serial digunakan sebagai protokol komunikasi antara Arduino dan Komputer dalam sistem ini. ROS Serial menyediakan cara yang efektif untuk mentransfer data antar platform, memungkinkan pengiriman dan penerimaan pesan yang diimplementasikan dalam lingkungan ROS. Dengan menggunakan ROS Serial, Arduino dan Komputer dapat saling berkomunikasi dan berbagi data dengan mudah, memungkinkan koordinasi dan pengendalian yang terintegrasi.

Implementasi Node untuk Membaca dan Mengontrol Perangkat Keras pada Arduino Untuk mengontrol motor driver, Arduino menggunakan sinyal keluaran untuk mengendalikan kecepatan dan arah pergerakan motor. Selain itu, Arduino juga membaca input dari encoder untuk mendapatkan informasi posisi dan putaran motor. Data dari MPU9265 dan HCSR juga diambil oleh Arduino melalui koneksi yang sesuai.

Arduino mengirimkan data ini melalui komunikasi serial ke Komputer menggunakan ROS Serial. Komputer menerima data ini dan menggunakan informasi tersebut untuk melakukan pengolahan lebih lanjut, seperti pengendalian motor berdasarkan instruksi pengguna atau tugas-tugas yang diberikan oleh sistem kendali robot.

Sebaliknya, Komputer juga mengirimkan instruksi dan perintah ke Arduino melalui ROS Serial. Instruksi ini berfungsi untuk mengontrol motor sesuai dengan kebutuhan sistem, seperti kecepatan, arah putaran, atau gerakan yang diinginkan. Arduino menerima instruksi ini dan melakukan pengendalian perangkat keras motor sesuai dengan perintah yang diterima.

Dengan demikian, integrasi Arduino dengan SBC Komputer menggunakan ROS Serial memungkinkan komunikasi dua arah yang efisien antara kedua platform tersebut. Arduino bertanggung jawab dalam mengontrol perangkat keras, seperti motor driver, encoder, MPU9265, dan HCSR, sementara Komputer berperan dalam pengolahan data, pengiriman instruksi, dan koordinasi sistem secara keseluruhan.

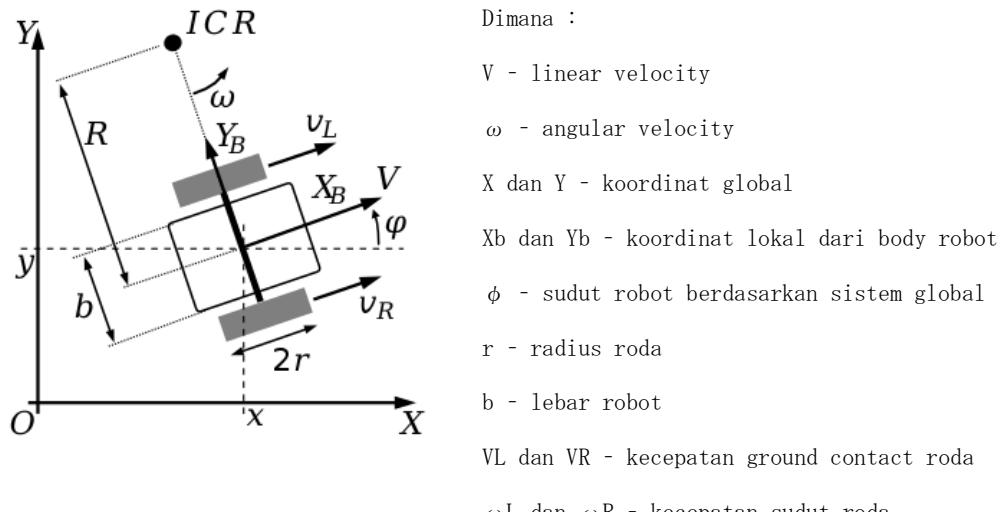
### 3.4.3.2 Ros Controller package – Differential Drive

Dalam perancangan ini, robot menggunakan kendali diferensial (diff drive) untuk mengatur gerakan dan posisi roda. Dengan menggunakan informasi mengenai jarak antar roda dan diameter roda, kendali diferensial memungkinkan pengendalian yang akurat terhadap kecepatan dan arah putaran robot.

Integrasi Kendali dengan Node ROS Untuk mengimplementasikan kendali diferensial menggunakan ROS Control, sebuah node ROS khusus dibangun. Node ini bertanggung jawab untuk menerima perintah dari pengguna atau sistem kendali robot, menghitung gerakan yang diperlukan berdasarkan perintah tersebut, dan mengirimkan instruksi ke robot.

Dalam node tersebut, perhitungan gerakan berdasarkan perintah menggunakan parameter seperti jarak antar roda dan diameter roda. I

Informasi ini digunakan untuk menghasilkan kecepatan dan arah putaran yang sesuai untuk setiap roda, sehingga robot dapat bergerak sejauh dengan instruksi yang diberikan.



Gambar 35 Kinematics of Differential Drive Robots

yang dapat dirumuskan dengan persamaan matematis berikut :

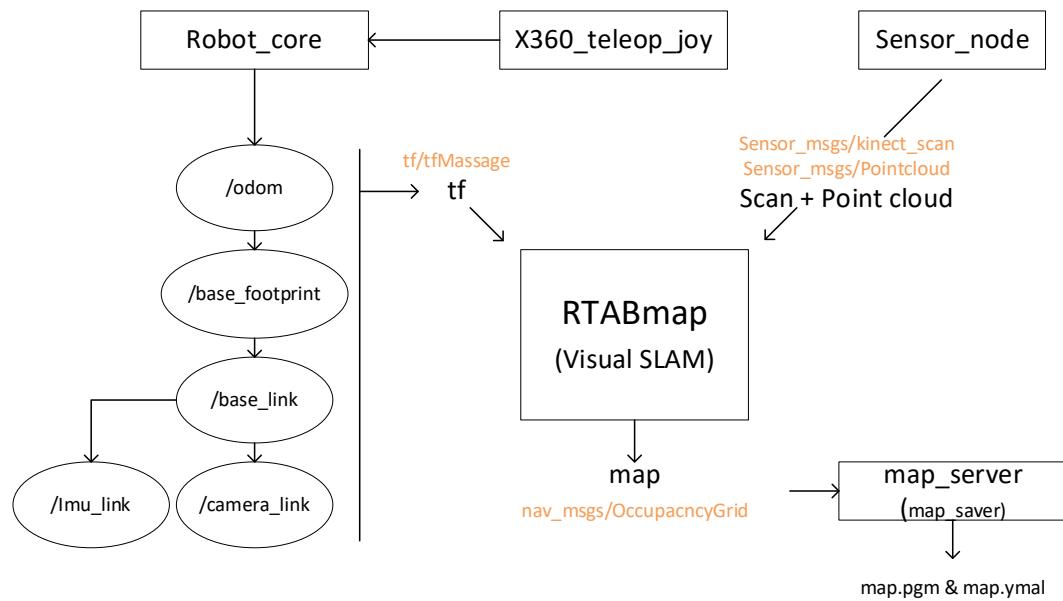
$$\omega_L = \frac{V - \omega \cdot b/2}{r} \quad \text{dan} \quad \omega_R = \frac{V + \omega \cdot b/2}{r}$$

Integrasi kendali dengan node ROS memungkinkan komunikasi yang mulus antara sistem kendali robot dan komponen lainnya, seperti sensor atau sistem navigasi. Ini memungkinkan penggunaan informasi dari sensor, seperti odometri dengan visual IMU dan encoder, untuk memberikan umpan balik yang akurat dalam pengendalian robot.

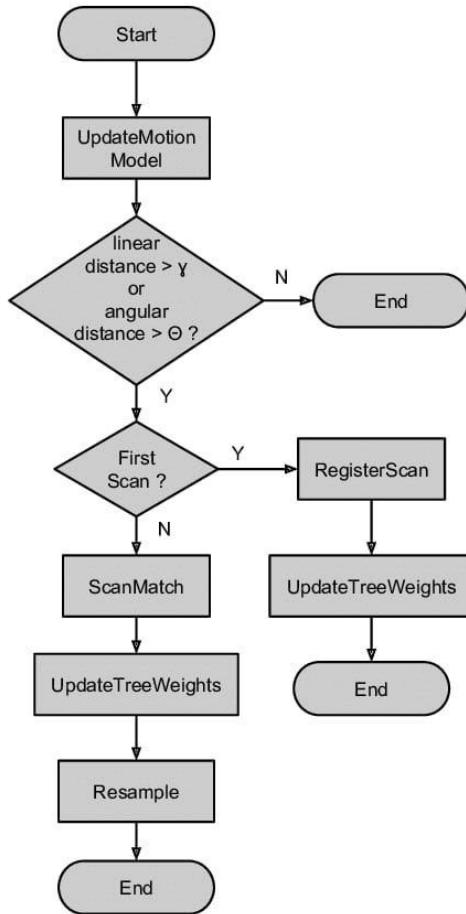
Dengan mengintegrasikan kendali diferensial menggunakan ROS Control dan memanfaatkan node ROS, robot dapat dikendalikan dengan presisi dan fleksibilitas yang tinggi. Hal ini memungkinkan robot untuk bergerak secara akurat sesuai dengan perintah pengguna atau s

istem kendali, memungkinkan navigasi yang efisien dan responsif dalam sistem kendali robot otonom, pembuatan peta menggunakan RTAB-M ap SLAM, dan lokalisasi dengan AMCL.

### 3. 4. 3. 3 Real-Time Appearance-Based Mapping



Gambar 36 Blok diagram pemetaan dan lokalisasi dengan RTABMAP



Gambar 37 Flowchart sistem pemetaan

RTAB-Map merupakan algoritme pemetaan berbasis visual yang menggabungkan pemetaan simultan (SLAM) dan pemetaan berbasis penampilan. Algoritme ini menggunakan sensor visual, seperti kamera Kinect Xbox 360, untuk mengumpulkan data citra dan informasi kedalaman dari lingkungan sekitar robot. Dengan memanfaatkan informasi visual ini, RTAB-Map dapat membangun peta lingkungan secara real-time dan melakukan lokalisasi dalam lingkungan yang tidak diketahui.

Tahapan Pemetaan dengan RTAB-Map:

1. Persiapan Lingkungan: Tahap pertama adalah menyiapkan lingkungan dan menghubungkan sensor Kinect Xbox 360 dengan robot.

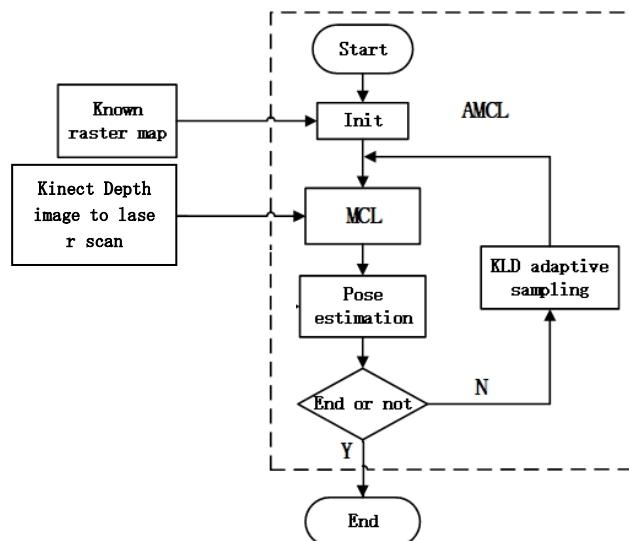
Sensor ini akan digunakan untuk mengambil data citra dan informasi kedalaman lingkungan.

2. Pengumpulan Data: Selanjutnya, robot bergerak di sekitar lingkungan untuk mengumpulkan data citra dan informasi kedalaman menggunakan Kinect Xbox 360. Data ini akan digunakan sebagai input untuk proses pemetaan.
3. Ekstraksi Fitur: Data citra dan kedalaman yang dikumpulkan dari sensor dianalisis untuk mengekstraksi fitur-fitur penting, seperti titik-titik kunci atau pola yang dapat digunakan untuk mengidentifikasi objek dan fitur lingkungan.
4. Pencocokan Fitur dan Pembangunan Peta: Pada tahap ini, RTAB-Map membandingkan fitur-fitur yang diekstraksi dengan fitur-fitur yang sudah ada dalam peta yang sedang dibangun. RTAB-Map menggunakan metode pencocokan fitur untuk menentukan letak relatif objek baru dalam peta dan membangun peta lingkungan secara bertahap.
5. Loop Closure: Ketika robot kembali ke area yang sudah dilalui sebelumnya, RTAB-Map dapat mendeteksi loop closure, yaitu mengenali bahwa posisi yang sama telah dilihat sebelumnya dalam perjalanan robot. Loop closure membantu meningkatkan akurasi dan konsistensi peta dengan memperbaiki estimasi posisi dan mengurangi kesalahan yang terjadi seiring perjalanan robot.
6. Optimisasi Peta: Setelah loop closure terdeteksi, peta lingkungan diperbarui dan dioptimasi untuk meningkatkan kualitasnya. Ini melibatkan penyesuaian posisi objek dalam peta untuk mengurangi kesalahan atau ketidaksesuaian yang mungkin terjadi.

7. Occupancy Grid Map: Salah satu bentuk peta yang dihasilkan oleh RTAB-Map adalah occupancy grid map. Occupancy grid map adalah representasi visual dari lingkungan yang dibagi menjadi sel-sel berukuran tetap. Setiap sel dapat berisi informasi tentang status okupansi, yaitu apakah sel tersebut terisi oleh objek atau tidak. Data dari Kinect Xbox 360 digunakan untuk mengisi grid map ini dengan informasi okupansi.

RTAB-Map dapat menghasilkan peta lingkungan yang semakin berkualitas seiring pergerakan robot dan pengumpulan data yang lebih banyak. Peta tersebut dapat digunakan untuk berbagai tujuan dalam sistem kendali robot otonom, termasuk navigasi, perencanaan jalur, dan penghindaran rintangan.

#### 3.4.3.4 Adaptive Montecarlo localization



Gambar 38 Flowchart lokalisasi dengan AMCL

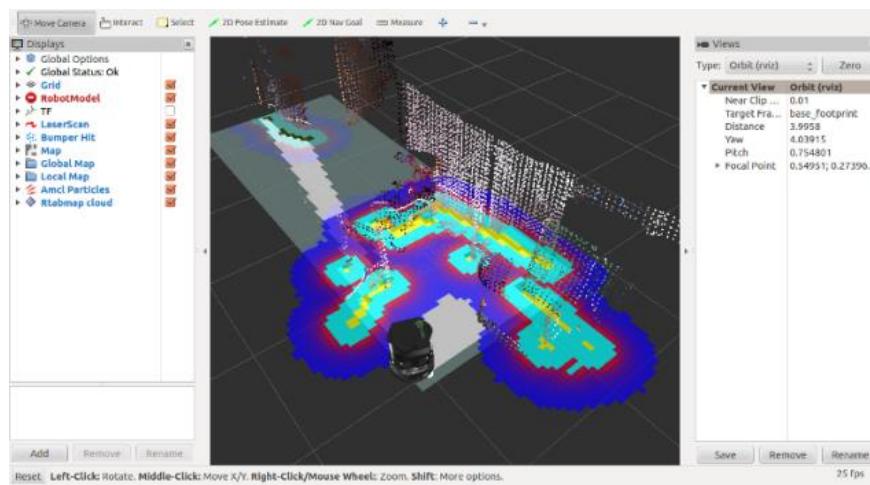
Proses kerja lokalisasi dengan AMCL (Adaptive Monte Carlo Localization) menggunakan grid map yang dihasilkan oleh RTAB-Map adalah sebagai berikut:

1. Grid Map sebagai Referensi: Grid map yang dihasilkan oleh RTAB –Map digunakan sebagai peta referensi untuk proses lokalisasi. Grid map ini berisi informasi okupansi di setiap sel, yang men cerminkan apakah sel tersebut ditempati oleh objek atau tidak.
2. Persiapan Awal: Sebelum memulai proses lokalisasi, AMCL memerl ukan informasi awal berupa estimasi posisi dan orientasi robot . Estimasi ini bisa berasal dari sensor odometri atau informasi lokalisasi awal yang diberikan.
3. Pengamatan Sensor: Selama proses lokalisasi, AMCL mengamati da ta dari sensor yang ada, seperti sensor lidar atau sensor jara k lainnya. Data sensor ini berisi informasi tentang lingkungan sekitar robot, seperti jarak objek, titik-titik fitur, atau la ndmark yang terdeteksi.
4. Sampel Partikel: AMCL menggunakan pendekatan Monte Carlo, di mana sejumlah partikel yang mewakili estimasi posisi robot seca ra acak dihasilkan. Partikel-partikel ini memiliki berbagai es timasi posisi dan orientasi yang mungkin.
5. Perkiraan Pergerakan: Berdasarkan data sensor dan informasi od ometri, setiap partikel diupdate dengan memperkirakan pergerak an robot yang mungkin terjadi. Ini dilakukan dengan menggunakan model kinematika atau model pergerakan yang sesuai dengan ro bot yang digunakan.
6. Pencocokan dengan Grid Map: Setiap partikel dievaluasi dengan m embandingkan pengamatan sensor yang diperoleh dengan informasi pada grid map. Partikel-partikel yang memiliki kesesuaian terb aik dengan data sensor akan memiliki bobot yang lebih tinggi.
7. Resampling: Partikel-partikel yang memiliki bobot yang lebih t inggi akan menjadi lebih dominan dalam representasi estimasi p osisi robot. Proses resampling dilakukan untuk menghasilkan di

stribusi partikel yang lebih fokus pada estimasi yang lebih akurat. Partikel dengan bobot rendah dihapus dan partikel dengan bobot tinggi diperbanyak.

8. Estimasi Posisi: Estimasi posisi robot diambil dari partikel dengan bobot tertinggi atau dengan pendekatan statistik, seperti rata-rata atau median, dari semua partikel yang tersisa setelah resampling.
9. Update Kontinu: Proses langkah-langkah 3 hingga 8 diulang secara berkelanjutan untuk memperbarui estimasi posisi robot secara real-time dengan mengamati data sensor yang terus berubah.

Dengan menggunakan AMCL, robot dapat mengestimasi posisi dan orientasinya dalam lingkungan yang telah dipetakan dengan grid map. Lokalisasi yang akurat memungkinkan robot untuk menavigasi dengan tepat, menjaga posisi relatifnya terhadap peta, dan membuat keputusan yang lebih baik dalam perencanaan jalur, penghindaran rintangan, atau tugas-tugas lainnya dalam sistem kendali robot otonom.



Gambar 39 Visualisasi Lokalisasi dengan AMCL Menggunakan Rviz

### 3.5 Rancangan Pengujian

#### 1. Rancangan Pengujian Data Pengamatan Sensor

Data pengamatan sensor digunakan dalam proses lokalisasi dan pembaruan posisi. Data ini meliputi pengamatan visual dari sensor Kinect, yang dapat berupa deteksi fitur atau poin kunci pada gambar, serta pengamatan lain seperti data IMU (Inertial Measurement Unit) dan data encoder untuk informasi orientasi atau percepatan.

#### 2. Rancangan Pengujian Odometry

Data odometri digunakan untuk memperbaiki estimasi posisi robot. Kita dapat mengambil data odometri dengan menggunakan encoder yang terpasang pada motor robot untuk mengukur putaran roda atau menggunakan data visual odometry dari sensor Kinect yang mengestimasi pergerakan robot berdasarkan perubahan gambar yang diterima.

#### 3. Rancangan Pengujian Ground Truth

Untuk menguji keakuratan hasil pemetaan dan lokalisasi, Kita perlu memiliki data ground truth yang merupakan informasi posisi dan bentuk lingkungan yang sebenarnya. Data ground truth ini dapat diperoleh dengan menggunakan metode pengukuran langsung seperti pemetaan manual menggunakan perangkat ukur atau menggunakan sistem pemetaan yang diketahui keakuratannya.

#### 3.5.1 Evaluasi Hasil Pemetaan dengan Metode RMSE

Root Mean Square Error (RMSE) adalah metode statistik yang digunakan untuk mengukur sejauh mana data atau hasil prediksi membedakan diri dari nilai yang sebenarnya atau referensi. Dalam konteks pemetaan, RMSE digunakan untuk mengevaluasi akurasi posisi dan orientasi robot selama proses navigasi.

taan Simultaneous Localization and Mapping (SLAM), RMSE digunakan untuk mengukur akurasi hasil pemetaan terhadap peta referensi yang sebenarnya. RMSE memberikan nilai numerik yang menggambarkan sejauh mana pemetaan SLAM mendekati peta referensi dengan tingkat kesalahan yang ada.

RMSE dihitung dengan menghitung selisih antara setiap titik data dalam hasil pemetaan dengan titik data yang sesuai dalam peta referensi, kemudian mengkuadratkannya, dan akhirnya menghitung akar kuadrat dari rata-rata kesalahan kuadrat ini. Persamaan matematis RMS E adalah sebagai berikut:

..... (1) \_\_\_\_\_

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \hat{x}_i)^2}$$

di mana:

- N adalah jumlah titik data atau pengukuran.
- $x_i$  adalah nilai pengukuran dalam hasil pemetaan pada titik data ke-i.
- $\hat{x}_i$  adalah nilai yang sesuai dalam peta referensi pada titik data ke-i.

Berikut adalah pseudocode sederhana untuk menghitung RMSE antara hasil pemetaan dan peta referensi dalam bahasa pemrograman Matlab:

```
function rmse = calculateRMSE(mappingResult, referenceMap)

    % Menghitung jumlah titik data atau pengukuran

    N = length(mappingResult);
```

```
% Menginisialisasi variabel untuk menyimpan kesalahan kuadrat
errorSquaredSum = 0;

% Loop untuk menghitung kesalahan kuadrat
for i = 1:N

    % Mengambil nilai dari hasil pemetaan dan referensi
    xi = mappingResult(i);

    xhati = referenceMap(i);

    % Menghitung kesalahan kuadrat pada titik data ke-i
    errorSquared = (xi - xhati)^2;

    % Menambahkan kesalahan kuadrat ke dalam jumlah kesalahan
    errorSquaredSum = errorSquaredSum + errorSquared;

end

% Menghitung rata-rata kesalahan kuadrat dan menghitung akar
% kuadratnya
mse = errorSquaredSum / N;
rmse = sqrt(mse);

end
```

RMSE digunakan dalam evaluasi kinerja pemetaan SLAM karena memberikan metrik numerik yang mudah dipahami untuk mengukur tingkat kesalahan antara hasil pemetaan dengan peta referensi. Semakin rendah nilai RMSE, semakin baik akurasi pemetaan SLAM. Ini membantu peneliti dan praktisi SLAM untuk mengidentifikasi dan mengatasi masalah dalam sistem pemetaan mereka, membandingkan berbagai metode p

emetaan, dan memastikan hasil pemetaan dapat diandalkan dalam berbagai aplikasi, termasuk navigasi otomatis dan penginderaan jarak jauh.

### 3. 5. 2 Absolute trajectory error (ATE)

Pengujian hasil lokalisasi merupakan tahap kritis dalam mengalidasi kinerja sistem Simultaneous Localization and Mapping (SLAM) pada robot yang telah dikembangkan. Salah satu metode yang umum digunakan dalam pengujian ini adalah metode Absolute Trajectory Error (ATE). Metode ATE digunakan untuk mengukur sejauh mana perbedaan antara jalur yang sebenarnya (ground truth) dan jalur yang d'estimasi oleh sistem SLAM.

Metode ATE melibatkan perbandingan antara lintasan yang dihasilkan oleh sistem SLAM dengan lintasan referensi yang dianggap benar (ground truth). Lintasan referensi ini biasanya diperoleh melalui sensor atau sumber lain yang memiliki akurasi tinggi. Setiap titik dalam lintasan estimasi diukur jaraknya terhadap titik yang sesuai dalam lintasan referensi. Jarak ini kemudian diakumulasikan untuk menghitung nilai total kesalahan lintasan. Metode ATE tidak hanya memberikan informasi tentang akurasi posisi robot, tetapi juga menggambarkan kualitas estimasi pergerakan robot secara keseluruhan.

Pengujian ATE dilakukan dalam beberapa tahap. Pertama, lintasan referensi yang akurat harus disiapkan. Ini dapat dilakukan melalui penggunaan lintasan uji khusus yang telah disediakan sebelumnya. Selanjutnya, robot yang dilengkapi dengan sistem SLAM ditempatkan dalam lingkungan pengujian yang telah ditentukan sebelumnya. Robot kemudian diperintahkan untuk menjalani lintasan tertentu, seme-

ntara sistem SLAM akan merekam dan mengestimasi posisi robot sepanjang perjalanan.

Setelah pengujian selesai, data lintasan estimasi dari sistem SLAM dibandingkan dengan data lintasan referensi. Perhitungan ATE dilakukan dengan mengukur jarak antara setiap titik dalam lintasan estimasi dan lintasan referensi yang sesuai. Total kesalahan dihitung dengan mengakumulasikan jarak-jarak ini. Hasilnya dapat dinyatakan dengan menggunakan persamaan dibawah.

ATE didefinisikan sebagai akar rata-rata error kuadrat dari matriks error:

$$ATE_{rmsc} = \left( \frac{1}{n} \sum_{i=1}^n \|ns(E_i) - ns(E_i)\|^2 \right)^{\frac{1}{2}} \quad \dots\dots\dots (4)$$

(Prokhorov, David dkk 2019)

### **3.5.3 Relative pose error (RPE)**

Pengujian kinerja sistem SLAM pada robot yang dikembangkan dapat diperdalam dengan memanfaatkan metode Relative Pose Error (RPE). Metode ini memungkinkan kita untuk mengevaluasi perbedaan antara orientasi dan posisi relatif antara dua lintasan: lintasan referensi (ground truth) dan lintasan estimasi dari sistem SLAM. Dengan menganalisis perbedaan ini, kita dapat mendapatkan wawasan lebih mendalam tentang akurasi pergerakan dan orientasi robot.

Metode RPE melibatkan perbandingan antara parameter-parameter relatif, seperti pergeseran translasi dan rotasi, antara dua lintasan yang diuji. Pada tahap awal, lintasan referensi dan lintasan estimasi direpresentasikan dalam bentuk rangkaian posisi dan orientasi yang direkam sepanjang waktu. Setiap titik waktu dalam lintasan

n estimasi dihubungkan dengan titik waktu yang sesuai dalam lintasan referensi.

Perhitungan RPE melibatkan perbandingan vektor pergeseran translasi dan rotasi antara dua lintasan. Pada setiap titik waktu, perbedaan antara vektor translasi dan rotasi dari lintasan estimasi dengan lintasan referensi dihitung. Nilai ini menggambarkan besarnya kesalahan relatif antara kedua lintasan.

Perhitungan pergeseran translasi dapat menggunakan metrik jarak seperti Euclidean distance, sedangkan perhitungan rotasi umumnya menggunakan representasi quaternion atau matriks rotasi. Kesalahan translasi dan rotasi dapat dihitung secara terpisah atau digabungkan menjadi satu metrik kesalahan.

Setelah pengujian, perhitungan RPE dilakukan dengan membandingkan parameter-parameter relatif pada setiap titik waktu antara lintasan estimasi dan lintasan referensi. Hasilnya adalah deretan nilai error translasi dan rotasi yang merepresentasikan perbedaan antara kedua lintasan.

$$\text{RPE}_{trans}^{i,\Delta} = \left( \frac{1}{m} \sum_{i=1}^m \| \text{trans}(F_i) \|^2 \right)^{\frac{1}{2}} \dots\dots\dots (5)$$

## BAB IV

### ANALISIS DAN PEMBAHASAN

#### 4.1 Pengujian odometry dengan position tracking

Pengujian pembacaan odometri dan position tracking memiliki tujuan untuk mengukur sejauh mana estimasi pergerakan robot dari se

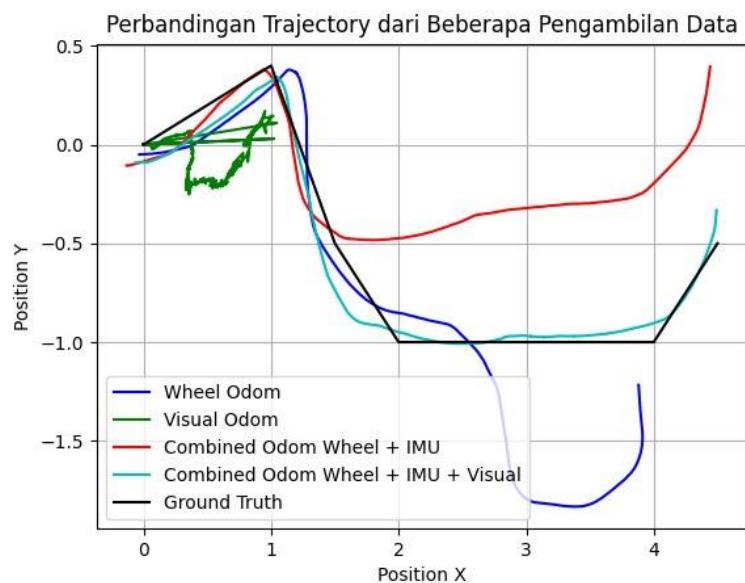
nsor-sensor tertentu dapat merefleksikan pergerakan sebenarnya di dunia nyata. Dalam sub bab ini, penulis akan membandingkan lintasan yang dihitung berdasarkan data odometri dari berbagai sensor, seperti encoder, IMU (Inertial Measurement Unit), dan visual odometry dari Kinect, dengan lintasan sebenarnya yang telah dipersiapkan sebelumnya. Analisis perbedaan antara kedua lintasan ini akan memberikan wawasan tentang akurasi dan keandalan estimasi pergerakan robot.

Pada pengujian ini, penulis memanfaatkan beberapa sensor untuk mendapatkan data odometri yang diperlukan untuk mengestimasi pergerakan robot. Encoder digunakan untuk mengukur perubahan putaran roda yang dikonversi menjadi pergerakan translasi. IMU digunakan untuk mendeteksi perubahan orientasi dan percepatan robot. Visual odometry dari Kinect digunakan untuk memanfaatkan data visual dalam mengestimasi pergerakan relatif terhadap lingkungan sekitar.

Pada tahap implementasi, robot ditempatkan dalam lingkungan pengujian yang telah dipersiapkan sebelumnya. Robot diinstruksikan untuk menjalani lintasan tertentu yang mencakup variasi pergerakan translasi dan rotasi. Selama perjalanan, data odometri dari masing-masing sensor direkam. Lintasan sebenarnya yang diharapkan dari pergerakan robot juga direkam sebagai ground truth



Gambar 40 Pengujian Pembacaan Odometry



Gambar 41 Perbandingan trajectory dari sumber odometry berbeda terhadap ground truth

Hasil pengujian pembacaan odometri dengan position tracking akan memberikan pemahaman tentang performa sensor-sensor yang digunakan dalam mengukur pergerakan robot. Data odometri yang berasal dari berbagai sensor dapat dibandingkan untuk mengidentifikasi sensor mana yang memberikan estimasi yang lebih akurat dan stabil dalam berbagai situasi pergerakan.

$$\max. deviation = \max(\sqrt{(x_i^* - x_i)^2 + (y_i^* - y_i)^2}), \dots \dots \dots (2)$$

$$\text{aver. deviation} = \frac{1}{N} \sum_{i=1}^N \sqrt{(x_i^* - x_i)^2 + (y_i^* - y_i)^2} ) \dots \dots \dots (2)$$

Dimana  $X_i^*$   $Y_i^*$  adalah koordinat dari garis penanda hasil kalkulasi *trajectory* dan N adalah jumlah grid

Tabel 5 Simpangan odometry terhadap ground truth

No	Sumber Odometry	Simpangan rata-rata	Simpangan maksimal
1	Wheel	0.37071054754957544	0.8190219777270937
2	Visual	1.7723274719631417	2.9472869049347743
3	Wheel + IMU	0.6365240992350174	0.9856089488230106
4	Wheel + IMU + Visual	0.016235338306387802	0.09144397191723462
	1		

Pada tabel di atas dapat diamati, data deviasi pergerakan antara estimasi dan ground truth telah dianalisis. Dalam konteks ini, ditemukan bahwa sumber odometri "Wheel" memiliki simpangan rata-rata sebesar 0.371 dan simpangan maksimal sebesar 0.819. Sumber odometri "Visual," di sisi lain, menunjukkan tingkat deviasi yang lebih tinggi, dengan simpangan rata-rata sebesar 1.772 dan simpangan maksimal sebesar 2.947. Sumber odometri "Wheel + IMU" menghasilkan simpangan rata-rata sebesar 0.637 dan simpangan maksimal sebesar 0.986. Sementara itu, penggunaan semua sumber odometri, yaitu "Wheel + IMU + Visual," menghasilkan hasil yang paling akurat, dengan simpangan rata-rata sebesar 0.016 dan simpangan maksimal sebesar 0.091. Hasil ini menunjukkan bahwa penggabungan data dari berbagai sumber odometri secara bersamaan dapat meningkatkan akurasi estimasi pergerakan robot dalam penelitian ini.

#### 4.2 pengujian hasil pemetaan

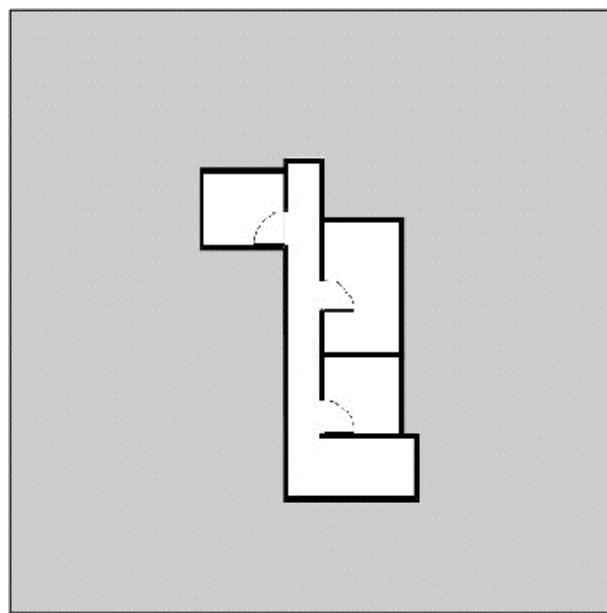
Pada tahap ini, akan membahas pengujian hasil pemetaan yang dihasilkan oleh sistem Simultaneous Localization and Mapping (SLAM) dengan memanfaatkan sensor Kinect. Pengujian ini bertujuan untuk

engevaluasi akurasi dan kualitas peta yang dihasilkan oleh sistem pemetaan dalam berbagai skenario lingkungan dan kondisi. Peta yang dihasilkan akan dibandingkan dengan peta sesungguhnya dari ruangan uji, yang akan dijadikan ground truth.

#### **4.2.1 Pemetaan pada ruang uji berbeda**

Pengujian dilakukan pada tiga ruang uji yang berbeda untuk mempertimbangkan variasi dalam ukuran, bentuk, dan fitur ruangan. Ruang uji yang berbeda akan memberikan pandangan tentang sejauh mana sistem SLAM dapat menghadapi perbedaan lingkungan dan bagaimana itu memengaruhi hasil pemetaan.

Ruang Uji 1 (LAB Sistem Kendali dan Instrumentasi)



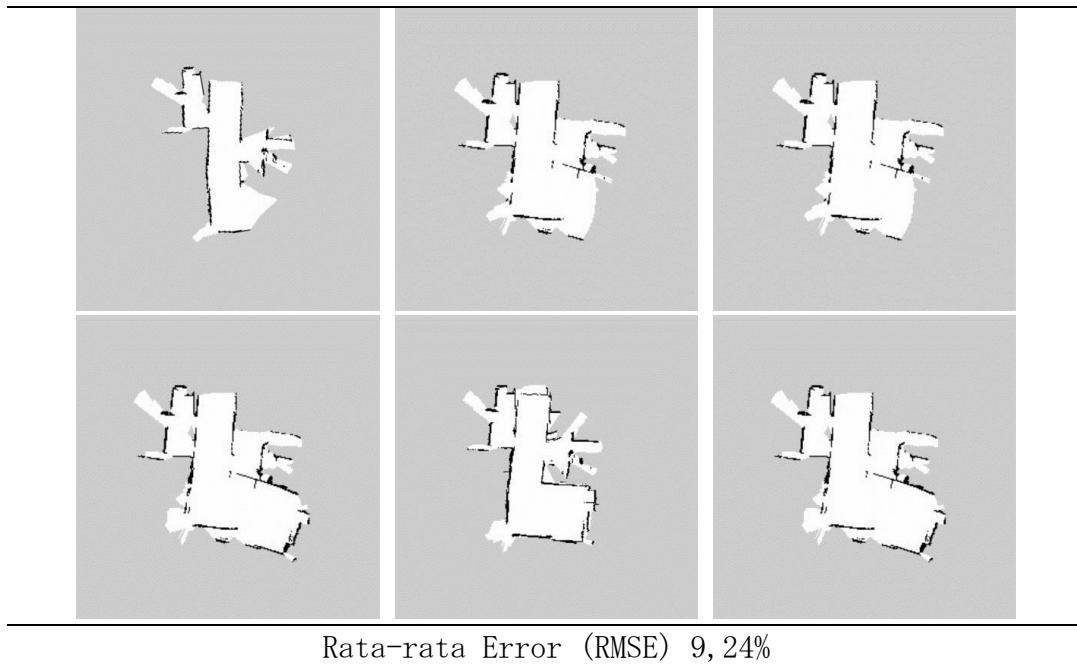
Gambar 42 Denah Lab Sistem kendali dan Instrumenasi

Tabel 6 Hasil pemetaan ruang uji 1

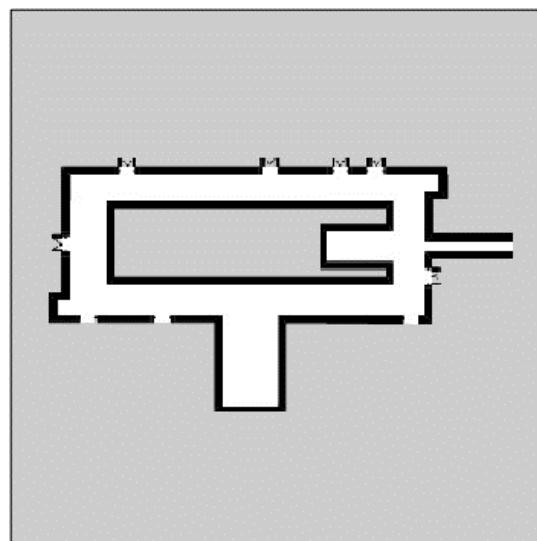
---

Ruang Uji 1 (Lab Sistem Kendali dan Instrumenasi)

---



Ruang Uji 2 (Gedung COT Lantai 2)



Gambar 43 Denah Gedung COT Lantai 2

Tabel 7 Hasil pemetaan ruang uji 2

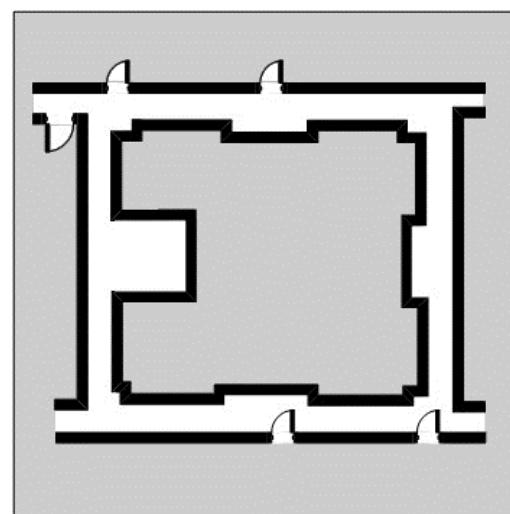
---

Ruang Uji 2 (Gedung COT Lantai 2)

---



Ruang Uji 3 (Gedung Elektro Lantai 3)



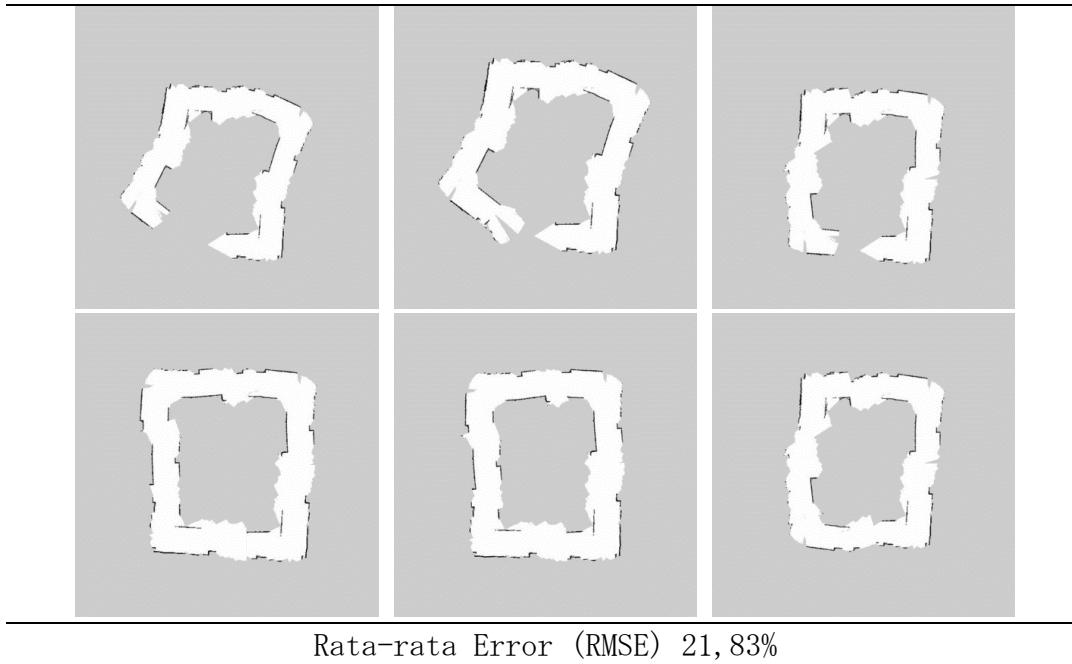
Gambar 44 Denah Gedung Elektro Lantai 3

Tabel 8 Hasil pemetaan ruang uji 3

---

Ruang Uji 3 (Gedung Elektro Lantai 3)

---



Dari ketiga ruangan uji didapatkan persentase beda terbesar antara hasil pemetaan dengan ground truth yaitu 30.54% pada ruang uji 3, sedangkan persentase beda terkecil adalah 4.90% pada ruang uji 1. Hal ini menunjukkan bahwa parameter luas lingkungan yang dipetakan memiliki pengaruh signifikan dimana semakin luas area ruangan yang dipetakan maka akurasi pemetaan akan semakin menurun.

Penurunan akurasi pemetaan pada area luas dapat dipengaruhi oleh banyak faktor. Namun, salah satu yang paling signifikan dalam penelitian ini adalah faktor error odometry yang akan terakumulasi seiring berjalannya proses pemetaan. Hal ini juga menunjukkan pengaruh signifikan dari akurasi odometry terhadap hasil pemetaan dengan SLAM.

#### **4.2.2 Pemetaan dengan tingkat *Luminansi berbeda***

Dalam pengujian ini, penulis akan mempertimbangkan pengaruh tingkat luminansi yang berbeda pada hasil pemetaan. Pengujian akan mencakup situasi dengan dan tanpa paparan sinar matahari. Variasi ini akan menguji kemampuan sensor Kinect dalam berbagai kondisi pe-

ncahayaan dan bagaimana itu memengaruhi kualitas peta yang dihasilkan.

Pengujian pada kondisi luminensi skenario 1



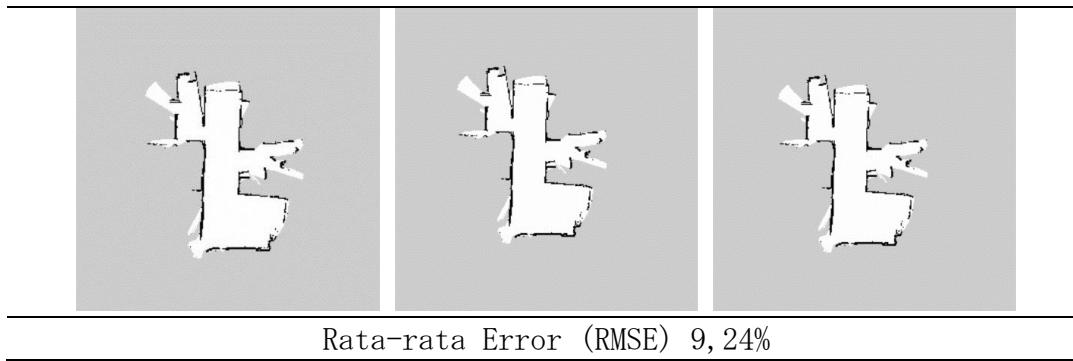
Gambar 45 Pengujian pada skenario 1



Gambar 46 Data lux pada skenario 1

Tabel 9 Hasil pemetaan pada kondisi lux skenario 1

Skenario Uji 1 (malam hari) (rata-rata lumen : 430 lux)



Pengujian pada kondisi luminensi skenario 2



Gambar 47 Pengujian pada skenari  
o 2



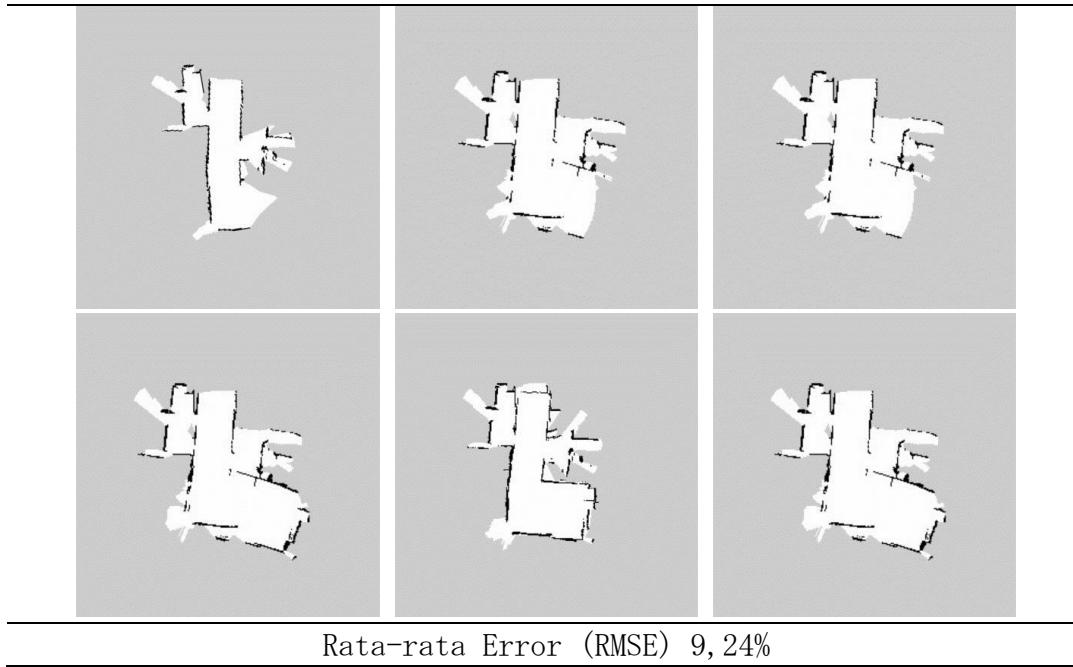
Gambar 48 Data lux pada skenari  
o 2

Tabel 10 Hasil pemetaan pada kondisi lux skenario 3

---

Skenario Uji 2 (siang hari) (rata-rata lumen : 1121 lux)

---



Dari hasil pengujian dengan luminensi berbeda yang telah dilakukan dapat diketahui bahwa faktor luminensi mempengaruhi kinerja sensor Kinect. Hal ini menyebabkan gangguan pada sistem odometry visual dan proses pembuatan peta. dapat diamati bahwa pada skenario pengujian 1 peta yang dihasilkan memiliki tingkat persentase perbedaan dengan ground truth terkecil yaitu hanya 4,90% pada luminensi rata-rata 430 lux. Sedangkan, pada pengujian skenario 2 dengan luminensi rata-rata mencapai 1623 lux dengan paparan sinar matahari menyebabkan hasil pemetaan yang lebih buruk dengan persentase perbedaan dengan ground truth yang sangat besar yatu 9,24%

#### **4.3 pengujian kemampuan lokalisasi posisi robot dalam peta**

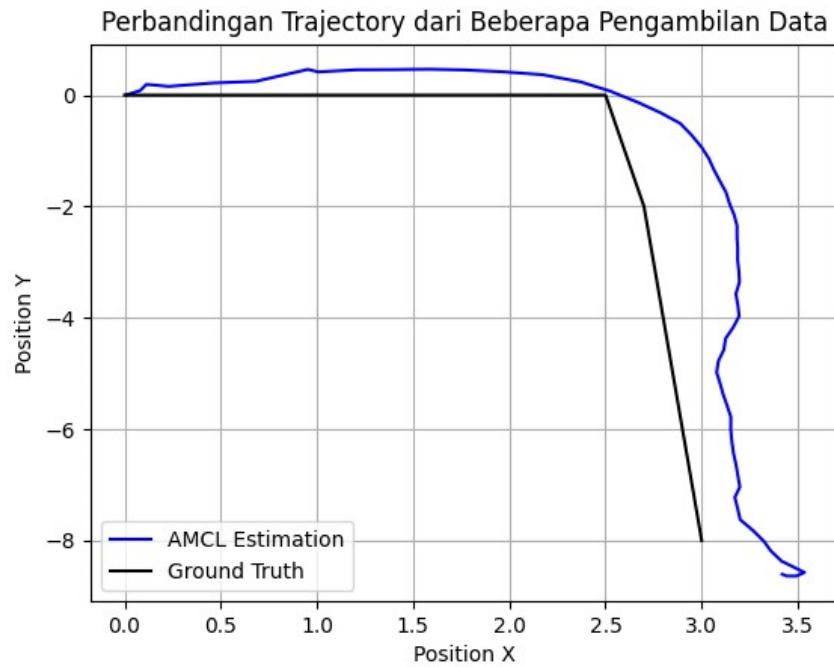
##### **4.3.1 Hasil Pengujian ATE dan RPE**

Sub bab ini akan merinci hasil pengujian sistem dengan fokus pada Error Translasi Absolut (Absolute Translational Error, ATE) dan Error Posisi Relatif (Relative Pose Error, RPE). Pertama, kami akan memperlihatkan jalur yang ditempuh oleh robot dalam pengujian

ini. Selanjutnya, kami akan menjalankan program yang secara cermat mencatat pose dan posisi robot saat robot berjalan menyusuri lintasan di dalam peta. Setelah itu, hasil pengukuran tersebut akan digunakan untuk menghitung nilai ATE dan RPE yang mengukur sejauh mana robot mendekati ground truth.



Gambar 49 Visualisasi Rviz Robot AMCL Estimasi Pose



Gambar 50 Perbandingan Trajectory Lokalisasi AMCL Terhadap Ground Truth  
Gambar

Tabel 11 Analisis ATE dan RPE terhadap hasil estimasi lokalisasi

Timestamp	X <sub>gt</sub>	y <sub>gt</sub>	yaw <sub>gt</sub>	X <sub>est</sub>	y <sub>est</sub>	yaw <sub>est</sub>	ATE	RPE
169329006 3	0	0	0	0.0089	0.002	0.00455	0.009	0.01
				489313	78771	0594451	37308	0419
				86	9609		6662	3408
169329008 2	0.	0	0	0.0809	0.081	0.19538	0.426	0.46
	5			066248	05240	03586	85916	9448
				7	448		81	8618
169329008 3	1	0	0	0.1132	0.192	0.35272	0.907	0.97
				059153	06180	98183	35411	3503
				88			34	8838
169329008 4	1.	0	0	0.2293	0.155	0.09002	1.280	1.28
	5			010785	80655	850812	21538	3377
				27			5	015
169329021 2	2	0	0	3.1837	-2.55	-1.5097	2.813	3.19
				27376	22945	29843	43525	2914
				47			9	367

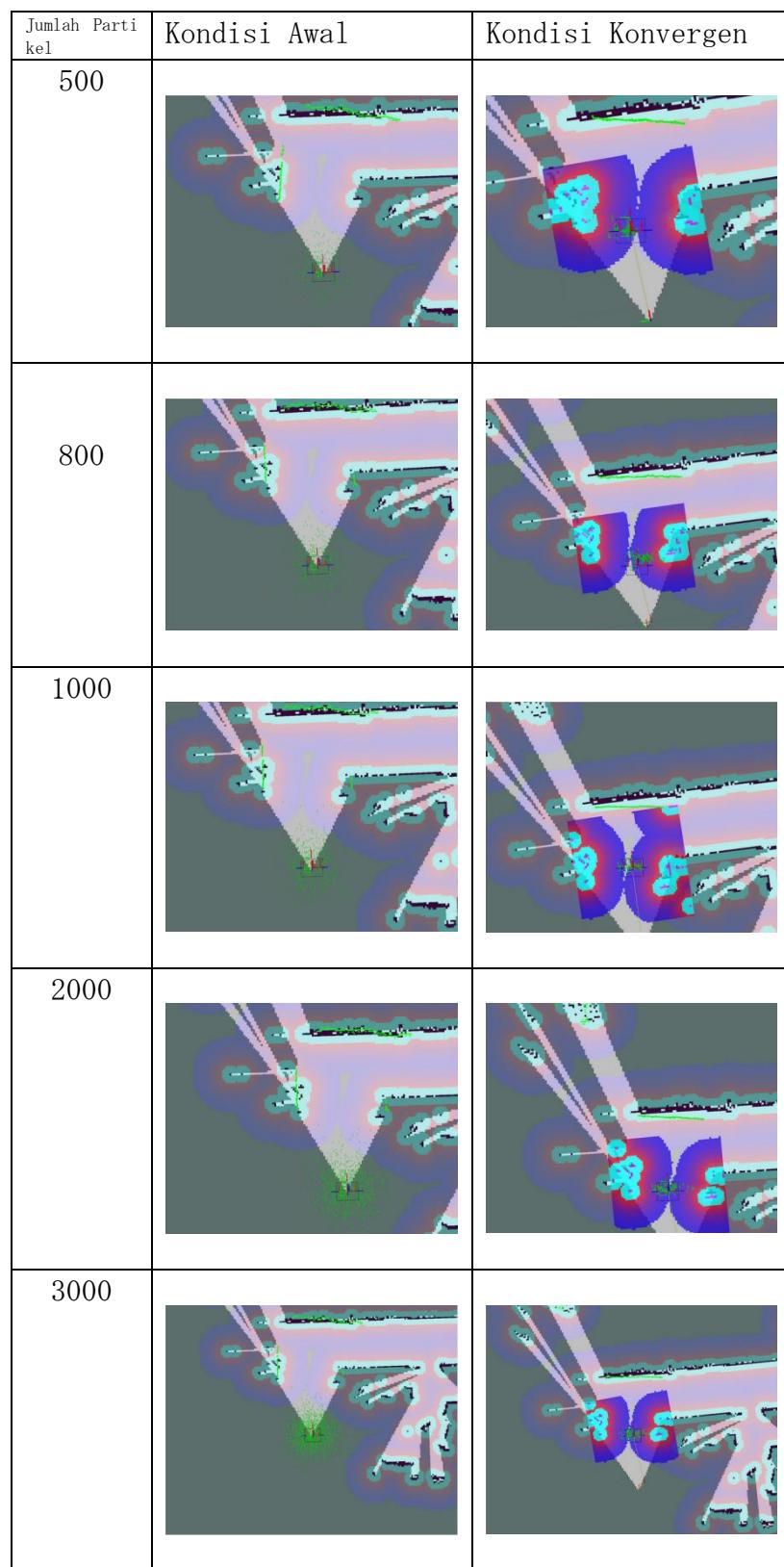
Timestamp	X <sub>gt</sub>	Y <sub>gt</sub>	yaw <sub>gt</sub>	X <sub>est</sub>	Y <sub>est</sub>	yaw <sub>est</sub>	ATE	RPE
169329021 8	2. 5	0	0	3.1870 89566 45	-2.75 06361	-1.5263 09435	2.835 15277 8	3.21 9893 13
169329022 2	2. 6	-1	-1	3.1963 68612 95	-3.35 79437	-1.5525 22084	2.432 19128 8	2.49 4160 203
169329022 7	2. 7	-2	-1	3.4931 94596 47	-8.63 72090	-1.8058 75409	6.684 43726 9	6.73 2840 171
169329023 2	2. 8	-4	-1.5	3.4427 18991 42	-8.63 69706	-2.0826 2312	4.681 30157 5	4.71 7418 164
169329035 7	3	-8	-2	3.4183 13793 76	-8.60 66387	-2.2756 83922	0.736 88332 46	0.78 6764 6783
Rata-rata							2.280 72032 5	2.38 8073 981

Hasil evaluasi menunjukkan bahwa sistem SLAM ini memiliki tingkat akurasi yang memadai, dengan nilai ATE (Absolute Translation Error) sebesar 2.28 dan RPE (Relative Pose Error) sebesar 2.39. Temuan ini mengindikasikan bahwa sistem mampu mendekati posisi dan orientasi sebenarnya dengan tingkat kesalahan yang relatif rendah.

#### 4.3.3 Pengujian Penentuan posisi dengan partikel filter

Pada pengujian ini dilakukan dengan mengujicobakan beberapa partikel dengan jumlah yang berbeda-beda yaitu 500, 800, 1000, 2000 dan 3000 partikel. Untuk sensoring model digunakan sistem persepsi yang telah dijelaskan pada bab III. Berikut ini hasil pengujian dengan menggunakan jumlah partikel yang berbeda. (partikel yang dimaksud adalah istilah untuk menggambarkan titik kemungkinan keberadaan robot didalam peta)

Tabel 12 visualisasi lokalisasi dengan jumlah partikel berbeda



Dari semua data hasil pengujian simulasi menggunakan 200, 400, dan 600 partikel didapatkan data sebagai berikut :

Tabel 13 Perbandingan waktu untuk mencapai konvergensi pada jumlah partikel berbeda

Robot Position			500 Particles			800 Particles			1000 Particles			2000 Particles			3000 Particles		
x	y	$\theta$	x	y	$\theta$	x	y	$\theta$	x	y	$\theta$	x	y	$\theta$	x	y	$\theta$
1 .6 05	1 58 4	-1 .6 05	1, 24 4	0, ,6 05	-1 34 5	1, 09 05	0, .6 05	-1 33 4	1. 03 05	-0, ,6 05	-1 94 7	1, 88 05	-1 ,6 05	0, 57 3	0, 02 ,6 05	-1 3 05	
Error = $\frac{x - x_p}{x} \cdot 100\%$	5 8	75 ,6	0	34 ,5	9 1	0	33 ,4	0 9	10 9		94 ,7	1 2		4 3	47 ,7	0	
Waktu (detik)	00.13.34			00.20.67			00:23.33			00:24.84			00:27.45				

Dari data diatas dapat dianalisa bahwa semakin banyak jumlah partikel maka waktu yang dibutuhkan semakin lama tapi jumlah sumber daya untuk mencapai konvergen partikel lebih sedikit. Dan sebaliknya jika digunakan partikel dalam jumlah kecil maka waktu yang diperlukan lebih singkat tapi dibutuhkan jumlah sumber daya yang lebih banyak.

## BAB V

### KESIMPULAN DAN SARAN

#### 5.1 Kesimpulan

Pada penelitian ini, dapat disimpulkan beberapa hasil sebagai berikut :

1. Penggunaan sumber odometri yang beragam seperti roda, sensor IMU, dan data visual secara bersamaan meningkatkan akurasi estimasi pergerakan robot. Penggabungan menggunakan EKF ini menghasilkan hasil yang sangat akurat, dengan deviasi rata-rata yang sangat rendah (0.016 m) dan deviasi maksimal yang minimal (0.091 m). Dengan demikian, pendekatan ini sangat dianjurkan untuk aplikasi yang memerlukan estimasi pergerakan robot yang akurat dan presisi.
2. Pada pengujian dengan bentuk ruangan berbeda dapat diketahui pengaruh luas area pemetaan terhadap peta yang dihasilkan. Sistem menunjukkan performa terbaik pada ruang uji 1 yang memiliki luas area terkecil diantara ruang uji lainnya yang menghasilkan peta dengan persentase perbedaan 4,90% terhadap ground truth, sedangkan pada area uji 3 dengan luas area terbesar diantara ruang uji lainnya sistem menghasilkan peta yang kurang baik dengan persentase perbedaan 30,54% terhadap ground truth
3. Selain faktor luas area, faktor tingkat pencahayaan pada ruang uji juga mempengaruhi hasil pemetaan. Performa sistem terbaik didapatkan pada pengujian dengan tingkat pencahayaan rata-rata 430 lux yang menghasilkan akurasi peta terbaik yaitu perbedaan 4,90% terhadap ground truth. Sedangkan, pada pengujian dengan rata-rata tingkat pencahayaan 1623 lux disertai

dengan paparan langsung sinar matahari menghasilkan error se besar 12, 12% pada ruangan yang sama

4. Hasil evaluasi menunjukkan bahwa sistem SLAM ini memiliki tingkat akurasi yang memadai, dengan nilai ATE (Absolute Translation Error) sebesar 2.28 dan RPE (Relative Pose Error) sebesar 2.39. Temuan ini mengindikasikan bahwa sistem mampu mendekati posisi dan orientasi sebenarnya dengan tingkat kesalahan yang relatif rendah.
5. Berdasarkan eksperimen yang dilakukan dengan mengubah jumlah partikel dalam AMCL saat menjalankan SLAM dalam ROS, dapat disimpulkan bahwa peningkatan jumlah partikel dari 500 hingga 1000 secara signifikan meningkatkan akurasi estimasi posisi robot, dengan error relatif yang menurun dari 58% menjadi 33 %. Namun, setelah mencapai 1000 partikel, peningkatan lebih lanjut dalam jumlah partikel tidak berdampak besar pada akurasi. Selain itu, perlu diperhatikan bahwa peningkatan jumlah partikel juga meningkatkan waktu komputasi, meskipun masih dalam kisaran yang dapat diterima. Oleh karena itu, dalam aplikasi SLAM, harus mempertimbangkan kompromi antara akurasi dan waktu komputasi serta mengadaptasi jumlah partikel sesuai dengan kebutuhan spesifik sistem dan lingkungan yang digunakan.

## 5.2 Saran

Adapun saran yang dapat penulis berikan diantaranya :

1. Untuk meningkatkan akurasi pembacaan odometry dan menghindari terjadinya odometry lost, penempatan sensor Kinect pada posisi yang dapat memiliki tangkapan yang lebih luas terhadap fitur ruangan yang ingin di petakan dapat dipertimbangkan

2. Sebaiknya proses pemetaan dilakukan pada kondisi pencahayaan ruangan yang homogen tanpa adanya pengaruh Cahaya luar seperti sinar matahari untuk menghasilkan peta yang lebih akurat
3. Dapat meningkatkan hasil pemetaan dengan menggunakan pendekatan ataupun metode SLAM yang lebih mutakhir

## DAFTAR PUSTAKA

- Alamulhuda, M., Wahyudi, W., & Afrisal, H. (2023). PERANCANGAN GRASPING CONTROL PADA ROBOT TANGAN MENGGUNAKAN REINFORCEMENT LEARNING. *Transient: Jurnal Ilmiah Teknik Elektro*.
- Ajang, N., & Hendriyawan, M. S. (2019). Implementasi ROS (Robot Operating System) Pada Sistem Kendali Jarak Jauh Robot Bergerak Jenis Non-holonomic [Manuskrip tak dipublikasikan]. Program Studi Teknik Elektro, Fakultas Informasi Teknologi dan Elektro, Universitas Teknologi Yogyakarta.
- Anshar, Muh. (2010). Directed Learning-based Evolutionary Approach for Legged Robot Motion. Lambert Academic Publishing. Australia.

CuriousInventor. (2013, 16 Februari). How the Kinect Depth Sensor Works in 2 Minutes. <https://www.youtube.com/watch?v=uq9SEJxZiUg>

Dudek, M. J. (2008). Inertial Sensors, GPS, and Odometry. Dalam B. Siciliano & O. Khatib (Eds.), Springer Handbook of Robotics (hal. 477–490). Springer Berlin Heidelberg.

Firmansyah, R.A. (2015). Sistem Auto Docking Pada Service Robot Menggunakan Persepsi Visual.

Francis, K., & Bird, B. (2011, 28 Juni). Kinect Hackers Are Changing the Future of Robotics. *Wired.com*. [https://www.wired.com/2011/06/mf\\_kinect/](https://www.wired.com/2011/06/mf_kinect/)

Goris, Kristof. (2005). Autonomous Mobile Robot Mechanical Design. Ixelles: Vrije Universiteit Brussel.

Indrawan, Gede. (2008). Perancangan dan Implementasi Kecerdasan-Batan Robot Pencari Jalur Berbasis Mikrokontroler Basic Stamp. Tesis, Universitas Indonesia.

Kinect. (2017, 7 Januari). Wikipedia. <https://en.wikipedia.org/wiki/Kinect>

Labbé, M., & Michaud, F. (2018). Long-term online multi-session graph-based SLAM with memory management. *Autonomous Robots*, 42, 1133–1150. Springer.

- Labbé, M., & Michaud, F. (2019). 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*, 36, 416–446. Wiley Online Library.
- Mahandi, Y.D. (2021). Deteksi Objek Untuk Robot Bergerak Menggunakan Kamera Omnidirectional Berbasis Fitur Warna.
- MathWorks. (Tidak Tersedia). Localize TurtleBot Using Monte Carlo Localization. MathWorks. <https://www.mathworks.com/help/nav/ug/localize-turtlebot-using-monte-carlo-localization.html>
- Pranoto, B., & Firdaus, A.R. (2021). Rancang Bangun Lengan Robot dengan Sistem Kontrol Otomatis dan Human Machine Interface untuk Mesin Operasional Industri Manufaktur. *Jurnal Energi dan Teknologi Manufaktur (JETM)*.
- Prasetyo, T.H., Sirajuddin, I., & Sungkono, S. (2021). Sistem Kendali Wall Following Pada Mobile Robot Dengan Penggerak Mekanik Menggunakan Metode Fuzzy. *Jurnal Elektronika dan Otomasi Industri*.
- Riyanti, A., Pratiwi, N.D., Nainggolan, N.Y., Sardi, N.R., & Satrya, A.B. (2021). GLOBALISASI DAN TRANSFER TEKNOLOGI: PENOPANG INDUSTRI MANUFAKTUR PADA PERKEMBANGAN MARKETPLACE DI REGIONAL ASEAN.
- Shobirin, M. (2016). Sistem Kendali Nirkabel Robot Bulutangkis Berbasis Mikrokontroller.

Sidiq, I.M., Ihsanto, E., & Yuliza, Y. (2020). Rancang Bangun Sistem Navigasi Robot Hexapod 3 DOF di Ruangan Labirin Menggunakan Metode Fuzzy Sugeno Berbasis Arduino.

Springer Handbook of Robotics. (2008). Edited by B. Siciliano and O. Khatib. Springer.

Supriyanto, R., Hustinawati., Nugraini, R. W., Kurniawan, A. B., Pernadi, Y., Sa'ad, A. 2010. Robotika. Depok: Universitas Gunadarma.

Thrun, S., Burgard, W., & Fox, D. (2005). Probabilistic Robotics. The MIT Press.

Utomo, E.Y. (2015). Autonomous mobile robot berbasis landmark menggunakan particle filter dan occupancy grid maps untuk navigasi, penentuan posisi dan pemetaan.

Wicaksono, I. (2019). Navigasi Mobile Robot Darat Menggunakan Odometri Visual Berbasis Citra Stereo (Tesis Tugas Akhir, EE 184801). Institut Teknologi Sepuluh Nopember.

Zainuddin, N., Mustafah, Y., Shawgi, Y., & Rashid, N. M. (2014). Autonomous Navigation of Mobile Robot Using Kinect Sensor. Dalam International Conference on Computer & Communication Engineering, Kuala Lumpur, Malaysia.

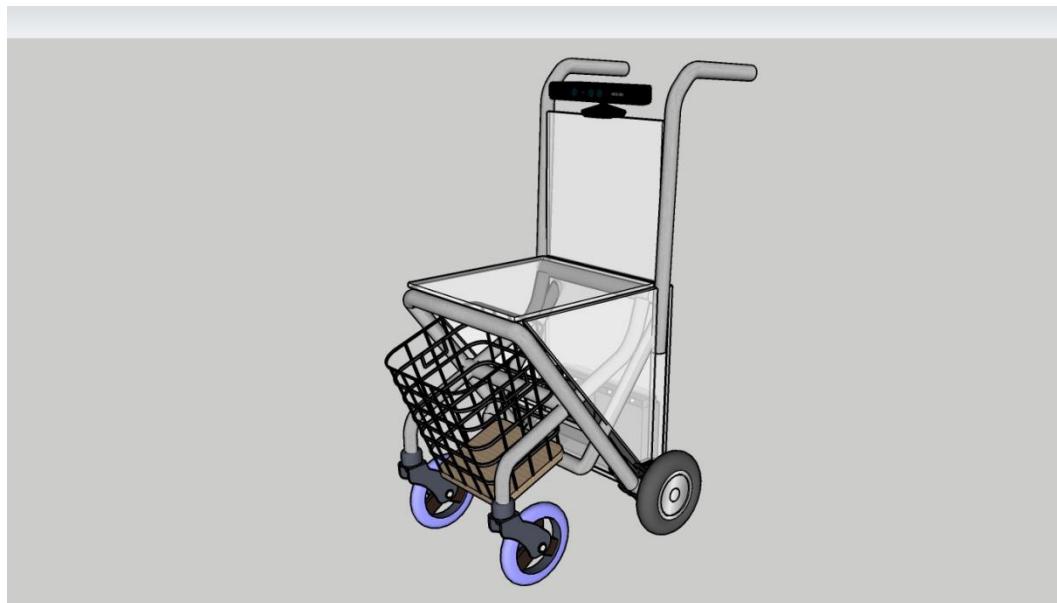
Zamisyak, O., Prayogo, S.I., & Ahmad, B.S. (2016). Educational Multifunction Robot (Indobot) sebagai Robot Edukasi.



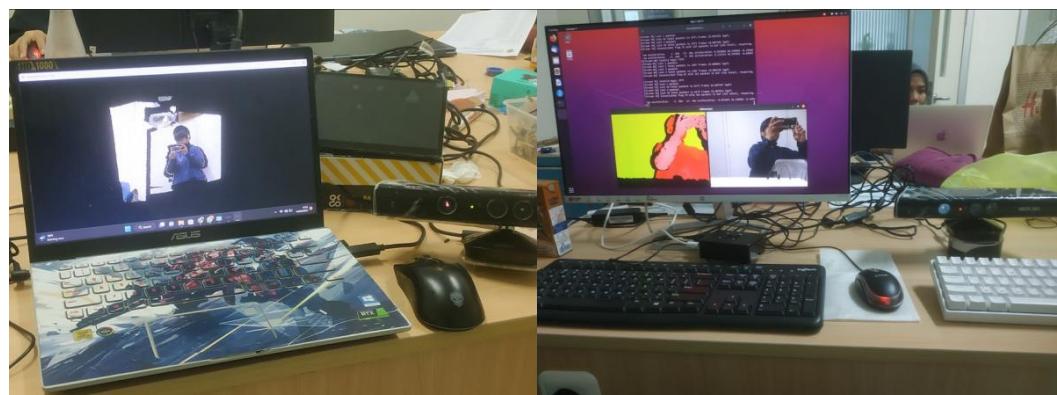
## LAMPIRAN

### Lampiran 1 Dokumentasi Pelaksanaan Penelitaian

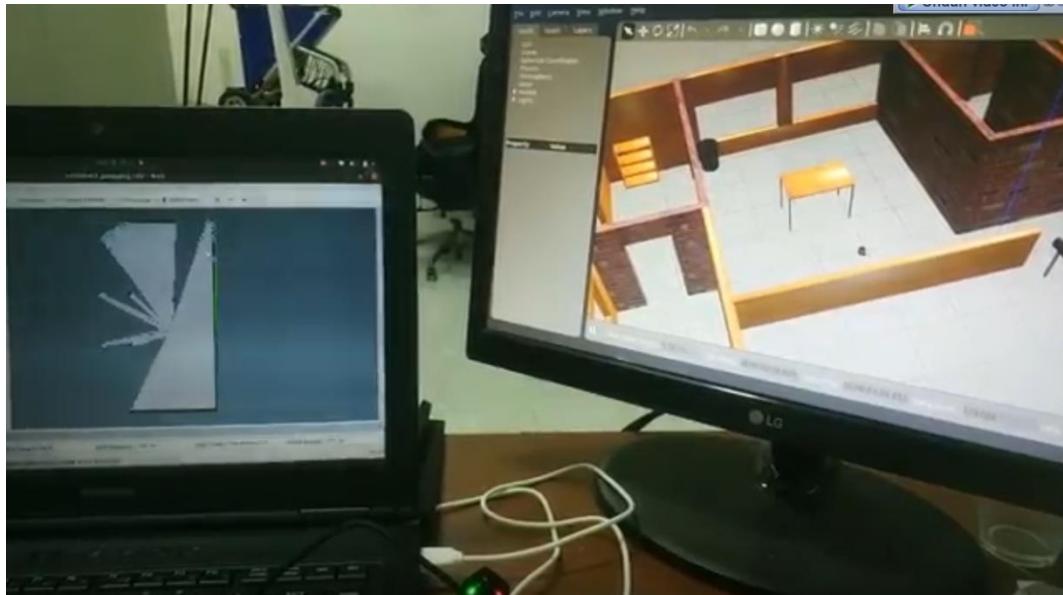
#### a. Tahap Perancangan



Rancangan Awal Desain Pada Software Sketchup dan Pemodelan URDF



Modifikasi Kamera Xbox Kinect Untuk Aplikasi Pada Robot



Pemodelan Algoritma SLAM dengan Software Simulasi Gazebo

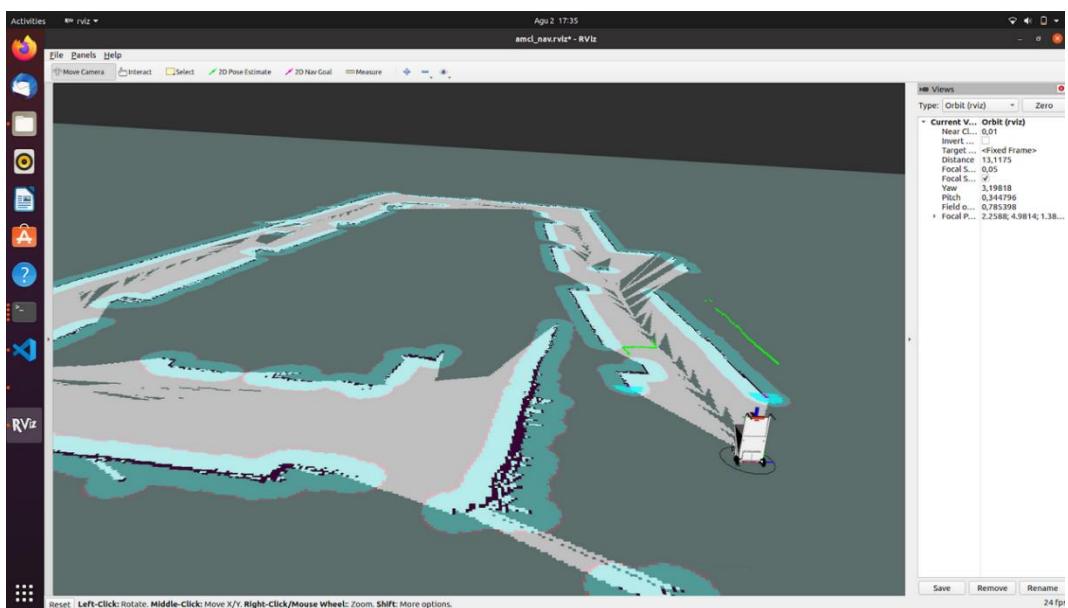


Instalasi perangkat keras

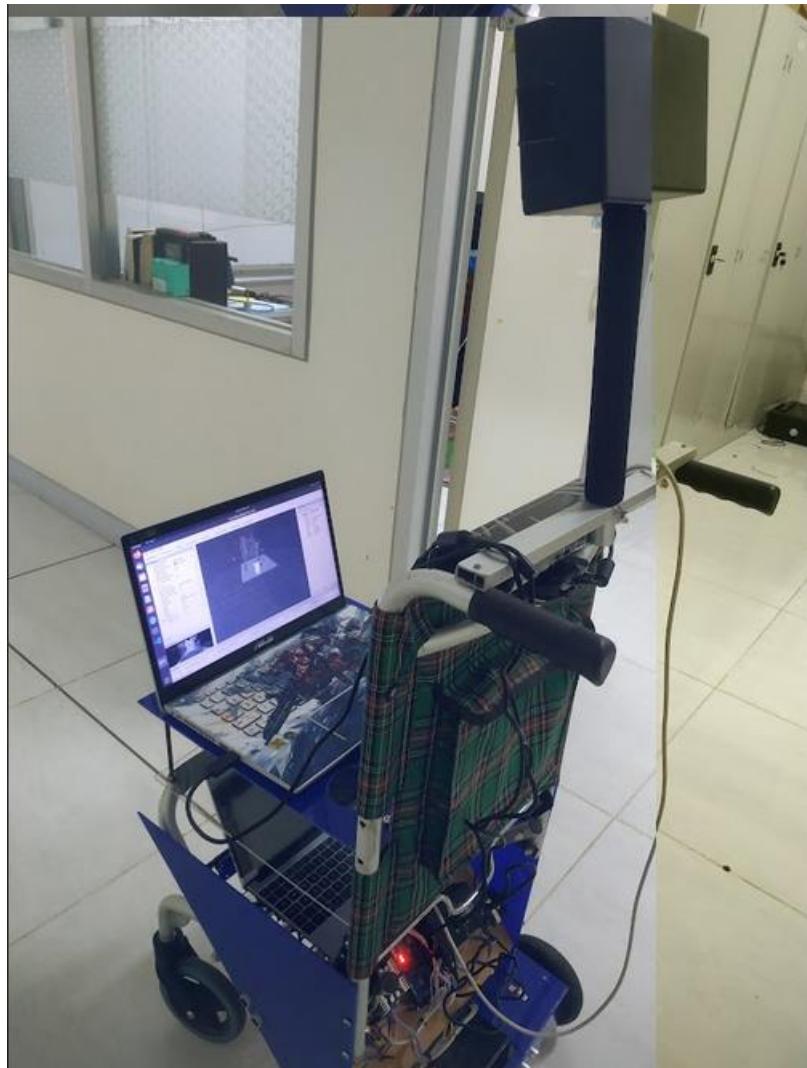
### b. Tahap Pengujian



Pengujian Estimasi Odometry Robot Mengikuti Jalur yang Telah ditetapkan



Pengujian Pemetaan Koridor Lantai 2 Gedung COT



Pengujian Pemetaan Lab Sistem Kendali dan Instrumentasi

c. Video Pengujian



<https://www.youtube.com/watch?si=DEtySbQt1YJKWk0g&v=tmyqf-2GT2k&feature=youtu.be>

## Lampiran 2 Kode Pemrograman Low Level

### a. Kendali Motor dan Pembacaan Encoder

```
#include <ros.h>

#include <std_msgs/Int16.h>

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

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

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

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

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

// Keep track of the number of wheel ticks
```

```
std_msgs::Int16 right_wheel_tick_count;  
ros::Publisher rightPub("right_ticks", &right_wheel_tick_count);  
  
std_msgs::Int16 left_wheel_tick_count;  
ros::Publisher leftPub("left_ticks", &left_wheel_tick_count);  
  
// 100ms interval for measurements  
const int interval = 100;  
long previousMillis = 0;  
long currentMillis = 0;  
  
// Increment the number of ticks  
void right_wheel_tick() {  
  
    // Read the value for the encoder for the right wheel  
    int val = digitalRead(ENC_IN_RIGHT_B);  
  
    if(val == LOW) {  
        Direction_right = false; // Reverse  
    }  
    else {  
        Direction_right = true; // Forward  
    }  
  
    if (Direction_right) {  
  
        if (right_wheel_tick_count.data == encoder_maximum) {  
            right_wheel_tick_count.data = encoder_minimum;  
        }  
    }  
}
```

```
    }

    else {
        right_wheel_tick_count.data++;
    }

}

else {

    if (right_wheel_tick_count.data == encoder_minimum) {

        right_wheel_tick_count.data = encoder_maximum;
    }

    else {

        right_wheel_tick_count.data--;
    }

}

// Increment the number of ticks

void left_wheel_tick() {

    // Read the value for the encoder for the left wheel

    int val = digitalRead(ENC_IN_LEFT_B);

    if(val == LOW) {

        Direction_left = true; // Reverse
    }

    else {

        Direction_left = false; // Forward
    }
}
```

```
if (Direction_left) {  
    if (left_wheel_tick_count.data == encoder_maximum) {  
        left_wheel_tick_count.data = encoder_minimum;  
    }  
    else {  
        left_wheel_tick_count.data++;  
    }  
}  
  
else {  
    if (left_wheel_tick_count.data == encoder_minimum) {  
        left_wheel_tick_count.data = encoder_maximum;  
    }  
    else {  
        left_wheel_tick_count.data--;  
    }  
}  
  
}  
  
void setup() {  
  
    // Set pin states of the encoder  
    pinMode(ENC_IN_LEFT_A , INPUT_PULLUP);  
    pinMode(ENC_IN_LEFT_B , INPUT);  
    pinMode(ENC_IN_RIGHT_A , INPUT_PULLUP);  
    pinMode(ENC_IN_RIGHT_B , INPUT);  
  
    // Every time the pin goes high, this is a tick
```

```

    attachInterrupt(digitalPinToInterrupt(ENC_IN_LEFT_A), left_wheel_tick,
RISING);

    attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGHT_A), right_wheel_tic
k, RISING);

// ROS Setup

nh.getHardware()->setBaud(115200);

nh.initNode();

nh.advertise(rightPub);

nh.advertise(leftPub);

}

void loop() {

// Record the time

currentMillis = millis();

// If 100ms have passed, print the number of ticks

if (currentMillis - previousMillis > interval) {

previousMillis = currentMillis;

rightPub.publish( &right_wheel_tick_count );

leftPub.publish( &left_wheel_tick_count );

nh.spinOnce();

}

}

```

### b. Interface MPU6050

```
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// ADO low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// ADO high = 0x69

MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for ADO high

/*
=====
=====

NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
depends on the MPU-6050's INT pin being connected to the Arduino's
external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
digital I/O pin 2.

* =====
===== */

```

```
/* =====
====
```

NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error

when using `Serial.write(buf, len)`. The Teapot output uses this method

The solution requires a modification to the Arduino USBAPI.h file, which

is fortunately simple, but annoying. This will be fixed in the next IDE

release. For more info, see these links:

<http://arduino.cc/forum/index.php/topic,109987.0.html>

<http://code.google.com/p/arduino/issues/detail?id=958>

```
* =====
==== */
```

```
// uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
// quaternion components in a [w, x, y, z] format (not best for parsing
// on a remote host such as Processing or something though)

#define OUTPUT_READABLE_QUATERNION
```

```
// uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
// (in degrees) calculated from the quaternions coming from the FIFO.

// Note that Euler angles suffer from gimbal lock (for more info, see
// http://en.wikipedia.org/wiki/Gimbal\_lock)
```

```
//#define OUTPUT_READABLE_EULER
```

```

// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
// more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
#define OUTPUT_READABLE_YAWPITCHROLL

// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
#define OUTPUT_READABLE_WORLDACCEL

// uncomment "OUTPUT_TEAPOT" if you want output that matches the
// format used for the InvenSense teapot demo
#define OUTPUT_TEAPOT

#define INTERRUPT_PIN 2 // use pin 2 on Arduino Uno & most boards

```

```

#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)

bool blinkState = false;

// MPU control/status vars

bool dmpReady = false; // set true if DMP init was successful

uint8_t mpuIntStatus; // holds actual interrupt status byte from MPU

uint8_t devStatus; // return status after each device operation (0
= success, !0 = error)

uint16_t packetSize; // expected DMP packet size (default is 42 bytes
)

uint16_t fifoCount; // count of all bytes currently in FIFO

uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars

Quaternion q; // [w, x, y, z] quaternion container

/*
VectorInt16 aa; // [x, y, z] accel sensor measurement
s

VectorInt16 aaReal; // [x, y, z] gravity-free accel senso
r measurements

VectorInt16 aaWorld; // [x, y, z] world-frame accel sensor
measurements

VectorFloat gravity; // [x, y, z] gravity vector

float euler[3]; // [psi, theta, phi] Euler angle container

float ypr[3]; // [yaw, pitch, roll] yaw/pitch/roll container
and gravity vector

// packet structure for InvenSense teapot demo

uint8_t teapotPacket[14] = { '$', 0x02, 0, 0, 0, 0, 0, 0, 0x00, 0x00,
'$', 'n' };

*/

```

```

#include <ros.h>
#include <ros/time.h>
#include <geometry_msgs/Quaternion.h>
ros::NodeHandle nh;
geometry_msgs::Quaternion quat_msg;
ros::Publisher quat_pub("quaternion", &quat_msg);

// =====
// ===          INTERRUPT DETECTION ROUTINE           ===
// =====

volatile bool mpuInterrupt = false;      // indicates whether MPU interrupt pin has gone high

void dmpDataReady() {
    mpuInterrupt = true;
}

// =====
// ===          INITIAL SETUP                         ===
// =====

void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    Wire.begin();

```

```

    Wire.setClock(400000); // 400kHz I2C clock. Comment this line if
    having compilation difficulties

    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE

        Fastwire::setup(400, true);

    #endif


    // initialize serial communication

    // (115200 chosen because it is required for Teapot Demo output, but
    it's

    // really up to you depending on your project)

    Serial.begin(115200);

    while (!Serial); // wait for Leonardo enumeration, others continue i
mmediately

    // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3V or A
rduino

    // Pro Mini running at 3.3V, cannot handle this baud rate reliably d
ue to

    // the baud timing being too misaligned with processor ticks. You mu
st use

    // 38400 or slower in these cases, or use some kind of external sepa
rate

    // crystal solution for the UART timer.

    // initialize device

    //Serial.println(F("Initializing I2C devices..."));

    mpu.initialize();

    pinMode(INTERRUPT_PIN, INPUT);

/*
    // verify connection

    Serial.println(F("Testing device connections..."));

```

```

    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

    // wait for ready

    Serial.println(F("\nSend any character to begin DMP programming and
demo: "));

    while (Serial.available() && Serial.read()); // empty buffer

    while (!Serial.available()); // wait for data

    while (Serial.available() && Serial.read()); // empty buffer again

    // load and configure the DMP

    Serial.println(F("Initializing DMP..."));

    */

    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(228);

    mpu.setYGyroOffset(93);

    mpu.setZGyroOffset(-64);

    mpu.setZAccelOffset(1421); // 1688 factory default for my test chip

    // make sure it worked (returns 0 if so)

    if (devStatus == 0) {

        // Calibration Time: generate offsets and calibrate our MPU6050
        mpu.CalibrateAccel(6);

        mpu.CalibrateGyro(6);

        mpu.PrintActiveOffsets();

        // turn on the DMP, now that it's ready
        //Serial.println(F("Enabling DMP..."));
    }
}

```

```

mpu.setDMPEnabled(true);

// enable Arduino interrupt detection

//Serial.print(F("Enabling interrupt detection (Arduino external
interrupt "));

//Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));

digitalPinToInterrupt(INTERRUPT_PIN);

//Serial.println(F(")..."));

attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataRea
dy, RISING);

mpuIntStatus = mpu.getIntStatus();

// set our DMP Ready flag so the main loop() function knows it's
okay to use it

//Serial.println(F("DMP ready! Waiting for first interrupt..."))
;

dmpReady = true;

// get expected DMP packet size for later comparison

packetSize = mpu.dmpGetFIFOPacketSize();

} else {

// ERROR!

// 1 = initial memory load failed

// 2 = DMP configuration updates failed

// (if it's going to break, usually the code will be 1)

Serial.print(F("DMP Initialization failed (code "));

Serial.print(devStatus);

Serial.println(F(")"));

}

```

```

// configure LED for output
pinMode(LED_PIN, OUTPUT);

nh.initNode();
nh.advertise(quat_pub);
}

// =====
// ===          MAIN PROGRAM LOOP          ===
// =====

void loop() {
    nh.spinOnce();

    // if programming failed, don't try to do anything
    if (!dmpReady) return;

    // read a packet from FIFO
    if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) { // Get the Latest packet

        #ifdef OUTPUT_READABLE_QUATERNION
            // display quaternion values in easy matrix form: w x y z
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            /*
            Serial.print("quat\t");
            Serial.print(q.w);

```

```

    Serial.print("t");
    Serial.print(q.x);
    Serial.print("t");
    Serial.print(q.y);
    Serial.print("t");
    Serial.println(q.z);

    */
    quat_msg.w = q.w;
    quat_msg.x = q.x;
    quat_msg.y = q.y;
    quat_msg.z = q.z;

    quat_pub.publish(&quat_msg);

#endif

#ifndef OUTPUT_READABLE_EULER
    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetEuler(euler, &q);
    Serial.print("eulert");
    Serial.print(euler[0] * 180/M_PI);
    Serial.print("t");
    Serial.print(euler[1] * 180/M_PI);
    Serial.print("t");
    Serial.print(euler[2] * 180/M_PI);
    Serial.println(euler[2] * 180/M_PI);

#endif

```

```

#define OUTPUT_READABLE_YAWPITCHROLL

    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);

    q.y = -q.y;
    q.z = -q.z;

    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
    Serial.print("ypr\t");
    Serial.print(ypr[0] * 180/M_PI);
    Serial.print("\t");
    Serial.print(ypr[1] * 180/M_PI);
    Serial.print("\t");
    Serial.println(ypr[2] * 180/M_PI);

#endif

#define OUTPUT_READABLE_REALACCEL

    // display real acceleration, adjusted to remove gravity
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    Serial.print("areal\t");
    Serial.print(aaReal.x);
    Serial.print("\t");
    Serial.print(aaReal.y);
    Serial.print("\t");
    Serial.println(aaReal.z);

#endif

```

```

#define OUTPUT_READABLE_WORLDACCEL

    // display initial world-frame acceleration, adjusted to remove gravity

    // and rotated based on known orientation from quaternion

    mpu.dmpGetQuaternion(&q, fifoBuffer);

    mpu.dmpGetAccel(&aa, fifoBuffer);

    mpu.dmpGetGravity(&gravity, &q);

    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);

    mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);

    Serial.print("aworld\t");

    Serial.print(aaWorld.x);

    Serial.print("\t");

    Serial.print(aaWorld.y);

    Serial.print("\t");

    Serial.print(aaWorld.z);

#endif

#define OUTPUT_TEAPOT

    // display quaternion values in InvenSense Teapot demo format:

    teapotPacket[2] = fifoBuffer[0];

    teapotPacket[3] = fifoBuffer[1];

    teapotPacket[4] = fifoBuffer[4];

    teapotPacket[5] = fifoBuffer[5];

    teapotPacket[6] = fifoBuffer[8];

    teapotPacket[7] = fifoBuffer[9];

    teapotPacket[8] = fifoBuffer[12];

```

```

teapotPacket[9] = fifoBuffer[13];

Serial.write(teapotPacket, 14);

teapotPacket[11]++; // packetCount, loops at 0xFF on purpose

#endif

// blink LED to indicate activity

blinkState = !blinkState;

digitalWrite(LED_PIN, blinkState);

}

}

```

### Lampiran 3 Pemodelan Robot dengan URDF

```

<?xml version="1.0"?>

<robot name="mobile_robot" xmlns:xacro="http://ros.org/wiki/xacro">

<material name="blue">
  <color rgba="0 0 0.8 1"/>
</material>

<material name="white">
  <color rgba="1 1 1 1"/>
</material>

```

```

<material name="black">
  <color rgba="0 0 0 1"/>
</material>

<material name="green">
  <color rgba="0.0 0.8 0.0 1.0"/>
</material>

<material name="orange">
  <color rgba="1 0.3 0.1 1"/>
</material>

<material name="red">
  <color rgba="1 0 0 1"/>
</material>

<link name="base_link">
  <!--<visual>
    <geometry>
      <box size="0.24 0.24 0.001"/>
    </geometry>
    <material name="white"/>
    <origin rpy="0 0 0" xyz="0 0 0.0005"/>
    <material>
      <color rgba="0 0 0.8 1" />
    </material>
  </visual> -->
  <collision>

```

```

<origin rpy="0 0 0"/>
<geometry>
  <box size="0.49 0.41 0.83"/>
</geometry>
</collision>
</link>

<joint name="base_link_joint" type="fixed">
  <origin xyz="-0.1 0 0"/>
  <parent link="base_link" />
  <child link="base_plate" />
</joint>

<link name="base_plate">
  <visual>
    <origin xyz="0.53 0.213 -0.1" rpy="0 0 ${pi}"/>
    <geometry>
      <mesh filename="package://autonomus_mobile_robot/stl/main_body.stl" scale="0.001 0.001 0.001" />
    </geometry>
    <material name="white"/>
  </visual>
  <collision>
    <origin xyz="0.53 0.213 -0.1" rpy="0 0 ${pi}"/>
    <geometry>
      <mesh filename="package://autonomus_mobile_robot/stl/main_body.stl" scale="0.001 0.001 0.001" />
    </geometry>
  </collision>
</link>
```

```

</link>

<!-- RIGHT WHEEL LINK -->

<joint name="right_wheel_joint" type="continuous">
  <origin xyz="0 -0.23 0" rpy="${pi/2} 0 0" />
  <parent link="base_plate" />
  <child link="right_wheel" />
  <axis xyz="0 0 -1"/>
</joint>

<link name="right_wheel">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://autonomus_mobile_robot/stl/wheel.stl"
        scale="0.001 0.001 0.001"/>
    </geometry>
    <material name="black"/>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://autonomus_mobile_robot/stl/wheel.stl"
        scale="0.001 0.001 0.001" />
    </geometry>
  </collision>
</link>
```

```

<!-- LEFT WHEEL LINK -->

<joint name="left_wheel_joint" type="continuous">
  <origin xyz="0 0.23 0" rpy="0 ${pi} ${pi/2}" />
  <parent link="base_plate" />
  <child link="left_wheel" />
  <axis xyz="-1 0 0" />
</joint>

<link name="left_wheel">
  <visual>
    <origin xyz="0 0 0" rpy="${pi} -${pi/2} 0"/>
    <geometry>
      <mesh filename="package://autonomus_mobile_robot/stl/wheel.stl"
        scale="0.001 0.001 0.001"/>
    </geometry>
    <material name="black"/>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="${pi} -${pi/2} 0"/>
    <geometry>
      <mesh filename="package://autonomus_mobile_robot/stl/wheel.stl"
        scale="0.001 0.001 0.001" />
    </geometry>
  </collision>
</link>

<!-- KINECT LINK -->

<joint name="laser_joint" type="fixed">
```

```

<origin xyz="-0 0 0.83" rpy="0 0 0"/>
<parent link="base_plate" />
<child link="kinect_link" />
</joint>

<link name="kinect_link">
<visual>
<origin xyz="0 0 0" rpy="0 0 ${pi}"/>
<geometry>
<mesh filename="package://autonomus_mobile_robot/stl/kinect.stl"
scale="0.001 0.001 0.001"/>
</geometry>
<material name="orange"/>
</visual>
<collision>
<origin xyz="0 0 0" rpy="0 0 ${pi}"/>
<geometry>
<mesh filename="package://autonomus_mobile_robot/stl/kinect.stl"
scale="0.001 0.001 0.001" />
</geometry>
</collision>
</link>

</robot>

```

#### Lampiran 4 Kode Pemrograman ROS

##### a. node imu quaternion converter

```
#include <ros/ros.h>
```

```
#include <geometry_msgs/Quaternion.h>
#include <geometry_msgs/PoseStamped.h>
#include <sensor_msgs/Imu.h>

ros::Publisher imuPub_;
ros::Publisher posePub_;

geometry_msgs::PoseStamped pose_msg;

sensor_msgs::Imu imu_msg;

void quaternionCb(const geometry_msgs::Quaternion::ConstPtr& msg) {

    bool pose_subscribed = (posePub_.getNumSubscribers() > 0);
    bool imu_subscribed = (imuPub_.getNumSubscribers() > 0);

    if (pose_subscribed || imu_subscribed) {

        //TODO make time offset param
        ros::Time sensor_data_time = ros::Time::now() - ros::Duration(0.002)
        ;

        if (imu_subscribed) {

            imu_msg.header.stamp = sensor_data_time;

            imu_msg.orientation = *msg;

            imuPub_.publish(imu_msg);
        }
    }
}
```

```
if (pose_subscribed) {  
    pose_msg.header.stamp = sensor_data_time;  
  
    pose_msg.pose.orientation = *msg;  
  
    posePub_.publish(pose_msg);  
}  
}  
  
}  
  
int main(int argc, char** argv) {  
  
    ros::init(argc, argv, "mpu6050_imu_converter");  
  
    ros::NodeHandle _nh;  
    ros::NodeHandle _pnh("~");  
  
    double linear_acceleration_stdev, angular_velocity_stdev;  
    std::string frame_id;  
    std::string frame_id2;  
  
    ros::param::param<double>("linear_acceleration_stdev", linear_acceleration_stdev, 0.06);  
    ros::param::param<double>("angular_velocity_stdev", angular_velocity_stdev, 0.005);  
    ros::param::param<std::string>("frame_id", frame_id, std::string("imu_link"));  
}
```

```

    double linear_acceleration_cov = linear_acceleration_stdev * linear_
acceleration_stdev;

    double angular_velocity_cov = angular_velocity_stdev * angular_veloc
ity_stdev;

    imu_msg.header.frame_id = frame_id;
pose_msg.header.frame_id = frame_id;

    imu_msg.orientation_covariance[0] = 0.1;
    imu_msg.orientation_covariance[4] = 0.1;
    imu_msg.orientation_covariance[8] = 0.1;

// Angular velocity entries are not filled, so set to -1.0
    imu_msg.angular_velocity_covariance[0] = 0.0; //angular_velocity_cov
;

    imu_msg.angular_velocity_covariance[4] = angular_velocity_cov;
    imu_msg.angular_velocity_covariance[8] = angular_velocity_cov;

// Linear acceleration entries are not filled, so set to -1.0
    imu_msg.linear_acceleration_covariance[0] = 0.0;
    imu_msg.linear_acceleration_covariance[4] = linear_acceleration_cov;
    imu_msg.linear_acceleration_covariance[8] = linear_acceleration_cov;

posePub_ = _pnh.advertise<geometry_msgs::PoseStamped>("pose", 10);
imuPub_ = _pnh.advertise<sensor_msgs::Imu>("imu", 10);

ros::Subscriber quatSub_ = _nh.subscribe("quaternion", 10, quaternionCb);

```

```
    ros::spin();
```

```
    return(0);
```

```
}
```

**b. node teleoperation joystick**

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

```
import rospy
```

```
from geometry_msgs.msg import Twist
```

```
from sensor_msgs.msg import Joy
```

```
def joy_callback(data):
```

```
    twist = Twist()
```

```
    twist.linear.x = data.axes[1] * 0.5 # Scale joystick input to half of maximum linear speed
```

```
    twist.angular.z = data.axes[0] * 0.5 # Scale joystick input to half of maximum angular speed
```

```
    pub.publish(twist)
```

```
rospy.init_node('joystick_controller')
```

```
pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
```

```
rospy.Subscriber('joy', Joy, joy_callback)
```

```
rospy.spin()
```

**c. node ekxtended Kalman filter odometry publisher**

```
// Include various libraries
```

```
#include "ros/ros.h"
```

```
#include "std_msgs/Int16.h"
#include <nav_msgs/Odometry.h>
#include <geometry_msgs/PoseStamped.h>
#include <tf2/LinearMath/Quaternion.h>
#include <tf2_ros/transform_broadcaster.h>
#include <cmath>

// Create odometry data publishers
ros::Publisher odom_data_pub;
ros::Publisher odom_data_pub_quat;
nav_msgs::Odometry odomNew;
nav_msgs::Odometry odomOld;

// Initial pose
const double initialX = 0.0;
const double initialY = 0.0;
const double initialTheta = 0.0000000001;
const double PI = 3.141592;

// Robot physical constants
const double TICKS_PER_REVOLUTION = 36; // For reference purposes.
const double WHEEL_RADIUS = 0.15; // Wheel radius in meters
const double WHEEL_BASE = 0.46; // Center of left tire to center of right tire
const double TICKS_PER_METER = 76.433121; // Original was 2800

// Distance both wheels have traveled
double distanceLeft = 0;
```

```

double distanceRight = 0;

// Flag to see if initial pose has been received
bool initialPoseRecieved = false;

using namespace std;

// Get initial_2d message from either Rviz clicks or a manual pose publisher

void set_initial_2d(const geometry_msgs::PoseStamped &rvizClick) {

    odomOld.pose.position.x = rvizClick.pose.position.x;
    odomOld.pose.position.y = rvizClick.pose.position.y;
    odomOld.pose.orientation.z = rvizClick.pose.orientation.z;
    initialPoseRecieved = true;
}

// Calculate the distance the left wheel has traveled since the last cycle

void Calc_Left(const std_msgs::Int16& leftCount) {

    static int lastCountL = 0;
    if(leftCount.data != 0 && lastCountL != 0) {

        int leftTicks = (leftCount.data - lastCountL);

        if (leftTicks > 10000) {
            leftTicks = 0 - (65535 - leftTicks);
    }
}

```

```

    }

    else if (leftTicks < -10000) {

        leftTicks = 65535-leftTicks;

    }

    else {}

    distanceLeft = leftTicks/TICKS_PER_METER;

}

lastCountL = leftCount.data;

}

// Calculate the distance the right wheel has traveled since the last cycle

void Calc_Right(const std_msgs::Int16& rightCount) {

    static int lastCountR = 0;

    if(rightCount.data != 0 && lastCountR != 0) {

        int rightTicks = rightCount.data - lastCountR;

        if (rightTicks > 10000) {

            distanceRight = (0 - (65535 - distanceRight))/TICKS_PER_METER;

        }

        else if (rightTicks < -10000) {

            rightTicks = 65535 - rightTicks;

        }

        else {}

        distanceRight = rightTicks/TICKS_PER_METER;

    }

}

```

```
lastCountR = rightCount.data;  
}  
  
// Publish a nav_msgs::Odometry message in quaternion format  
void publish_quat() {  
  
    tf2::Quaternion q;  
  
    q.setRPY(0, 0, odomNew.pose.pose.orientation.z);  
  
    nav_msgs::Odometry quat0dom;  
    quat0dom.header.stamp = odomNew.header.stamp;  
    quat0dom.header.frame_id = "odom";  
    quat0dom.child_frame_id = "base_link";  
    quat0dom.pose.pose.position.x = odomNew.pose.pose.position.x;  
    quat0dom.pose.pose.position.y = odomNew.pose.pose.position.y;  
    quat0dom.pose.pose.position.z = odomNew.pose.pose.position.z;  
    quat0dom.pose.pose.orientation.x = q.x();  
    quat0dom.pose.pose.orientation.y = q.y();  
    quat0dom.pose.pose.orientation.z = q.z();  
    quat0dom.pose.pose.orientation.w = q.w();  
    quat0dom.twist.twist.linear.x = odomNew.twist.twist.linear.x;  
    quat0dom.twist.twist.linear.y = odomNew.twist.twist.linear.y;  
    quat0dom.twist.twist.linear.z = odomNew.twist.twist.linear.z;  
    quat0dom.twist.twist.angular.x = odomNew.twist.twist.angular.x;  
    quat0dom.twist.twist.angular.y = odomNew.twist.twist.angular.y;  
    quat0dom.twist.twist.angular.z = odomNew.twist.twist.angular.z;
```

```

for(int i = 0; i<36; i++) {

    if(i == 0 || i == 7 || i == 14) {

        quat0dom.pose.covariance[i] = .01;

    }

    else if (i == 21 || i == 28 || i== 35) {

        quat0dom.pose.covariance[i] += 0.1;

    }

    else {

        quat0dom.pose.covariance[i] = 0;

    }

}

odom_data_pub_quat.publish(quat0dom);

}

// Update odometry information

void update_odom() {

    // Calculate the average distance

    double cycleDistance = (distanceRight + distanceLeft) / 2;

    // Calculate the number of radians the robot has turned since the last

    // cycle

    double cycleAngle = asin((distanceRight-distanceLeft)/WHEEL_BASE);

    // Average angle during the last cycle

    double avgAngle = cycleAngle/2 + odomOld.pose.orientation.z;
}

```

```

if (avgAngle > PI) {
    avgAngle -= 2*PI;
}

else if (avgAngle < -PI) {
    avgAngle += 2*PI;
}

else {}

// Calculate the new pose (x, y, and theta)

odomNew.pose.pose.position.x = odomOld.pose.pose.position.x + cos(avgAngle)*cycleDistance;

odomNew.pose.pose.position.y = odomOld.pose.pose.position.y + sin(avgAngle)*cycleDistance;

odomNew.pose.pose.orientation.z = cycleAngle + odomOld.pose.pose.orientation.z;

// Prevent lockup from a single bad cycle

if (isnan(odomNew.pose.pose.position.x) || isnan(odomNew.pose.pose.position.y)
    || isnan(odomNew.pose.pose.position.z)) {
    odomNew.pose.pose.position.x = odomOld.pose.pose.position.x;
    odomNew.pose.pose.position.y = odomOld.pose.pose.position.y;
    odomNew.pose.pose.orientation.z = odomOld.pose.pose.orientation.z;
}

// Make sure theta stays in the correct range

if (odomNew.pose.pose.orientation.z > PI) {
    odomNew.pose.pose.orientation.z -= 2 * PI;
}

```

```

else if (odomNew.pose.pose.orientation.z < -PI) {

    odomNew.pose.pose.orientation.z += 2 * PI;

}

else {}

// Compute the velocity

odomNew.header.stamp = ros::Time::now();

odomNew.twist.twist.linear.x = cycleDistance/(odomNew.header.stamp.toSec() - odomOld.header.stamp.toSec());

odomNew.twist.twist.angular.z = cycleAngle/(odomNew.header.stamp.toSec() - odomOld.header.stamp.toSec());



// Save the pose data for the next cycle

odomOld.pose.pose.position.x = odomNew.pose.pose.position.x;

odomOld.pose.pose.position.y = odomNew.pose.pose.position.y;

odomOld.pose.pose.orientation.z = odomNew.pose.pose.orientation.z;

odomOld.header.stamp = odomNew.header.stamp;




// Publish the odometry message

odom_data_pub.publish(odomNew);

}

int main(int argc, char **argv) {

// Set the data fields of the odometry message

odomNew.header.frame_id = "odom";

odomNew.pose.pose.position.z = 0;

odomNew.pose.pose.orientation.x = 0;

```

```

odomNew.pose.pose.orientation.y = 0;

odomNew.twist.twist.linear.x = 0;

odomNew.twist.twist.linear.y = 0;

odomNew.twist.twist.linear.z = 0;

odomNew.twist.twist.angular.x = 0;

odomNew.twist.twist.angular.y = 0;

odomNew.twist.twist.angular.z = 0;

odomOld.pose.pose.position.x = initialX;

odomOld.pose.pose.position.y = initialY;

odomOld.pose.pose.orientation.z = initialTheta;

// Launch ROS and create a node

ros::init(argc, argv, "ekf_odom_pub");

ros::NodeHandle node;

// Subscribe to ROS topics

ros::Subscriber subForRightCounts = node.subscribe("right_ticks", 100,
Calc_Right, ros::TransportHints().tcpNoDelay());

ros::Subscriber subForLeftCounts = node.subscribe("left_ticks", 100, C
alc_Left, ros::TransportHints().tcpNoDelay());

ros::Subscriber subInitialPose = node.subscribe("initial_2d", 1, set_i
nitial_2d);

// Publisher of simple odom message where orientation.z is an euler an
gle

odom_data_pub = node.advertise<nav_msgs::Odometry>("odom_data_euler",
100);

// Publisher of full odom message where orientation is quaternion

```

```
odom_data_pub_quat = node.advertise<nav_msgs::Odometry>("odom_data_quat", 100);

ros::Rate loop_rate(30);

while(ros::ok()) {

    if(initialPoseRecieved) {

        update_odom();

        publish_quat();

    }

    ros::spinOnce();

    loop_rate.sleep();

}

return 0;
}
```

### L:ampiran 5 Launch File SLAM

```
<?xml version="1.0"?>
<launch>
    <!-- kinect -->
    <include file="$(find freenect_launch)/launch/freenect.launch">
        <arg name="depth_registration" value="true" />
    </include>

    <!-- Kinect cloud to laser scan -->
    <node pkg="depthimage_to_laserscan" type="depthimage_to_laserscan" name="depthimage_to_laserscan">
        <remap from="image" to="/camera/depth_registered/image_raw"/>
        <remap from="camera_info" to="/camera/depth_registered/camera_info"/>
        <remap from="scan" to="/kinect_scan"/>
        <param name="range_max" type="double" value="4"/>
    </node>

    <!-- another-config -->
    <arg name="model" default="$(find autonomus_mobile_robot)/urdf/mobile_robot.urdf.xacro"/>
    <param name="robot_description" command="$(find xacro)/xacro $(arg model)" />
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" >
    </node>
```

```

<node name="joint_state_publisher_gui" pkg="joint_state_publisher_gui"
type="joint_state_publisher_gui" />

<node name="rviz" pkg="rviz" type="rviz" args="-d $(find autonomus_mobile_robot)/config/amcl_nav.rviz"/>

<node pkg="tf" type="static_transform_publisher" name="image_to_scan_broadcaster" args="0 0 0 0 0 0 ¥/camera_depth_frame camera 100" />

<arg name="pi/2" value="1.5707963267948966" />
<arg name="pi" value="3.14" />

<node pkg="tf" type="static_transform_publisher" name="camera_optical" args="-0 0 0.83 0 0 0) /base_link /camera_link 100" />

<!-- Odometry -->

<node pkg="nodelet" type="nodelet" name="rgbd_sync" args="standalone rtabmap_sync/rgbd_sync" output="screen">
    <remap from="rgb/image" to="/camera/rgb/image_rect_color"/>
    <remap from="depth/image" to="/camera/depth_registered/image_raw"/>
    <remap from="rgb/camera_info" to="/camera/rgb/camera_info"/>
    <remap from="rgbd_image" to="rgbd_image"/> <!-- output -->

<!-- Should be true for not synchronized camera topics
(e.g., false for kinectv2, zed, realsense, true for xtion, kinect360)-->

<param name="approx_sync" value="true"/>
</node>

<node pkg="rtabmap_odom" type="rgbd_odometry" name="rgbd_odometry" output="screen">
    <param name="subscribe_rgbd" type="bool" value="true"/>
    <param name="frame_id" type="string" value="base_link"/> <!-- ubah jadi "base_link" jika tidak menggunakan fusion -->
    <remap from="rgbd_image" to="rgbd_image"/>

```

```

<remap from="odom" to="vo"/> <!-- Mengubah topik odom menjadi "odom_
rgbd" jika mengaktifkan fusion -->
</node>

<node pkg="my_robot_pkg" type="ekf_odom_pub" name="ekf_odom_pub">
</node>

<node pkg="my_robot_pkg" type="rviz_click_to_2d" name="rviz_click_to_2
d">
</node>

<node pkg="rosserial_python" type="serial_node.py" name="serial_node">
<param name="port" value="/dev/ttyUSB0"/>
<param name="baud" value="115200"/>
</node>

<node pkg="tf" type="static_transform_publisher" name="imu_broadcaster"
" args="0 0.06 0.02 0 0 0 base_link imu_link 30" />
<node pkg="tf" type="static_transform_publisher" name="base_link_broad
caster" args="0 0 0.09 0 0 0 base_footprint base_link 30" />
<remap from="odom" to="odom_data_quat" />
<remap from="imu_data" to="imu_data" />
<node pkg="robot_pose_ekf" type="robot_pose_ekf" name="robot_pose_ekf"
>
<param name="output_frame" value="odom"/>
<param name="base_footprint_frame" value="base_footprint"/>
<param name="freq" value="30.0"/>
<param name="sensor_timeout" value="1.0"/>
<param name="odom_used" value="true"/>
<param name="imu_used" value="false"/>
<param name="vo_used" value="true"/>
<param name="gps_used" value="false"/>
<param name="debug" value="false"/>
<param name="self_diagnose" value="false"/>

```

```

</node>

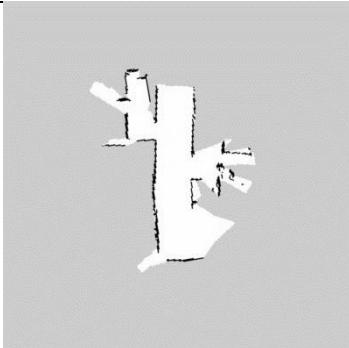
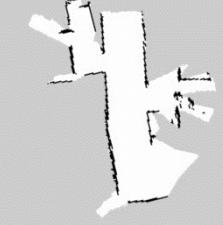
<!-- SLAM -->

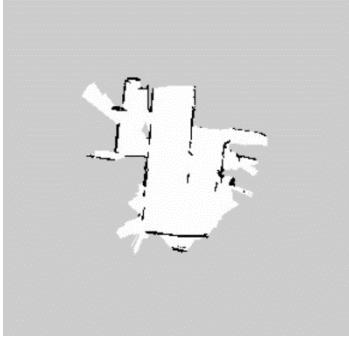
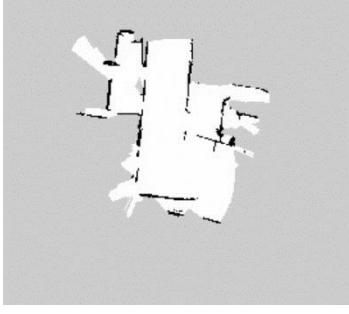
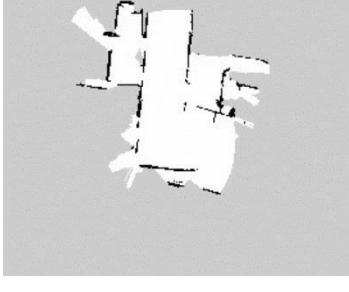
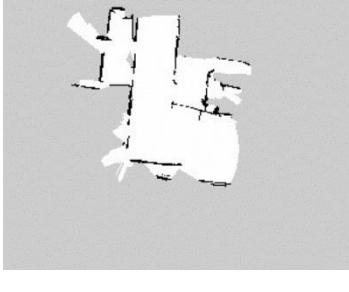
<node pkg="gmapping" type="slam_gmapping" name="gmapping_thing" output="screen" >
    <remap from="scan" to="/kinect_scan" />
    <param name="odom_frame" value="/odom" />
    <param name="base_frame" value="/base_link" />
</node>

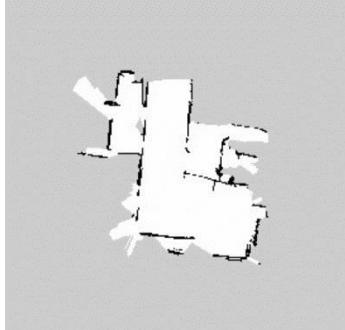
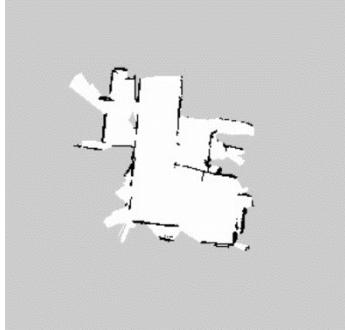
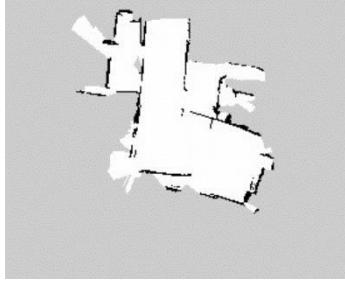
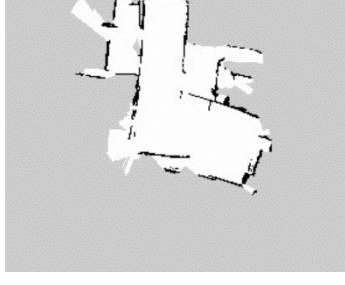
</launch>

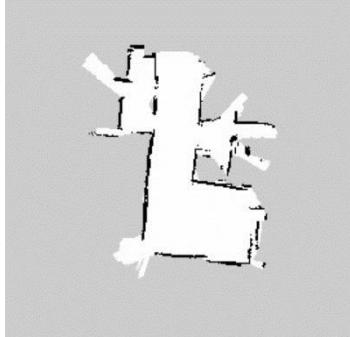
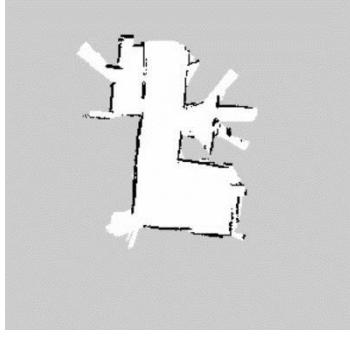
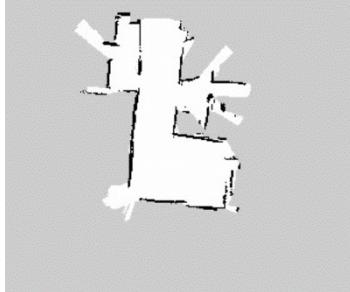
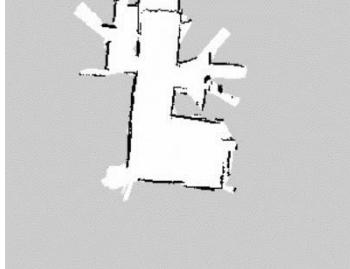
```

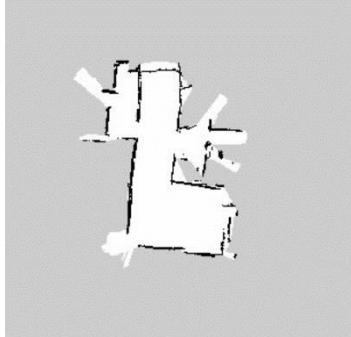
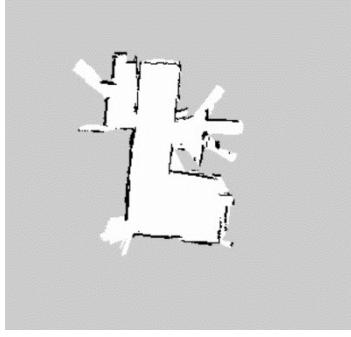
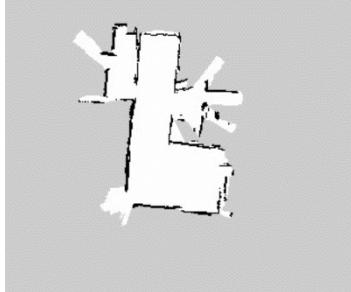
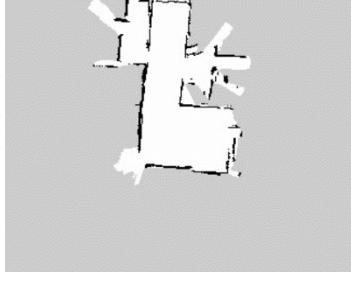
**Lampiran 6 Data Hasil Pemetaan Ruang Uji 1 (Laboratorium Sistem K  
endali dan instrumentasi)**

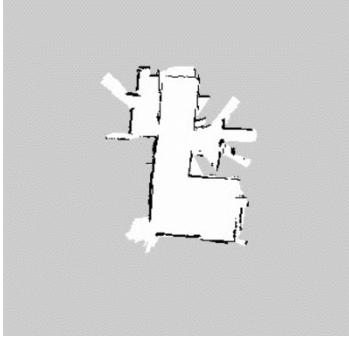
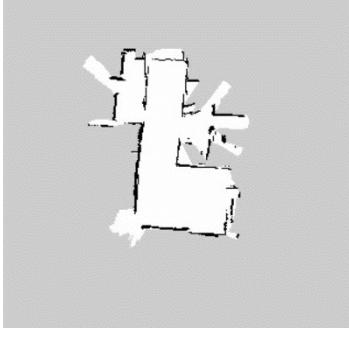
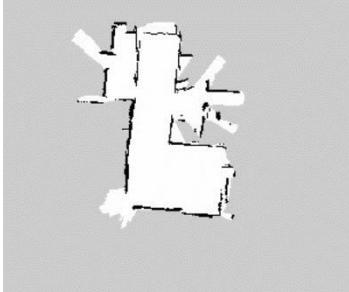
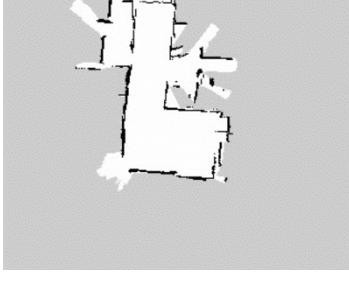
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
1			7.05%
2			7.94%

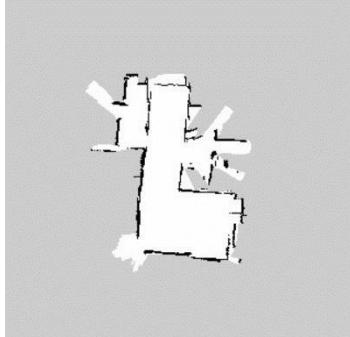
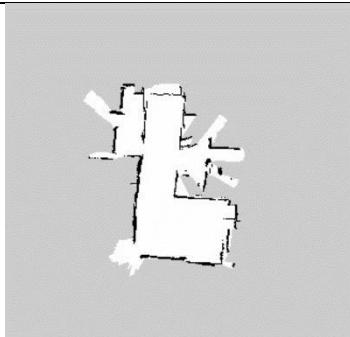
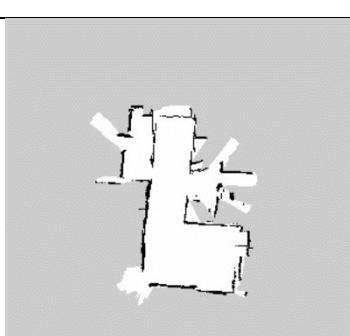
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
3			9.18%
4			9.26%
5			9.26%
6			9.46%

Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
7			10.57%
8			9.83%
9			11.00%
10			11.17%

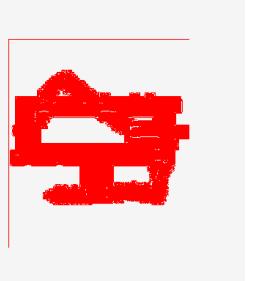
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
11			12. 12%
12			10. 87%
13			10. 19%
14			10. 11%

Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
15			9.90%
16			9.68%
17			9.67%
18			9.05%

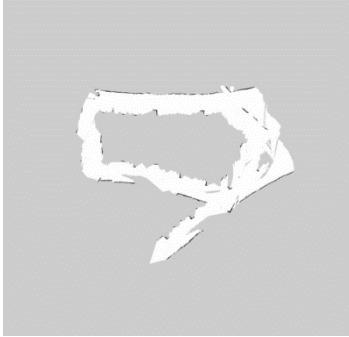
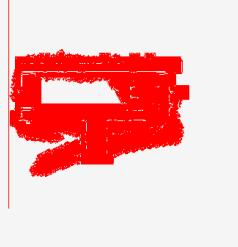
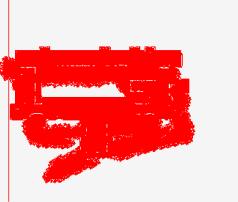
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
19			8.01%
20			9.20%
21			9.96%
22			10.37%

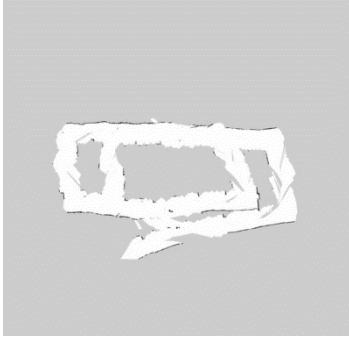
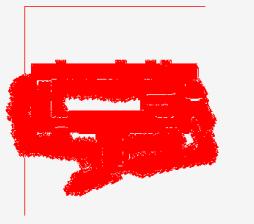
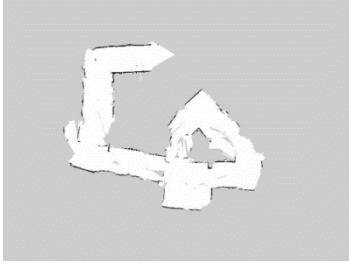
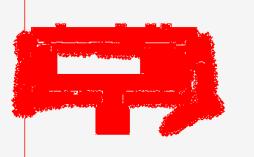
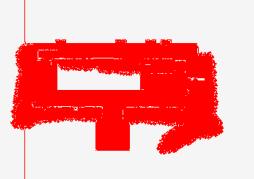
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
23			9. 06%
24			8. 79%
25			9. 26%
Rata-rata			9, 24%

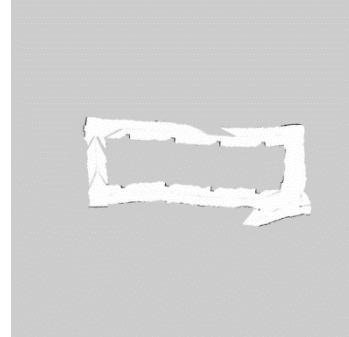
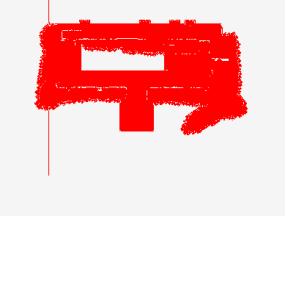
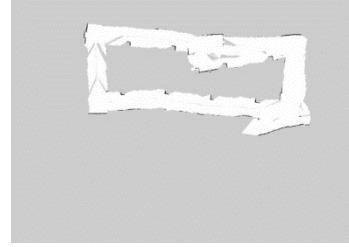
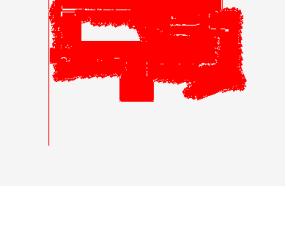
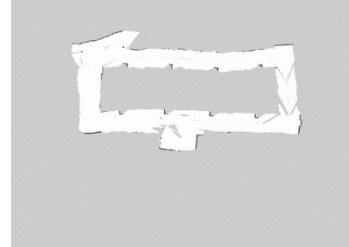
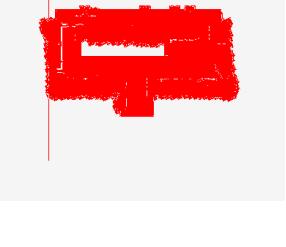
Lampiran 7 Data Hasil Pemetaan Ruang Uji 2 (Gedung Center Of Technology Lantai 2)

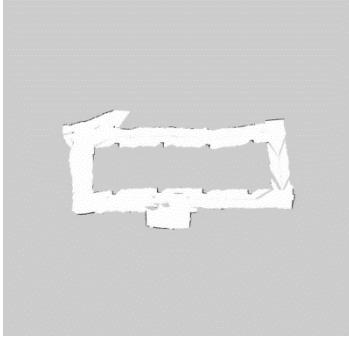
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
1			17.02%
2			18.76%

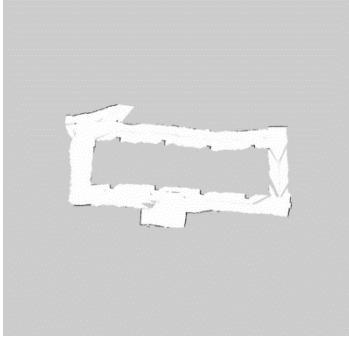
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
3			19. 34%
4			17. 64%
5			19. 63%
6			19. 63%

Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
7			18. 32%
8			17. 56%
9			20. 62%
10			22. 40%

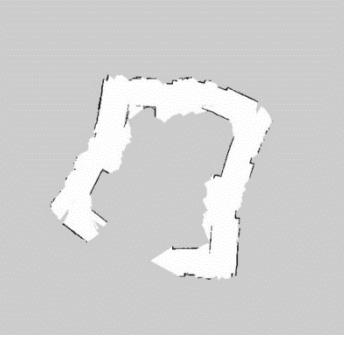
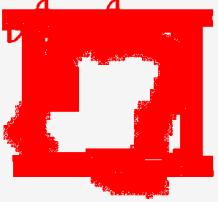
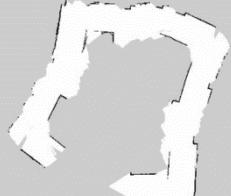
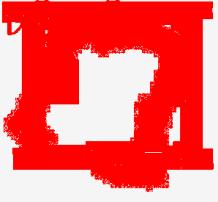
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
11			23. 43%
12			22. 72%
13			19. 58%
14			19. 38%

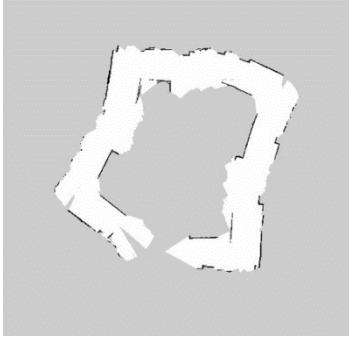
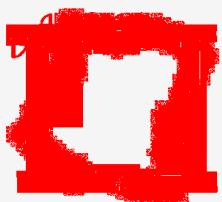
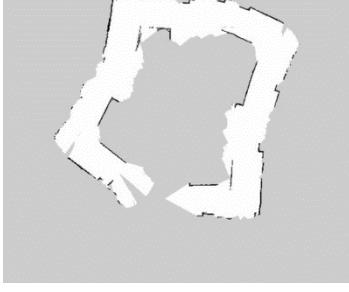
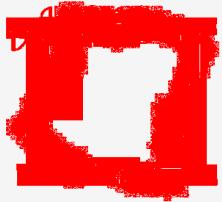
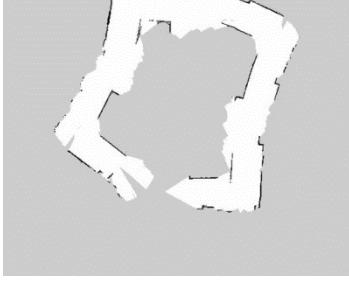
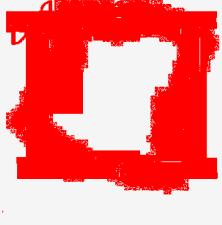
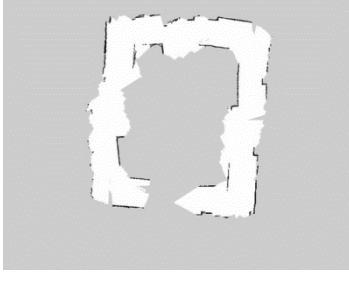
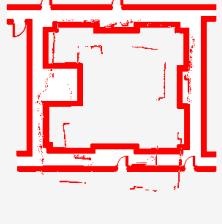
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
15			19. 38%
16			18. 42%
17			19. 64%
18			17. 55%

Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
19			15. 51%
20			15. 51%
21			15. 32%
22			15. 14%

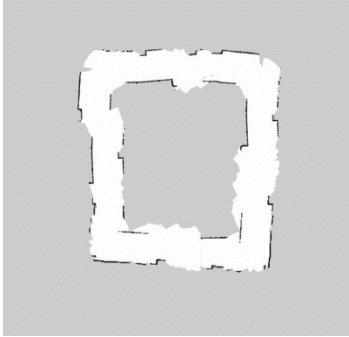
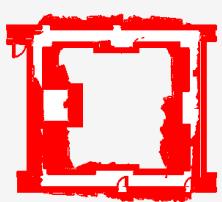
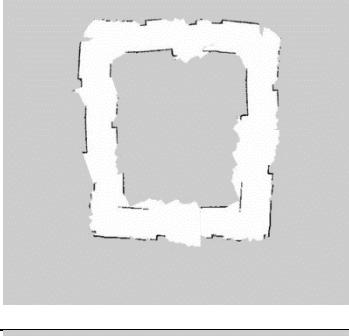
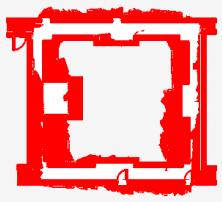
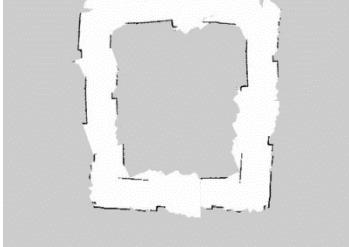
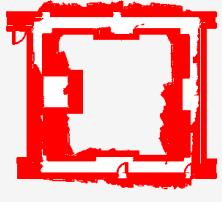
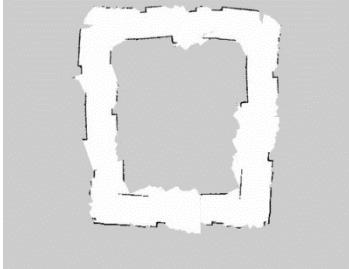
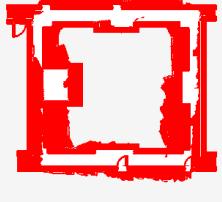
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
23			15. 32%
24			15. 14%
25			15. 11%
Rata-rata			18, 32%

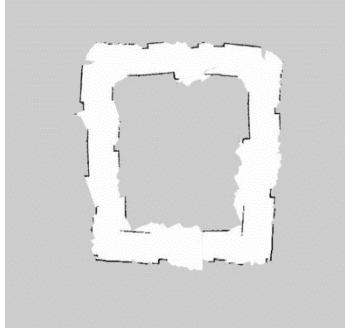
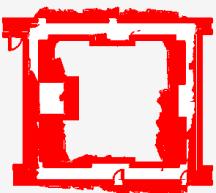
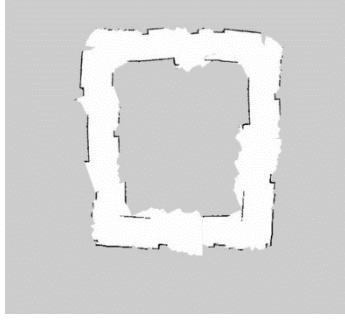
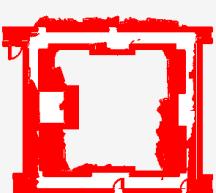
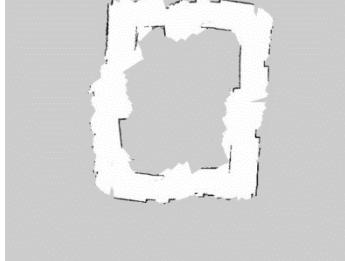
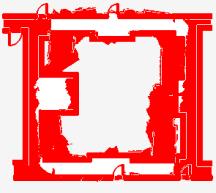
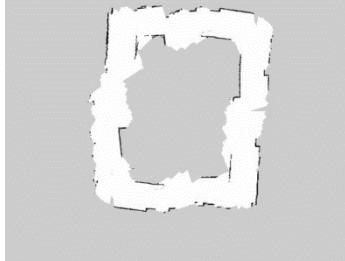
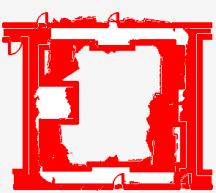
Lampiran 8 Data Hasil Pemetaan Ruang Uji 3 (Gedung Elektro Lantai 3)

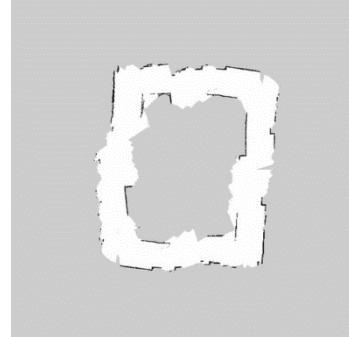
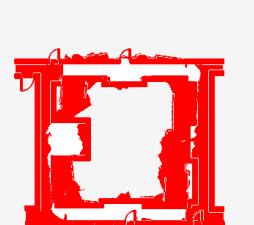
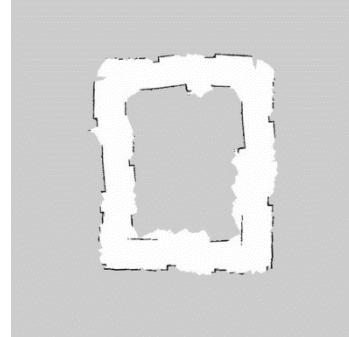
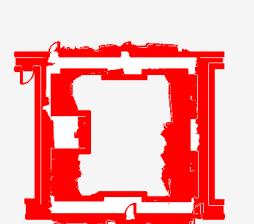
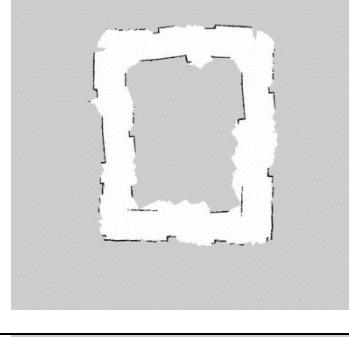
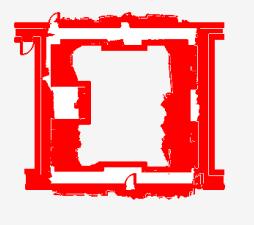
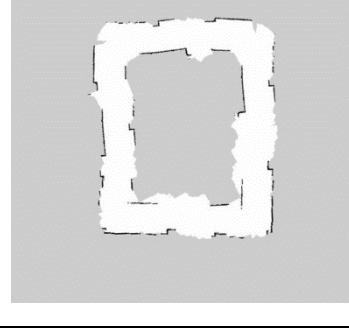
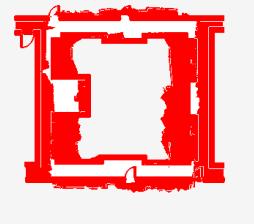
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
1			30.35%
2			30.54%

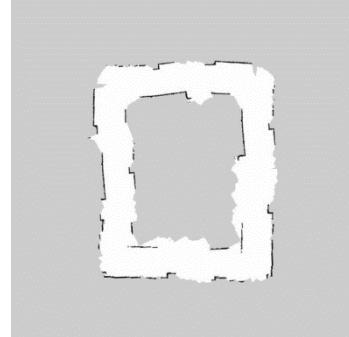
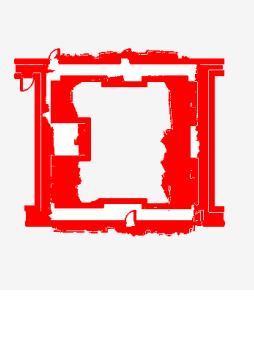
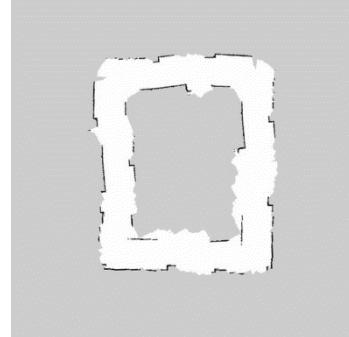
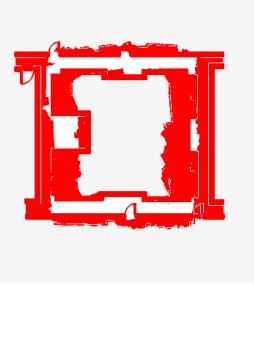
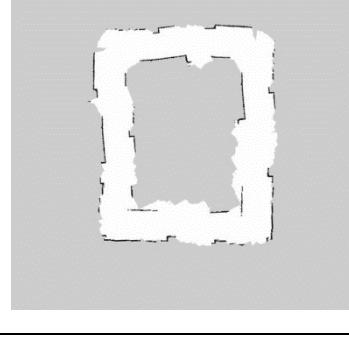
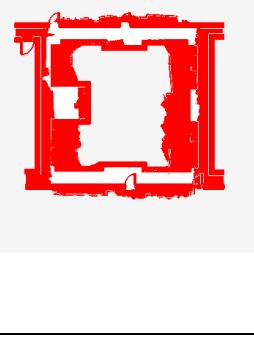
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
3			29. 87%
4			29. 66%
5			29. 39%
6			19. 41%

Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
7			19.41%
8			19.41%
9			19.41%
10			19.23%

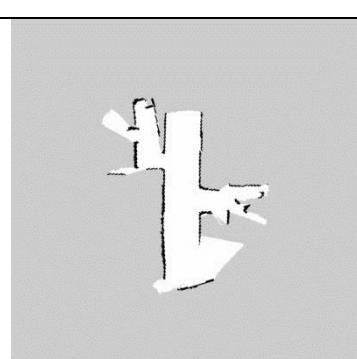
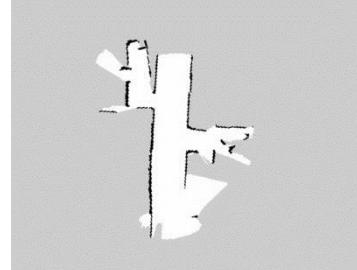
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
11			19.23%
12			19.23%
13			19.23%
14			19.23%

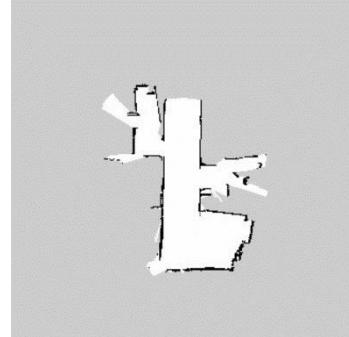
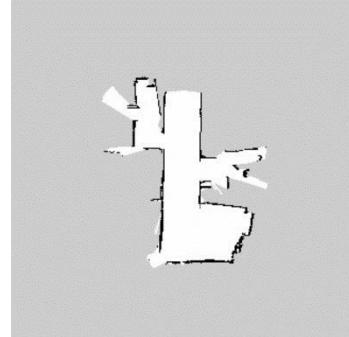
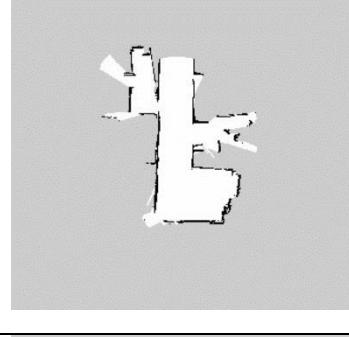
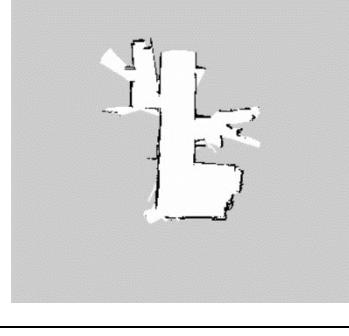
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
15			19.23%
16			19.23%
17			20.10%
18			20.10%

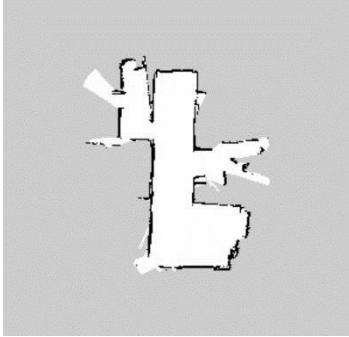
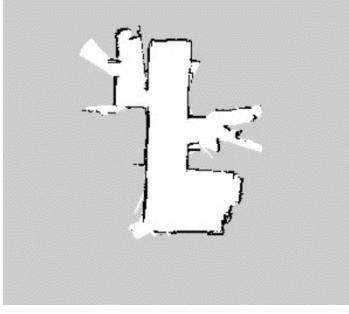
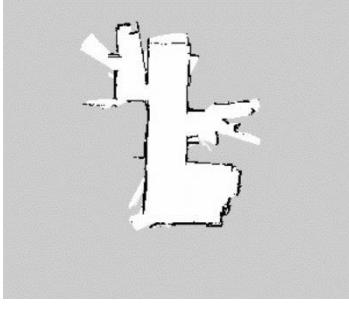
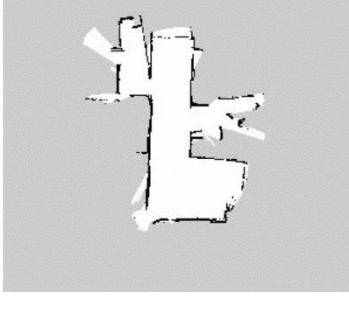
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
19			20. 10%
20			20. 64%
21			20. 64%
22			20. 64%

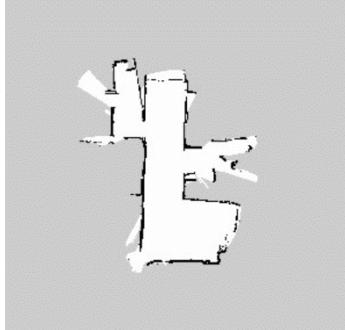
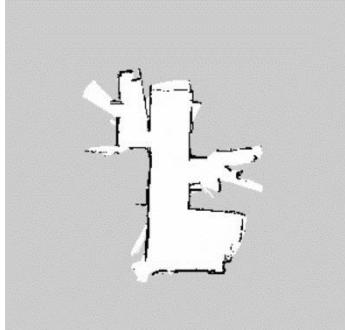
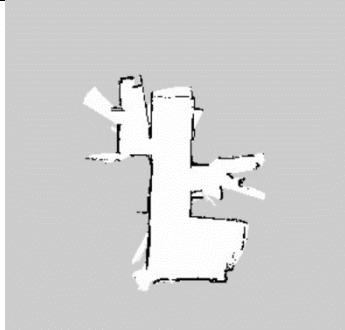
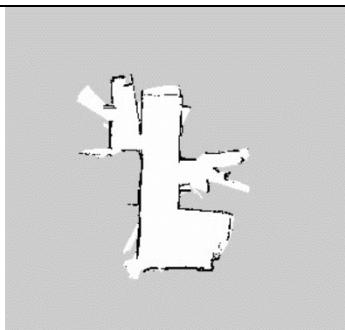
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
23			20. 64%
24			20. 43%
25			20. 43%
Rata-rata			21, 83%

Lampiran 9 Data Hasil Pemetaan pada kondisi luminensi skenario 1

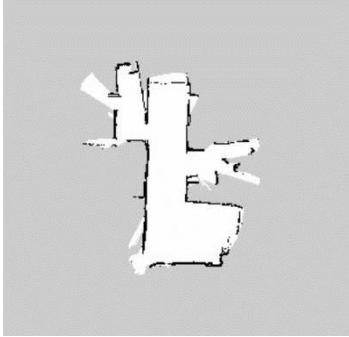
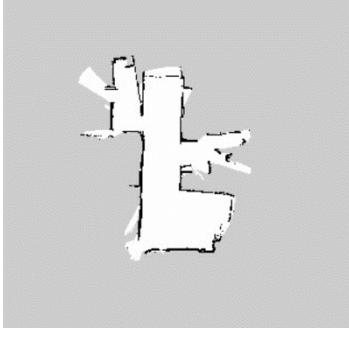
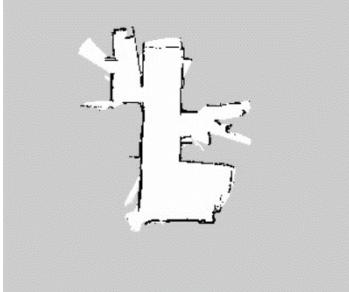
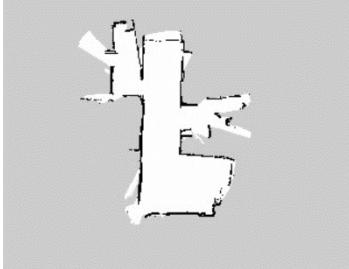
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
1			5. 17%
2			4. 90%

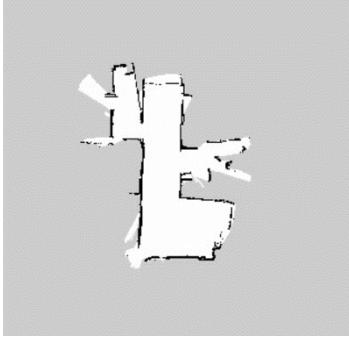
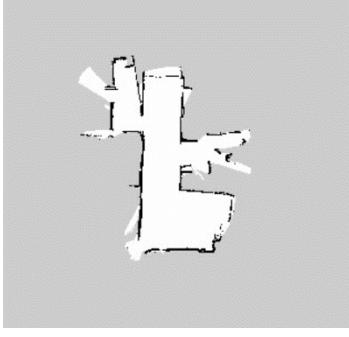
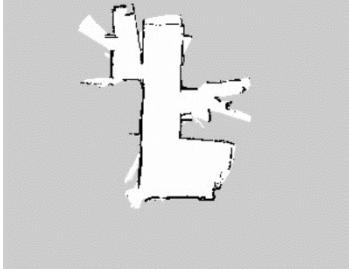
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
3			5. 54%
4			5. 54%
5			5. 08%
6			5. 11%

Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
7			5.71%
8			5.75%
9			5.61%
10			5.36%

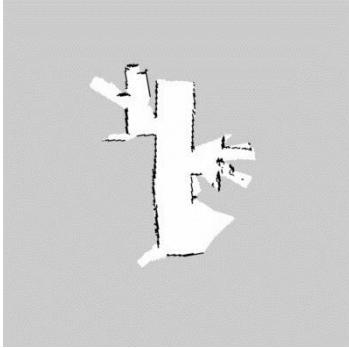
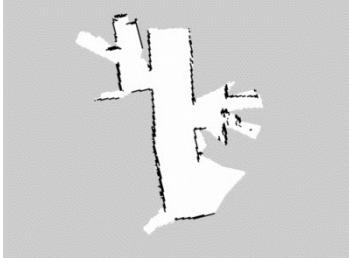
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
11			5.75%
12			5.57%
13			5.49%
14			5.62%

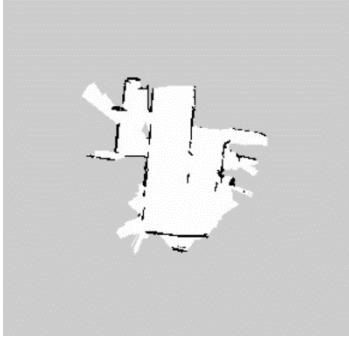
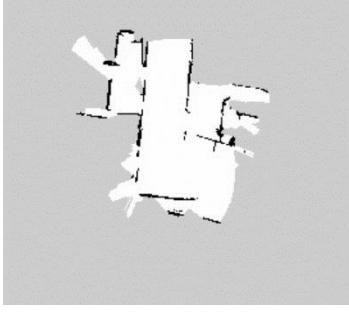
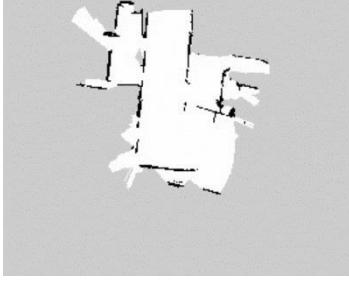
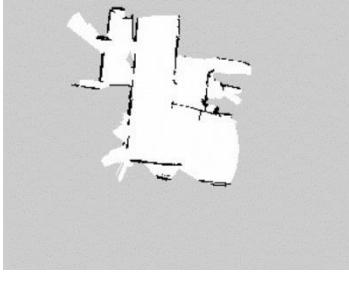
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
15			5.71%
16			5.79%
17			5.79%
18			5.34%

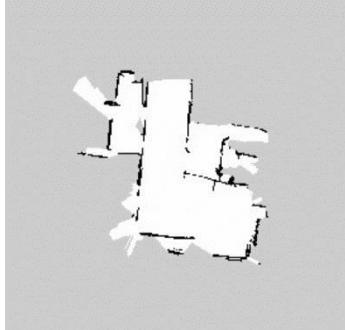
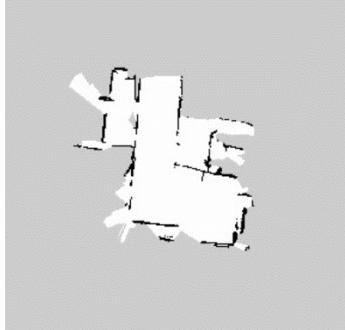
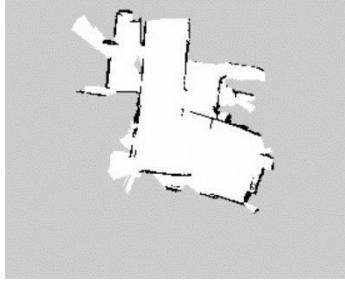
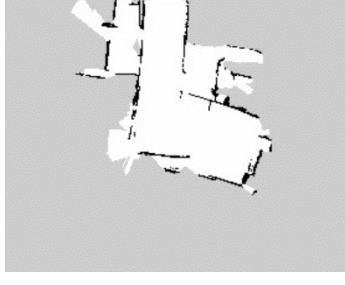
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
19			5. 58%
20			5. 72%
21			5. 72%
22			5. 72%

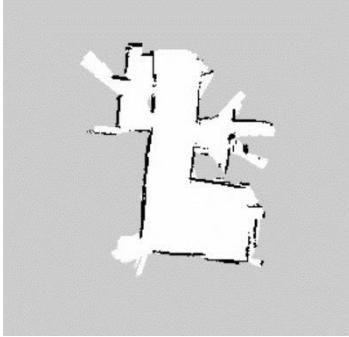
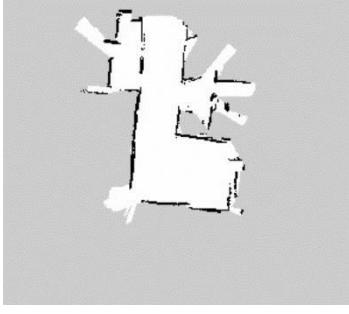
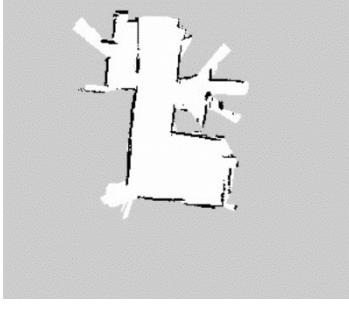
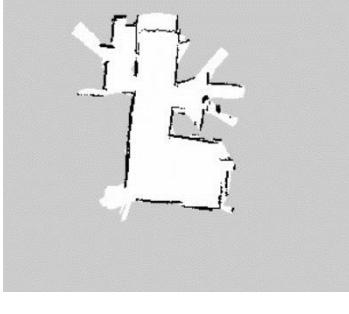
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
23			5.72%
24			5.72%
25			5.72%
Rata-rata			5,55%

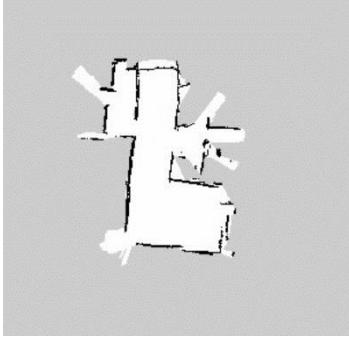
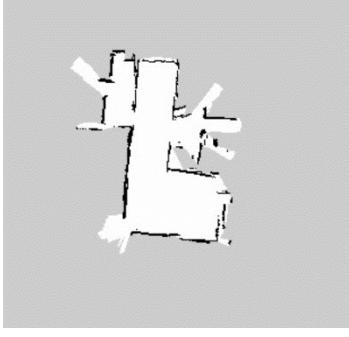
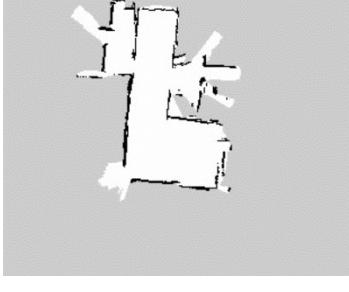
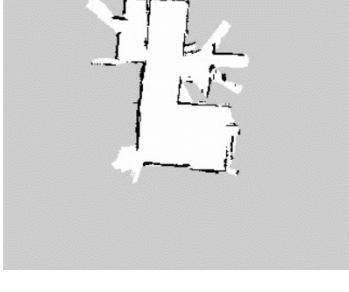
Lampiran 10 Data Hasil Pemetaan pada kondisi luminensi skenario 2

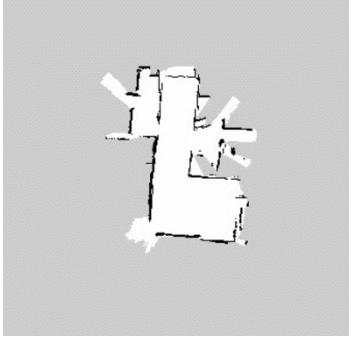
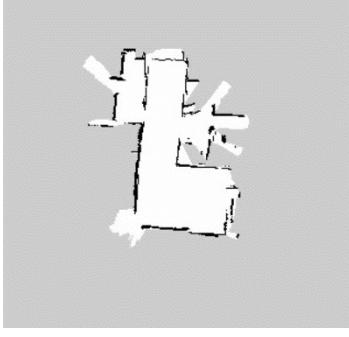
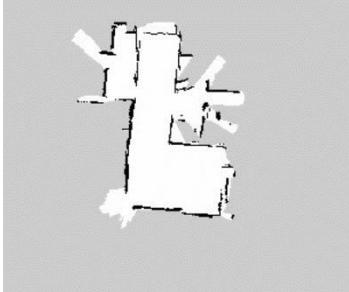
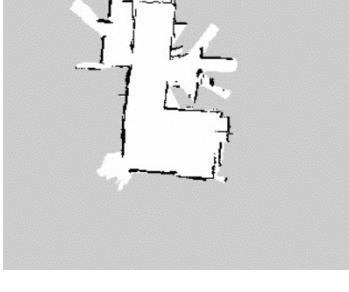
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
1			7. 05%
2			7. 94%

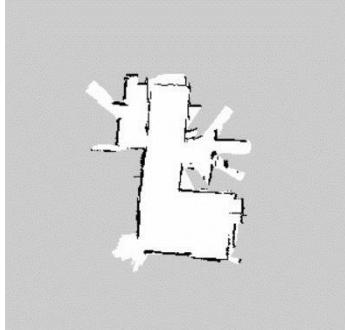
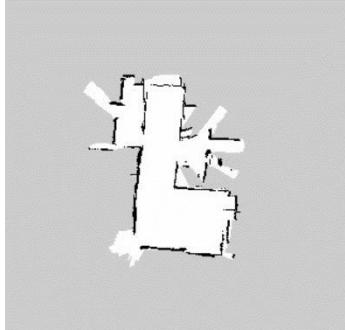
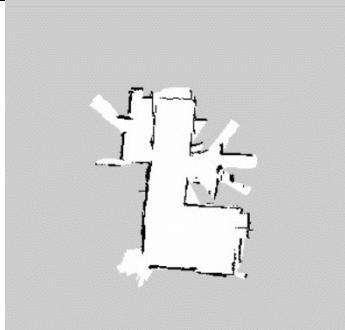
Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
3			9. 18%
4			9. 26%
5			9. 26%
6			9. 46%

Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
7			10.57%
8			9.83%
9			11.00%
10			11.17%

Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
11			12. 12%
12			10. 87%
13			10. 19%
14			10. 11%

Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
15			9.90%
16			9.68%
17			9.67%
18			9.05%

Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
19			8.01%
20			9.20%
21			9.96%
22			10.37%

Percobaan ke-n	Hasil pemetaan	Komparasi	% Error (RMSE)
23			9. 06%
24			8. 79%
25			9. 26%
Rata-rata			9, 24%