ABSTRAK

Bambang Gunawan Tanjung, Implementasi Pemetaan dan Navigasi Dalam Ruangan Pada *Autonomous Mobile Robot* Menggunakan Algoritme Hector SLAM Dan Navfn Dengan Sensor RPLIDAR.

Pembimbing: Rizal Maulana, S.T., M.T., M.Sc. dan Rakhmadhany Primananda, S.T., M. Kom.

Setiap tahunnya jumlah permintaan pengiriman barang selalu meningkat. Peningkatan ini harus disertai dengan sistem manajemen gudang yang baik karena jika tidak, maka dapat mengakibatkan kendala pada waktu dan distribusi pengiriman. Penggunaan *autonomous mobile robot* dirasa mampu membuat proses manajemen persediaan gudang menjadi lebih mudah dan efisien. Akan tetapi solusi ini memiliki permasalahan pada sistem navigasinya. Hal ini dikarenakan sistem navigasi GPS tidak bisa diimplementasikan di dalam gudang. Maka dari itu, penelitian ini menyajikan solusi utama dari permasalahan tersebut dengan mengimplementasikan sistem navigasi otonom yang berfokus pada lingkungan gudang di dalam ruangan.

Sistem ini terdiri dari robot yang mampu melakukan pemetaan lingkungan sekitarnya serta melakukan navigasi secara otonom dengan sensor *encoder*, RPLIDAR, dan IMU. Arsitektur perangkat lunak dari sistem ini dibangun pada platform *Robot Operating System* (ROS). ROS dipilih karena memiliki banyak *library* yang mampu menunjang lokalisasi, pemetaan, dan navigasi. Penelitian ini menerapkan berbagai algoritme seperti, Hector SLAM yang dikombinasikan dengan lokalisasi *Extended Kalman Filter* (EKF) untuk memvalidasi posisi robot. Algoritme Navfn yang digunakan untuk perencanaan jalur global dan algoritme *Dynamic Windows Approach* (DWA).

Dari hasil eksperimen, ditemukan bahwa algoritme Hector SLAM kurang sesuai digunakan pada ruangan yang sangat luas seperti gudang. Ini disebabkan karena kurangnya jangkauan laser pada dinding di sekitarnya sehingga terjadi kesalahan pada proses *particle filter*. Hal ini dibuktikan dengan hasil peta yang buruk pada saat pengujian yang dimulai dari posisi tengah gudang saat jauh dari dinding, sedangkan hasil yang sangat baik pada posisi pojok saat terdekat dengan dinding. Sedangkan untuk navigasi ditemukan bahwa robot bermanuver dengan sangat baik dan algoritme Navfn serta DWA yang digunakan sudah sangat tepat. Hal ini dibuktikan dari total 80% keberhasilan saat pengujian dengan halangan statis dan total 80% keberhasilan saat pengujian dengan halangan dinamis.

Kata kunci: *autonomous mobile robot*, *robot operating system*, *indoor*, rplidar, lokalisasi, pemetaan, navigasi

ABSTRACT

Bambang Gunawan Tanjung, Implementation of Localization, Mapping and Indoor Navigation on Autonomous Mobile Robots Using Rplidar Sensor, SLAM Hector Algorithm, And Robot Operating System-Based Navfn Algorithm.

Supervisor: **Rizal Maulana, S.T., M.T., M.Sc. dan Rakhmadhany Primananda, S.T., M. Kom.**

Every year the number of requests for delivery of goods is always increasing. This increase must be accompanied by a good warehouse management system because if not, it can lead to constraints on delivery time and distribution. The use of autonomous mobile robots is considered capable of making the warehouse inventory management process easier and more efficient. However, this solution has problems with the navigation system. This is because the GPS navigation system cannot be implemented in the warehouse. Therefore, this study presents the main solution to this problem by implementing an autonomous navigation system that focuses on the warehouse environment in the room.

This system consists of robots capable of mapping the surrounding environment and autonomously navigating with encoder sensors, RPLIDAR, and IMU. The software architecture of this system is built on the Robot Operating System (ROS) platform. ROS was chosen because it has many libraries that can support localization, mapping, and navigation. This research applies various algorithms such as Hector SLAM combined with Extended Kalman Filter (EKF) localization to validate the robot's position. The Navfn algorithm is used for global path planning and the Dynamic Windows Approach (DWA) algorithm.

From the experimental results, it was found that the Hector SLAM algorithm is not suitable for use in very large rooms such as warehouses. This is due to the lack of laser range on the surrounding wall, resulting in an error in the particle filter process. This is evidenced by poor map results when testing starting from the middle position of the warehouse when it is far from the wall, while excellent results at the corner position when it is closest to the wall. As for navigation, it was found that the robot maneuvered very well and the Navfn and DWA algorithms used were very precise. This is evidenced by a total of 80% success when testing with static obstacles and a total of 80% success when testing with dynamic obstacles.

Keywords*: autonomous mobile robot, robot operating system (ros), indoor, rplidar, localization, mapping, navigation*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

Pada bab pendahuluan akan membahas terkait latar belakang penelitian, rumusan masalah, tujuan penelitian, manfaat penelitian, serta sistematika penulisan dari penelitian ini.

1.1 Latar Belakang

Menurut Badan Pusat Statistik (BPS, 2019), Indonesia merupakan negara kepulauan yang banyak memanfaatkan jasa transportasi sebagai penghubung antar wilayah, sehingga menciptakan peluang dan pasar yang besar khususnya bagi sektor pos dan kurir modern. Pos dan kurir modern memiliki pasar yang besar pada jasa pengiriman paket, sehingga sektor pergudangan ekspedisi menjadi hal yang penting untuk diperhatikan. Berdasarkan data (BPS, 2018), sejak tahun 2014 hingga 2017, Indonesia mengalami peningkatan PDB di sektor pergudangan, jasa penunjang angkutan, pos dan kurir secara signifikan. Pada laporan Index Triwulan tahun 2019, (BPS, 2020) mengungkapkan bahwa sektor pergudangan mengalami kenaikan eksponensial. Salah satu faktornya disebabkan oleh pertumbuhan *e-commerce* yang cukup besar sejak tahun 2017. Menurut (Databoks, 2021) berdasarkan data (Statista, 2019), tercatat jumlah pengguna *e-commerce* di Indonesia pada 2017 mencapai 139 juta pengguna dan diperkirakan akan mencapai 212,2 juta pengguna pada tahun 2023.

Pada survei yang dilakukan oleh (Parcel Perform, 2019), mengemukakan bahwa 36% konsumen melihat pengiriman sebagai masalah paling besar dalam *e-commerce*. Dari 36% pelanggan yang menyatakan ketidakpuasan, 90% di antaranya mengeluhkan tentang keterlambatan pengiriman, waktu transit tidak memenuhi harapan, dan kurangnya komunikasi tentang status pengiriman. Dari kumpulan fakta tersebut, disimpulkan bahwa setiap tahunnya jumlah permintaan pengiriman barang selalu meningkat, akan tetapi sistem distribusi dan sortir di gudang penyimpanan mengalami kendala pada waktu karena banyaknya paket yang masuk dan lamanya proses sortir. Selain itu peningkatan kebutuhan jasa ekspedisi juga akan berdampak pada cepatnya proses masuk dan keluar barang yang terjadi di gudang penyimpanan ekspedisi. Proses suplai yang cepat ini akan mempengaruhi waktu, tenaga dan biaya yang digunakan di dalam gudang. Meningkatnya waktu dan tenaga yang dibutuhkan untuk manajemen persediaan akan mempengaruhi biaya operasional kerja.

Dalam jurnalnya (Koumanakos, 2008) mengatakan bahwa hampir semua literatur tentang manajemen persediaan yang optimal menggunakan kriteria biaya minimal dan keuntungan maksimal. Untuk itu, penting untuk memiliki biaya persediaan yang rendah dan pada saat yang sama dapat memberikan apa yang diinginkan konsumen. Sehingga diperlukannya solusi yang mampu mengurangi inefisiensi manajemen persediaan tanpa menambah jumlah waktu dan tenaga. Dari masalah yang ada, sangat diperlukan suatu sistem yang dapat menggantikan peran manusia dalam melakukan distribusi barang di dalam gudang.

Penulis berhipotesis sebuah solusi dari masalah ini berupa pengembangan *autonomous mobile robot* dengan sistem yang saling terintegrasi. Sistem saling terintegrasi antar robot atau dengan server akan membuat proses manajemen persediaan gudang menjadi lebih mudah dan efisien, hal ini di uji dengan waktu manuver dan komputasi sistem. Robot bergerak otonom memiliki tugas utama untuk membawa barang dari proses penyortiran hingga transit di dalam gudang. Jika semua proses sortir dikerjakan oleh robot-robot secara otomatis dan saling terintegrasi, maka proses distribusi akan semakin cepat sehingga dengan kondisi ini pihak ekspedisi bisa berfokus pada manajemen alur logistik dan penjadwalan.

Akan tetapi solusi dari robot ini memiliki permasalahan pada sistem navigasinya, hal ini disebabkan sistem navigasi global tidak bisa diimplementasikan di dalam gudang atau ruangan. Dalam tesisnya (Utomo, 2015) mengatakan bahwa permasalahan utama *autonomous mobile robot* terletak pada masalah navigasi, masalah penentuan posisi dan masalah pemetaan. Robot membutuhkan perencanaan jalur untuk navigasi, namun perencanaan jalur hanya bisa dilakukan jika lingkungan sekitar sudah diketahui. Jadi untuk melakukan navigasi, robot harus mampu melakukan pemetaan area kerja terlebih dahulu sejauh batas area cakupan di dalam ruangan gudang. Untuk melakukan pemetaan diperlukan lokasi robot secara akurat selama proses pemetaan yang dilakukan secara simultan. Ketika peta sudah dibuat maka robot baru dapat melakukan penentuan jalur dan akhirnya melakukan navigasi. Menurut (Galov, dkk., 2014) proses berurut ini merupakan metode yang dilakukan untuk permasalahan sistem robot di dalam ruangan yang memerlukan proses simultan dalam waktu yang singkat, metode komputasi ini disebut dengan *Simultaneous Localization and Mapping* (SLAM).

Ada beberapa penelitian yang dilakukan sebelumnya, penelitian yang pertama dilakukan oleh (Utomo, 2015) dengan menggunakan Hector SLAM sebagai algoritme pemetaan. Hasil pemetaan dari penelitian ini yang tidak begitu baik, disebabkan sensor ultrasonik yang memiliki jangkauan pembacaan terbatas. Penelitian kedua dilakukan oleh (Putra, dkk., 2020) yang mengimplementasikan sistem navigasi *indoor* berbasis ROS menggunakan Hector SLAM dan EKF sebagai lokalisasinya. Hasil pemetaan sangat baik, akan tetapi terdapat banyak noise sehingga pada proses navigasi, terdapat percobaan yang gagal karena mengira noise tersebut adalah halangan. Penelitian ketiga yang dilakukan oleh (Liu, dkk., 2021) dengan menggunakan algoritme Navfn dan DWA *local planner* sebagai perencanaan jalur dengan peta yang dibuat menggunakan GMapping. Keseluruhan penelitian ini berjalan dengan baik dan kombinasi algoritme navigasi yang digunakan sangat cocok satu dengan yang lain. Namun kekurangannya adalah proses pengujian tidak mencoba menggunakan halangan yang bergerak jauh lebih dinamis, sehingga tidak mengoptimalkan penggunaan perencanaan jalur lokal.

Untuk itu dari permasalahan ketiga peneliti sebelumnya, penulis mengusulkan suatu robot otonom dengan sensor laser yang memiliki pembacaan luas. Dengan akurasi yang tinggi, dapat diimplementasikan pada pemetaan dan

navigasi menggunakan algoritme Hector SLAM dan Navfn pada platform *Robot Operating System* (ROS). (Utomo, 2015) menyarankan untuk menggunakan sensor laser yang memiliki jangkauan dengan sudut 360 derajat seperti sensor RPLIDAR dan sensor IMU GY-521 untuk mendapatkan nilai akselerasi robot. Dalam bukunya (Brombach, 2021) menyarankan untuk menggunakan Jetson Nano 4GB sebagai kontroler utama robot karena *resource* pemrosesannya yang cukup besar dan sensor encoder untuk pembacaan data *odometry* roda.

Navigasi robot menggunakan perencanaan jalur global Navfn untuk mengoptimalkan waktu dan *resource* komputasi karena area gudang yang luas. Kemudian sebagai tambahan, digunakan perencanaan jalur lokal dengan DWA ketika mendeteksi halangan bergerak. Sebelum melakukan navigasi robot membutuhkan data peta lokal, maka dari itu Hector SLAM akan berperan sebagai pemetaan. Selama proses pemetaan dan navigasi berlangsung, robot perlu mengetahui lokasi terkini secara simultan melalui proses lokalisasi. Lokalisasi yang dipakai menggunakan dua sensor pendeteksi percepatan, yakni sensor IMU dan *odometry*. Kedua sensor ini digabungkan menggunakan algoritme EKF sekaligus melakukan koreksi kesalahan. Pada proses navigasi, untuk mengonfirmasi posisi robot agar benar-benar presisi dan menghindari kesalahan posisi yang besar dibutuhkan lokalisasi dengan data laser dan peta menggunakan algoritme AMCL.

Hasil akhir dari penelitian ini adalah purwarupa robot otonom yang digunakan di dalam gudang penyimpanan dengan mengimplementasikan proses navigasi menggunakan peta. Harapan dari penelitian ini, agar perusahaan jasa ekspedisi di Indonesia dapat meningkatkan efisiensi waktu, tenaga kerja dan mengurangi biaya produksi pada proses distribusi atau sortir barang dengan memanfaatkan pengembangan *autonomous mobile robot* dari hasil penelitian ini

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dijelaskan sebelumnya, maka rumusan masalah dalam penelitian ini adalah sebagai berikut.

1. Bagaimana hasil akurasi dari pembacaan sensor *Odometry*, IMU, dan RPLIDAR?
2. Bagaimana hasil bentuk peta yang dihasilkan pada proses pemetaan menggunakan algoritme Hector SLAM?
3. Bagaimana hasil akurasi dan waktu komputasi pada proses navigasi global dan lokal menggunakan algoritme Navfn dan DWA pada saat tidak ada halangan?
4. Bagaimana hasil akurasi dan waktu komputasi pada proses navigasi global dan lokal menggunakan algoritme Navfn dan DWA pada saat menghadapi halangan dinamis?

1.3 Tujuan

Berdasarkan rumusan masalah yang telah dijelaskan sebelumnya, maka tujuan dari penelitian ini adalah sebagai berikut.

1. Menguji hasil akurasi dari pembacaan sensor *Odometry*, IMU, dan RPLIDAR.
2. Menguji hasil akurasi bentuk peta yang dihasilkan pada proses pemetaan menggunakan algoritme Hector SLAM
3. Menguji hasil akurasi dan waktu komputasi pada proses navigasi global dan lokal menggunakan algoritme Navfn dan DWA pada saat tidak ada halangan
4. Menguji hasil akurasi dan waktu komputasi pada proses navigasi global dan lokal menggunakan algoritme Navfn dan DWA pada saat menghadapi halangan dinamis

1.4 Manfaat

Manfaat yang diharapkan adalah agar penelitian ini bisa menjadi solusi efisien dalam proses pendistribusian barang di dalam gudang penyimpanan ekspedisi, sehingga proses logistik pada sistem ekspedisi pengiriman paket menjadi lebih cepat dan terotomatisasi. Purwarupa *autonomous mobile robot* ini juga bisa menjadi referensi pengembang robot selanjutnya dalam pembuatan robot secara masif karena biaya pembuatan robot yang terjangkau. Selain itu penelitian ini juga diharapkan bermanfaat bagi akademik di lingkungan Universitas Brawijaya sebagai referensi penggunaan *middleware robot operating system* (ROS), sehingga mampu meningkatkan pengembangan robot mendekati pengembangan perangkat lunak yang digunakan di Industri.

1.5 Batasan Masalah

Untuk menjawab rumusan masalah dan agar penelitian ini dapat terselesaikan dengan tenggat waktu yang ada, maka peneliti akan menerapkan batasan masalah sebagaimana berikut:

1. Penelitian berfokus pada manuver pergerakan robot untuk mampu menjalankan sistem navigasi otomatis.
2. Robot tidak mampu menaiki tangga dan *lift* secara mandiri.
3. Robot tidak mampu membuka atau menutup pintu.
4. Robot hanya mampu melakukan navigasi berdasarkan koordinat tujuan yang dikirimkan dari komputer server.
5. Robot hanya mampu membawa barang tidak lebih dari 60 kg.
6. Robot tidak mampu mendeteksi objek berupa kaca, plastik transparan, air dan benda-benda lainnya yang mampu membelokkan cahaya *infrared*.
7. Robot tidak mampu mendeteksi halangan yang berada di bawah dan di atas bidang datar 2 dimensi laser sensor RPLIDAR, termasuk tanjakan/ganjalan parkir.

8. Robot tidak mampu mendeteksi selokan, lubang besar, anak tangga, dan permukaan yang tidak rata lainnya.
9. Robot tidak mampu mendeteksi berat atau kondisi barang yang dibawa.

Robot dan komputer server harus terus terhubung dengan *tethering smartphone* selama proses navigasi berlangsung.

1.6 Sistematika Pembahasan

Tugas akhir skripsi ini terbagi atas tujuh bagian utama dengan penjabaran sebagaimana berikut:

1. BAB 1: Pendahuluan

   Bab ini berisi penjelasan mengenai beberapa hal yang memberitahukan hal-hal mendasar dari sebuah penelitian, seperti latar belakang, rumusan masalah, tujuan, manfaat, batasan masalah, dan sistematika pembahasan.

2. BAB 2: Landasan Kepustakaan

   Bab ini membahas landasan teori yang akan digunakan dalam penelitian ini yang memiliki keterkaitan dengan robot otonom, *autonomous mobile*, robot, metode pemetaan, sistem navigasi, sensor yang digunakan dan penjelasan beberapa algoritme yang digunakan serta penjelasan penggunaan *middleware Robot Operating System* (ROS) beserta *library/framework* yang digunakan sebagai penunjang dalam proses penelitian, tentunya tidak keluar dari batasan masalah yang sudah ditetapkan.

3. BAB 3: Metodologi

   Bab ini menjelaskan tahapan apa saja yang perlu dilakukan oleh peneliti untuk menyelesaikan permasalahan yang dibahas dalam penelitian. Bab ini juga memuat informasi tipe penelitian apa yang diambil oleh penulis dan juga pembahasan mengenai strategi dari penelitian.

4. BAB 4: Rekayasa Kebutuhan Sistem

   Bab ini membahas penguraian kebutuhan pokok baik fungsional maupun non fungsional mengenai mekanisme dari perancangan perangkat keras dan perangkat lunak yang akan digunakan dalam penelitian ini.

5. BAB 5: Perancangan dan Implementasi

   Bab ini berisi penjelasan mengenai proses perancangan pada sistem dan cara pengoperasiannya sebagai jawaban dari permasalahan dan rekayasa kebutuhan yang sudah didefinisikan sebelumnya. Terdiri dari perancangan dan implementasi perangkat keras maupun perangkat lunak.

6. BAB 6: Pengujian dan Analisis

Bab ini membahas percobaan dari hasil perancangan dan implementasi mengenai data dari keluaran sistem dengan bentuk pengujian yang berbeda-beda dan melakukan analisa data dengan parameter yang berbeda untuk bisa membandingkan tingkat akurasi dan keberhasilan sistem.

7. BAB 7: Penutup

Bab ini berisi tentang kesimpulan yang ditarik dari hasil dari pengujian dan analisis yang dilakukan sebelumnya dari sisi perancangan, implementasi, dan pengujian sehingga mampu dijadikan acuan dalam pengembangan penelitian sejenis yang terangkum dalam saran.

BAB 2 LANDASAN KEPUSTAKAAN

Dalam bab landasan kepustakaan ini, penulis akan menjabarkan mengenai tinjauan pustaka yang telah penulis pelajari sebelumnya dan dasar teori yang berisikan teori pendukung penelitian ini. Adapun tinjauan pustaka adalah sekumpulan penelitian sebelumnya yang dilakukan oleh peneliti di dunia dalam lingkup serupa dan bagaimana hasil penelitian tersebut. Lalu mengenai dasar teori, penulis akan menjabarkan dasar fundamental agar tercapainya tujuan akhir penelitian ini.

2.1 Tinjauan Pustaka

Penelitian ini merupakan bentuk dari pengembangan dan gabungan dari penelitian-penelitian sebelumnya yang berhubungan dengan metode komputasi, pemetaan, lokalisasi, navigasi, sensor-sensor dan algoritme yang digunakan. Tinjauan pustaka dilakukan untuk memperdalam teori dasar yang akan dibahas dan sebagai bahan pembelajaran pada penelitian sebelumnya yang terangkum dan tersusun ke dalam sebuah tabel yang dapat dilihat pada Tabel 2.1.

Tabel 2.1 Tinjauan Penelitian Sebelumnya

No	Tinjauan Pustaka	Kesamaan	Perbedaan	
			Penelitian Terdahulu	Rencana Penelitian
1	(Utomo, 2015). *Autonomous Mobile* Robot Berbasis *Landmark* Menggunakan *Particle Filter* Dan *Occupancy Grid Maps* Untuk Navigasi, Penentuan Posisi, Dan Pemetaan	Menggunakan jenis algoritme lokalisasi yang sama *Extended Kalman Filter* (EKF) untuk melakukan *fusion* pada sensor	Penelitian menggunakan sensor ultrasonik, PSD, akselerometer, sensor kamera dari platform robot HBE Robocar	Penelitian menggunakan sensor RPLIDAR, IMU dan *Odometer* dari DYI robot beroda *diff-drive* yang otonom
			Pengujian dilakukan dengan membandingkan hasil pemetaan	Pengujian dilakukan dengan menunjukkan hasil pemetaan dan hasil navigasi
			Arsitektur akhir penelitian menggunakan pemetaan *Occupancy grid* dan navigasi	Arsitektur akhir penelitian menggunakan pemetaan Hector SLAM dan navigasi

			menggunakan *Dynamic* A*	menggunakan *global planning* dengan Navfn dan *local planning* dengan *Dynamic Windows Approach* (DWA)
2	(Zhang, dkk., 2020). 2D LiDAR-*Based SLAM and Path Planning for Indoor Rescue Using Mobile Robots*	Salah satu algoritme yang digunakan sebagai perbandingan pada penelitian sama, yakni Hector SLAM	Pengujian berfokus pada *mapping* saja seperti perbandingan algoritme *mapping* menggunakan GMapping, Hector SLAM, dan Cartographer untuk mencari hasil pemetaan yang optimal	Pengujian berfokus pada hal yang lebih luas seperti pengujian semua sensor yang digunakan hingga pemetaan, navigasi dan lokalisasi.
		Menggunakan platform perangkat lunak yang sama, yakni ROS	Menggunakan perangkat personal komputer atau laptop intel i5	Menggunakan perangkat Jetson Nano
3	(Putra, dkk., 2020). Navigasi *Indoor* Berbasis Peta Pada Robot Beroda Dengan Platform *Robot Operating System*	Penelitian menggunakan metode pemetaan yang sama, yakni Hector SLAM	Pengontrol utama robot adalah ODROID-XU4 dan robot dikontrol dengan 4 buah roda sebagai penggeraknya	Menggunakan jenis robot *differential drive* pada kemudinya, yakni hanya menggunakan 2 buah roda sebagai penggeraknya
		Penelitian ini menggunakan algoritme *fusion* EKF,	Menggunakan algoritme Dynamic A-Star untuk *global path planning*	Menggunakan algoritme Navfn untuk *global path planning*

4	(Erlangga, dkk., 2019). Sistem Navigasi Mobile Robot Dalam Ruangan Berbasis *Autonomous Navigation*	Penelitian ini menggunakan algoritme AMCL untuk lokalisasi navigasinya	Menggunakan algoritme g-*mapping* untuk pemetaan	Menggunakan algoritme Hector SLAM untuk pemetaan
		Penelitian ini menggunakan algoritme DWA untuk *local planner*	Menggunakan sensor RGB-D untuk mendapatkan data	Menggunakan sensor RPLIDAR
5	(Vamsi, dkk., 2021). ROS *Based Autonomous Disinfectant Mobile Robot for Hospitals*	Penelitian ini menggunakan perencanaan global yang sama yaitu Navfn dan DWA *local planner* untuk menentukan pengambilan jalur	Kegunaan untuk menyemprotkan desinfektan pada ruangan di rumah sakit	Kegunaan untuk membawa barang dari satu titik ke titik yang lain
6	(Pyo, dkk., 2017). *Robot Programming from The Basic Concept to Practical Programming and Robot Application*	Buku ini membahas penuh mengenai *Robot Operating System* dan semua parameter di dalamnya	Menggunakan robot simulasi	Menggunakan robot sungguhan

2.1.1 *Autonomous Mobile Robot* Berbasis Landmark Menggunakan Particle Filter Dan Occupancy Grid Maps Untuk Navigasi, Penentuan Posisi, Dan Pemetaan

Penelitian yang dilakukan oleh (Utomo, 2015) mengimplementasikan mengenai 3 proses utama dalam *autonomous mobile robot* yaitu navigasi, posisi dan pemetaan. Kesimpulan yang didapatkan dari penelitian ini adalah bahwa sensor yang terbatas seperti kamera dan ultrasonik bisa digunakan sebagai pemetaan dan menerapkan metode SLAM. Hal ini didapat setelah melihat hasil dari pengujian, didapati hasil berupa hasil pemetaan. Dari hasil *mapping*

menunjukkan cell *occupied* (warna putih) tidak berada di tepi peta. Hal ini disebabkan karena sensor ultrasonik pada robot hanya berada di bagian depan robot. Sedangkan di sisi kanan dan kiri pada badan robot tidak terdapat sensor ultrasonik. Kemudian Semakin kecil jumlah partikel, diperlukan iterasi yang lebih banyak dalam mencapai konvergen partikel (estimasi pose (x, y, θ)). Lalu Semakin banyak jumlah partikel, waktu komputasi yang diperlukan semakin lama. Hasil dari algoritmenya Penggunaan algoritme A* dapat membantu robot menemukan jalur terpendek dengan *error* pose robot untuk data arah sebesar 4.34 % dan *error* jarak tempuh robot sebesar 4.8 %.

2.1.2 2D LiDAR-Based SLAM and Path Planning for Indoor Rescue Using Mobile Robots

Penelitian yang dilakukan oleh (Zhang, dkk., 2020) mengimplementasikan 4 algoritme untuk pemetaan yakni GMapping, Hector-SLAM, Cartografer dan RGB-D. Penelitian ini membahas mengenai pemetaan dan navigasi akan tetapi yang lebih terlihat dengan detail adalah proses penggunaan algoritme untuk proses *Mapping*. Dari pengujian yang dilakukan didapatkan kesimpulan bahwa algoritme GMapping, Hector-SLAM dan Cartographer memiliki performa yang baik di dalam ruangan. Sedangkan RGB-D sangat buruk karena tidak bisa mendeteksi pada kondisi pencahayaan yang redup. Kemudian didapatkan data bahwa GMapping dan Cartographer memiliki potensi yang sangat baik dalam melakukan pemetaan. Terutama hasil peta yang dibuat dengan Cartographer menghasilkan peta yang paling akurat dengan bentuk bangunan aslinya (Zhang, dkk., 2020).

2.1.3 *Navigasi Indoor* Berbasis Peta Pada Robot Beroda Dengan Platform Robot Operating System

Penelitian yang dilakukan oleh (Putra, dkk., 2020) mengimplementasikan sistem navigasi *indoor* berbasis peta pada robot beroda dengan platform *Robot Operating System* (ROS) menggunakan algoritme Hector *Mapping* untuk pemetaan. Hasilnya adalah robot berhasil melakukan pemetaan lingkungan *indoor* dengan *error* rata-rata 0.174 meter dan berhasil melakukan lokalisasi dengan *error* rata-rata 0.05 meter pada koordinat x, 0.028 meter pada koordinat y, dan 1.506° pada sudut orientasi. *Path planner* memiliki tingkat keberhasilan sebesar 62.5% dalam menghasilkan jalur yang dapat dilewati robot dan tingkat keberhasilan 75% dalam mengikut jalurnya. Kesimpulan yang didapatkan pada jurnal ini adalah bahwa pemetaan menggunakan Hector SLAM, lokalisasi data fusion menggunakan Extended Kalman Filter (EKF), *path planning* dengan Field Dynamic A-Star dan Trajectory tracking dengan pengontrol *on-off* berhasil diimplementasikan dengan ROS. Hasil yang didapat berupa kesimpulan bahwa kecepatan robot berpengaruh terhadap hasil pemetaan. Lokalisasi *Wheel Odometer* berhasil menentukan posisi robot dengan tingkat kesalahan kurang dari 0.05 meter. Untuk navigasi peneliti sebelumnya menyarankan untuk menggunakan *local path planner* agar robot dapat beroperasi pada lingkungan dinamis serta menggunakan PID untuk meningkatkan performa robot.

2.1.4 Sistem Navigasi *Mobile Robot* Dalam Ruangan Berbasis *Autonomous Navigation*

Penelitian yang dilakukan oleh (Erlangga, dkk., 2019) mengimplementasikan navigasi *mobile robot* dalam ruangan yang menerapkan sistem kemudi otonom. Pembuatan peta menggunakan algoritme *Simultaneous Localization and Mapping* (SLAM) yang mengolah data dari sensor kamera RGB-D dan bumper yang di konversi ke laser *scan* dan *point cloud* digunakan untuk memperoleh *perception*. Sedangkan *wheel encoder* dan *gyroscope* digunakan untuk mendapatkan data *odometry* yang digunakan untuk membangun peta perjalanan dengan algoritme SLAM, *GMapping* dan melakukan *autonomous navigation*. Sistem terdiri dari tiga sub-sistem yaitu: sensor sebagai input, single *board* computer untuk proses, dan aktuator sebagai penggerak. Hasil dari pengujian menunjukkan sistem dapat memberikan data *depth* yang dikonversi ke laser scan, data *bumper*, dan data *odometry* kepada single *board* computer berbasis ROS sehingga *mobile* robot yang dikendalikan secara *Wireless* dari *Workstation* dapat membangun peta *grid* 2 dimensi dengan akurasi total *error* rate sebesar 0.987%. Dengan menggunakan peta, data dari sensor, dan *odometry* tersebut *mobile* robot dapat melakukan *autonomous navigation* secara konsisten dan mampu melakukan *path replanning*, menghindari rintangan-rintangan statis dan terus menerus melakukan *localization* untuk mencapai titik tujuan.

2.1.5 Based Autonomous Disinfectant Mobile Robot for Hospitals

Penelitian ini dilakukan oleh (Vamsi, dkk., 2021) dengan menggunakan simulasi 3D Gazebo untuk menyimulasikan robot desinfektan pada rumah sakit. Data dari sensor yang digunakan antara lain adalah LIDAR, IMU, dan *Odometry*. Pada penelitian ini metode pencarian jalur global menggunakan algoritme Navfn dan pencarian jalur global menggunakan DWA. Pada hasil implementasi simulasinya, didapatkan hasil bahwa robot dapat bergerak secara dinamis menghindari halangan. Perencanaan global dengan algoritme Navfn memberikan hasil yang baik dibuktikan dengan output dari sistem yang berhasil mencapai tujuan. Kesimpulan dari penelitian ini adalah simulasi *autonomous mobile robot* ini menunjukkan bahwa dimungkinkannya menggunakan robot ini di dunia nyata dengan lingkungan yang berubah-ubah secara dinamis seperti di rumah sakit. Kemudian semua algoritme yang digunakan sudah saling melengkapi, sangat cocok, dan efisien berdasarkan hasil pengujian.

2.1.6 Robot Programming from The Basic Concept to Practical Programming and Robot Application

Buku ini ditulis oleh (Pyo, dkk., 2017) membahas dengan detail mengenai *robot operating system* (ROS) dari pengertian hingga proses navigasi di dalam ROS. Dalam bukunya, penulis juga menjelaskan secara detail bagaimana membuat aplikasi dengan *middleware* ROS dan parameter-parameter apa saja yang digunakan. Banyak sekali metode dan algoritma yang dibahas dalam buku ini, salah satunya adalah algoritma Navfn dan *Dynamic Window Approach.* Kedua

algoritma ini digunakan untuk proses penentuan jalur pada navigasi. Selain itu buku ini juga membahas dengan lengkap mengenai lokalisasi dan pemetaan. Kesimpulan dari buku ini adalah dijadikannya buku ini sebagai panduan utama dalam melakukan penelitian. Buku yang ditulis oleh (Pyo, dkk., 2017) dirasa akan sangat membantu penulis di dalam melakukan Penelitian, sehingga buku ini dijadikan sebagai tinjauan pustaka utama.

2.2 Dasar Teori

Pada subbab dasar teori menjabarkan terkait teori yang digunakan dalam melakukan penelitian, tujuannya adalah memperjelas sistematika dan kerangka pemikiran dari fakta yang ada dalam rumusan suatu hubungan konsep dari masing-masing ilmu pengetahuan.

2.2.1 Robot

Robot adalah alat berupa orang-orangan dan sebagainya yang dapat bergerak (berbuat seperti manusia) yang dikendalikan oleh mesin (KBBI, 2016). Menurut (Bekey, 2005) robot didefinisikan sebagai sebuah mesin yang mampu merasakan, berpikir, dan bertindak. Dengan demikian, robot harus dilengkapi dengan sensor, kemampuan pemrosesan yang meniru beberapa aspek kecerdasan dan aktuator. Sensor dibutuhkan untuk memperoleh informasi dari lingkungan, kecerdasan bawaan diperlukan jika robot ingin melakukan tugas-tugas penting secara mandiri, dan *actuation* diperlukan untuk memungkinkan robot mengerahkan kekuatan pada lingkungan. Ilustrasi mengenai robot dapat dilihat pada Gambar 2.1.

Gambar 2.1 Robot *Humanoid* Pada Lomba *RoboCup*

Sumber: (www.robocupgermanopen.de, 2020)

2.2.2 Autonomous Mobile Robot

Menurut (Bekey, 2005) *autonomous robot* atau robot otonom adalah mesin cerdas yang mampu melakukan tugas di dunia sendiri, tanpa kontrol manusia yang eksplisit. Kata otonom mengacu pada sistem yang mampu beroperasi di lingkungan dunia nyata tanpa kontrol eksternal untuk waktu yang lama. Menurut (Todd, 1986) *mobile robot* atau robot bergerak adalah mesin yang dapat bergerak secara keseluruhan dengan cara yang terkontrol dengan beberapa tingkat otonomi. Sebagian besar robot bergerak menggunakan roda karena

metode penggeraknya yang paling sederhana. Contoh dari *autonomous mobile robot* adalah robot helikopter otonom hingga *Roomba*, robot penyedot debu. Ilustrasi mengenai *Roomba,* robot penyedot debu dapat dilihat pada Gambar 2.2.

Gambar 2.2 Robot *Roomba* Penyedot Debu

Sumber: (www.irobot.com, 2013)

Menurut (Bekey, 2005) robot memiliki banyak tipe dan jenis yang berbeda-beda, banyak jenis robot yang hanya digunakan untuk proses navigasi saja menggunakan remote *control* atau ada pula robot yang hanya digunakan sebagai sistem tertutup yang memerlukan input dari manusia, jadi tidak semuanya bisa disebut sebagai robot otonom.

Gambar 2.3 *Autonomous Mobile Robot*

Sumber: (www.panasonic.com, 2020)

Pada Gambar 2.3 dapat dilihat bahwa terdapat robot drone dan robot beroda yang memiliki tipe penggerak yang beda namun memiliki tugas yang sama. Menurut (Bräunl, 2008) salah satu tipe penggerak pada *mobile* robot adalah tipe robot beroda *differential drive.* Penggerak ini terdiri dari dua roda penggerak dan memiliki satu atau lebih roda pasif atau *caster*. Keuntungan dari desain penggerak *differential drive* adalah aksi kemudi dan berbelok telah sepenuhnya dipisahkan dengan menggunakan dua motor yang berbeda. Oleh karena itu, perangkat lunak kontrol untuk mengemudi akan sangat sederhana. Kerugian dari desain ini adalah bahwa robot tidak dapat berputar di tempat, karena roda yang digerakkan tidak terletak di tengahnya.

Menurut (Bodrenko, 2019) *autonomous mobile robot* tepat digunakan pada sektor yang mengangkut barang dalam jumlah banyak, misalnya sektor pergudangan ekspedisi. Robot bergerak dapat dimanfaatkan pada proses distribusi ataupun penyortiran paket kiriman.

2.2.2.1 *Autonomous Mobile Robot* Di Bidang Pergudangan

Gambar 2.4 Penggunaan Robot Pada Gudang Logistik

Sumber: (www.ttnews.com, 2020)

Mengutip dari (Zetes, 2021) perusahaan pengembang *autonomous mobile robot* untuk logistik, menjelaskan bahwa *autonomous mobile robot* bekerja secara kolaboratif dengan operator gudang, baik itu memindahkan stok selama pemetikan, mengambil pengisian ulang, atau mentransfer stok massal. Robot ini melakukan tugas dengan mengangkut barang ke tahap pemrosesan berikutnya, dengan ini memungkinkan pekerja untuk melanjutkan ke tugas berikutnya sehingga proses distribusi barang menjadi lebih cepat seperti yang diperlihatkan pada Gambar 2.4. *Autonomous Mobile Robot* (AMR) memiliki karakteristik dan tugas utama dalam pekerjaannya, meliputi (Zetes, 2021):

a. Beroperasi secara mandiri, AMR tidak memerlukan infrastruktur khusus untuk bekerja, seperti sebuah garis atau jalur lintasan khusus.
b. Sensor, peta, dan sistem pemrosesan terpasang memungkinkan robot untuk merencanakan rute dan beradaptasi secara dinamis dengan perubahan di sekitarnya.
c. AMR dapat memindahkan stok di dalam gudang atau antar fasilitas. Robot ini juga dapat melakukan tugas rutin seperti penghitungan inventaris dan pengecekan stok barang.

Gambar 2.5 Robot Membawa Paket Di Gudang Ekspedisi

Sumber: (www.zetes.com, 2022)

Penelitian akan mengadaptasi bentuk robot dari pergudangan yang memiliki bagian atap dengan luas yang lebar sebagai alas atau dudukan barang. Ilustrasi dari robot pembawa barang di dalam gudang dapat dilihat pada Gambar 2.5.

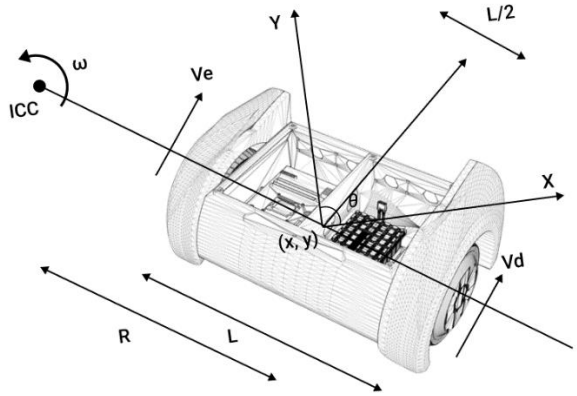
2.2.3 Mekanisme Kemudi Robot Beroda

Robot beroda memiliki beberapa jenis penggerak dengan teknik yang berbeda. Sistem penggerak yang paling sering dan umum digunakan pada robot beroda, yaitu *Single Wheel Drive, Tracked Robots*, *Ackerman Steering* dan *Differential Drive.* Namun pada penelitian ini akan digunakan sistem *steering differential drive* dengan satu roda statis di bagian depan sehingga terdapat 3 buah roda, penjelasan mengenai mekanisme kemudi sebagai berikut.

2.2.3.1 Differential Drive Robot

Menurut (Dudek, dkk., 2001) dalam bukunya, *differential drive* atau penggerak diferensial adalah sistem penggerak dua roda dengan aktuator independen untuk setiap roda. Meskipun kita dapat memvariasikan kecepatan setiap roda agar robot dapat melakukan gerakan bergulir, robot harus berputar di sekitar titik yang terletak di sepanjang sumbu roda kiri dan kanan. Titik di mana robot berputar dikenal sebagai ICC (*Instantaneous Center of Curvature).*

(Dudek, dkk., 2001) juga menjelaskan bahwa variasi kecepatan kedua roda menentukan lintasan yang diambil robot. Sistem kerja dari mekanisme kemudi ini memanfaatkan gerak *forward kinematic* dengan mengandalkan putaran independen kedua roda untuk mendapatkan manuver yang diinginkan. Roda penggerak biasanya ditempatkan di setiap sisi robot dan mengarah ke depan atau sebaliknya. Menurut (Bräunl, 2008) keuntungan dari desain ini adalah motor dan roda berada pada posisi tetap dan tidak perlu diputar, ini sangat menyederhanakan desain mekanik robot



Gambar 2.6 *Kinematics Differential Drive*

Pada Gambar 2.6 terdapat diagram kinematika dari *differential drive*. Dikarenakan laju putaran terhadap ICC harus sama untuk kedua roda, dapat dituliskan persamaan berikut (Dudek, dkk., 2001):

$$\omega \left(\mathrm{R} + \frac{\mathrm{l}}{2}\right) = \mathrm{Vr} \tag{2.1}$$

$$\omega \left(R - \frac{l}{2}\right) = Vl \tag{2.2}$$

Di mana l adalah jarak dari pusat antara kedua roda, Vl dan Vr adalah kecepatan roda (translasi), R merupakan jarak antara titik tengah l $(l/2)$ jika diukur dari salah satu roda) dan ICC yang merupakan titik di mana robot berotasi (Dudek, dkk., 2001).

Jika persamaan 2.1 dan 2.2 digabungkan dengan persamaan v = ω*R* maka,

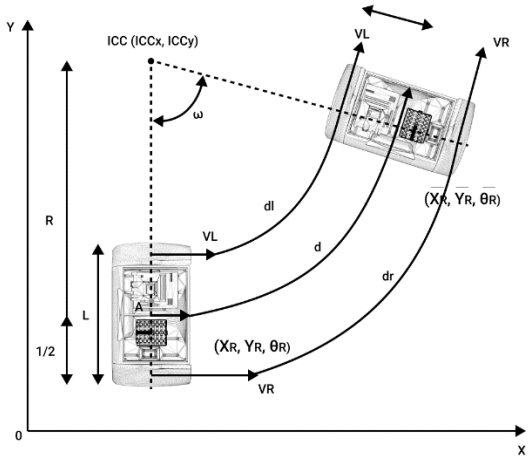
$$R = \left(\frac{L}{2}\frac{vr + vl}{vr - vl}\right) \tag{2.3}$$

$$\omega = \left(\frac{vr - vl}{l}\right) \tag{2.4}$$

Dari persamaan-persamaan di atas dapat ditarik beberapa hal, antara lain:

1. Jika vl = vr, maka akan diperoleh gerak lurus linear ke depan.
2. Jika vl = $-vr$, maka akan diperoleh rotasi pada titik l/2 atau berputar di tempat.
3. Jika vl = 0, maka akan diperoleh gerak poros ke kiri karena R = l/2, kemudian sebaliknya untuk roda kanan jika vr = 0.

Differential drive robot tidak bisa bergerak ke arah sumbu kedua roda atau dengan kata lain, memiliki *non-holonomic constraint*. Selain itu, kebanyakan *differential drive* robot memiliki sebuah roda tambahan yang pasif sebagai *support wheel* agar lebih stabil karena kesalahan kecil dalam kecepatan relatif antara roda dapat mempengaruhi lintasan robot. *Differential drive* juga sangat sensitif terhadap variasi kecil di bidang tanah, dan mungkin memerlukan roda *caster* menopangnya (Dudek, dkk., 2001).



Gambar 2.7 *Forward Kinematics Differential Drive*

Dari Gambar 2.7 digambarkan bahwa untuk mendapatkan gerak berbelok dibutuhkan persamaan *forward kinematic* dengan memanipulasi parameter kontrol Vl dan Vr. Dengan mengetahui kecepatan Vl, Vr, dan menggunakan persamaan 2.3 dan 2.4 kita dapat menemukan nilai ICC (Dudek, dkk., 2001):

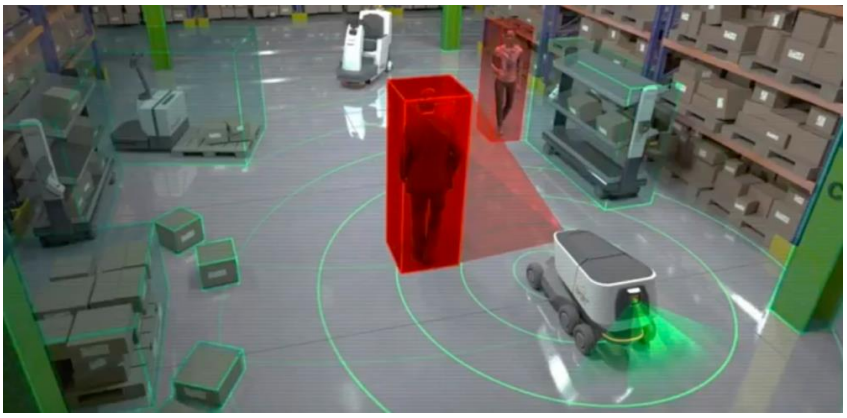
$$ICC = [x - R\,sin(\theta), y + R\,cos(\theta)] \quad (2.5)$$

Dan pada waktu t + δt maka pose robot menjadi:

$$\begin{bmatrix}\bar{x}\\ \bar{y}\\ \bar{\theta}\end{bmatrix} = \begin{bmatrix}cos\,(\omega\delta t) & -sin(\omega\delta t) & 0\\ sin(\omega\delta t) & cos(\omega\delta t) & 0\\ 0 & 0 & 1\end{bmatrix} \quad (2.6)$$

Persamaan ini secara sederhana menggambarkan gerakan robot yang berputar sejauh R terhadap ICC dengan kecepatan sudut ω

2.2.4 Indoor Navigation



Gambar 2.8 Ilustrasi *Indoor Navigation Pada Autonomous Mobile Robot*

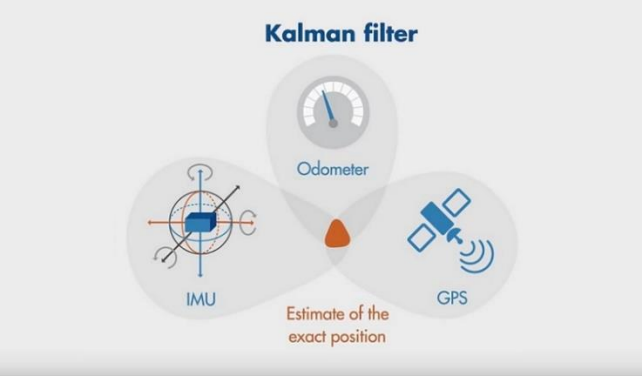
Sumber: (www.therobotreport.com, 2018)

Menurut (Michelson, 2017) navigasi otonom berarti bahwa kendaraan dapat merencanakan jalurnya dan melaksanakan rencananya tanpa campur tangan manusia. Ilustrasi mengenai *autonomous* navigation dapat dilihat pada Gambar 2.8. Cakupan sistem penentuan posisi yang sering digunakan dalam robotika

adalah dan *outdoor navigation* dan *indoor navigation*. Namun pada penelitian kali ini, kita akan mengimplementasikan sistem *indoor navigation*.

Menurut (Rajendra, 2015) navigasi dalam ruangan adalah sistem yang digunakan untuk menemukan lokasi yang tepat di dalam gedung. Dikutip dari jurnal (Putra, dkk., 2020), navigasi berbasis peta (*map-based navigation*) merupakan salah satu metode navigasi *autonomous* yang bisa diterapkan di lingkungan *indoor* terstruktur seperti gudang atau di dalam pusat perbelanjaan.

2.2.5 Kalman Filter Untuk Lokalisasi



Gambar 2.9 Kalman Filter Dengan IMU, Odometer, dan GPS

Dalam buku yang berjudul "*ROS: Robot Programming*" (Pyo, dkk., 2017) menjelaskan bahwa *Kalman Filter* yang digunakan dalam proyek Apollo NASA dikembangkan oleh Dr. Rudolf E. Kalman, yang sejak saat itu menjadi terkenal menjadi sebuah algoritme. Filternya adalah filter *recursive* yang melacak keadaan objek dalam sistem linier dengan noise. *Filter* didasarkan pada probabilitas Bayes yang mengasumsikan model dan menggunakan model ini untuk memprediksi keadaan saat ini dari keadaan sebelumnya. Kemudian, kesalahan antara nilai prediksi dari langkah sebelumnya dan nilai aktual terukur yang diperoleh oleh alat ukur digunakan untuk melakukan langkah pembaruan untuk memperkirakan nilai keadaan yang lebih akurat seperti yang terlihat pada Gambar 2.9.

Berdasarkan penjelasan (Siegwart, dkk., 2004) di dalam teori filter Kalman, sistem diasumsikan linier dan dengan noise *Gaussian*. Teori filter Kalman didasarkan pada asumsi bahwa sistemnya linier dan bahwa secara keseluruhan konfigurasi robot, model kesalahan *odometry*, dan model kesalahan pengukuran dipengaruhi oleh *noise* putih *Gaussian.* Menurut (Pyo, dkk., 2017) *Kalman Filter* hanya dapat diaplikasikan pada sistem yang linier, sedangkan kebanyakan robot memiliki sistem non linier. Maka dari itu dibuat sebuah varian algoritme dari hasil modifikasi Kalman Filter yang dikenal dengan *Extended Kalman Filter* (EKF).

2.2.5.1 Extended Kalman Filter (EKF)

Berdasarkan penjelasan dari (Siegwart, dkk., 2004) sebelumnya, *Extended Kalman Filter* (EKF) adalah versi non linier dari filter Kalman yang terlinear mengenai perkiraan rata-rata dan kovarians saat ini. EKF sering disebut sebagai algoritme fusion sensor karena dalam penggunaannya EKF digunakan untuk

menggabungkan dua buah atau lebih sensor, seperti sensor IMU, GPS, kompas dan *odometry* menjadi satu output. Hasil dari algoritme ini menghasilkan keluaran gabungan sensor yang mampu melacak posisi suatu robot dengan mengalkulasi nilai dari setiap keluaran sensor yang dimasukkan ke dalam parameter EKF. Algoritme lokalisasi ini bertujuan untuk mencapai tingkat akurasi yang tinggi dan cakupan yang luas.

(Pyo, dkk., 2017) mengatakan bahwa EKF *Localization* tergabung ke dalam paket *robot localization* yang tersedia pada paket *driver* ROS dengan *node* paketnya bernama *ekf_localization_node*. Paket ini berisi *template launcher* dan *file* konfigurasi untuk membantu pengguna dalam melakukan lokalisasi. Untuk penggunaannya, paket lokalisasi ini dapat digunakan *pada file launch* secara langsung atau dengan membuat *file config* dengan extensi .yaml yang nantinya juga diimpor pada *file launch.*

(Siegwart, dkk., 2004) menjelaskan bahwa EKF berbasis pada proses iterasi untuk mengestimasi posisi terkini atau suatu state berdasarkan informasi sebelumnya. berdasarkan informasi sebelumnya. Proses estimasi tersebut dapat dideskripsikan sebagai sistem dinamis non linier yang ditunjukkan pada persamaan 2.7. *Observation vector* atau *measurement vector* yang merupakan hasil pengukuran ditunjukkan pada persamaan 2.8. Variabel x_k adalah state sistem robot pada waktu k, x_{k-1} adalah state sistem pada waktu $k-1$, f adalah fungsi yang menunjukkan hubungan state sistem pada waktu k dengan state sistem pada waktu $k-1$. W_{k-1} adalah *process noise,* Z_k *adalah observation vector* (pengukuran pada waktu k), h adalah fungsi non linier yang mendeskripsikan hubungan antara x_k dengan Z_k dan V_k adalah measurement noise dengan distribusi normal. Berikut adalah persamaan yang digunakan pada EKF.

$$x_k = f(x_{k-1} + W_{k-1}) \tag{2.7}$$

$$Z_k = h(x_k + V_k) \tag{2.8}$$

(Siegwart, dkk., 2004) menambahkan bahwa algoritma EKF terdiri dari 2 fase, yaitu *state predict* dan *state update*. *State predict* merupakan fase prediksi terhadap informasi *priori state vector* $\widehat{x_k}$ dengan menggunakan informasi *posteriori state vector* x_{k-1}. Pada fase ini informasi *priori state vector* diperbaiki dengan menggunakan *observation vector* dan Fase *state predict* ditunjukkan oleh persamaan 2.9 dan 2.10 serta fase state *update* ditunjukkan oleh persamaan 2.11, 2.12, dan 2.13.

$$\widehat{x_k} = f(x_{k-1}) \quad (2.9)$$

$$\widehat{P_k} = FP_{k-1}F^T + Q \quad (2.10)$$

$$K = \widehat{P_k}H^T(H\widehat{P_k}H^T + R)^{-1} \quad (2.11)$$

$$x_k = \widehat{x_k} + K(z_k - H\widehat{x_k}) \quad (2.12)$$

$$P_k = (1 - KH)\widehat{P_k}(1 - KH)^T + KRK^T \quad (2.13)$$

Tabel 2.2 Penjelasan Variabel Persamaan Fungsi EKF

Sumber: (Siegwart, dkk., 2004)

No	Notasi	Keterangan
1	$\widehat{x_k}$	Hasil prediksi state sistem (*priori state vector*)
2	P_k	Kovariansi *error* sistem pada waktu k
3	P_{k-1}	Kovariansi *error* pada waktu k-1
4	$\widehat{P_k}$	Hasil prediksi kovariansi *error*
5	*F*	Matriks *Jacobian* dari f
6	Q	*Process noise covariance*
7	K	*Kalman gain*
8	H	Matriks Jacobian dari h
9	R	Kovariansi pengukuran

2.2.6 Simultaneous Localization and Mapping (SLAM)

Menurut (Pyo, dkk., 2017) SLAM (Simultaneous Localization And *Mapping*) artinya menjelajahi dan memetakan lingkungan yang tidak diketahui sambil memperkirakan pose robot itu sendiri dengan menggunakan sensor yang terpasang pada robot. SLAM dikembangkan untuk memungkinkan robot membuat peta dengan atau tanpa bantuan manusia. Ini adalah metode membuat peta saat robot menjelajahi ruang yang tidak diketahui dan mendeteksi sekitarnya dan memperkirakan lokasinya saat ini serta membuat peta.

Menurut (Siegwart, dkk., 2004) tujuan SLAM adalah untuk memulihkan jalur robot dan peta lingkungan hanya dengan menggunakan data yang dikumpulkan oleh sensor *proprioceptive* dan *exteroception*. Data ini biasanya adalah perpindahan robot yang diperkirakan dari *odometry* dan fitur (misalnya, sudut, garis, bidang) yang diekstraksi dari gambar laser, ultrasonik, atau kamera. SLAM berurusan dengan masalah ayam dan telur dalam membangun peta. Untuk melakukan pemetaan dibutuhkan pembaharuan lokasi robot terkini secara simultan. Sedangkan untuk melakukan pembaharuan lokasi dibutuhkan data peta dari hasil pemetaan. Istilah SLAM digunakan untuk masalah itu sendiri serta

untuk solusi algoritme yang mencoba menyelesaikan permasalahan seperti yang diilustrasikan pada Gambar 2.10.

Gambar 2.10 Ilustrasi Permasalahan Komputasi SLAM

Sumber: (arreverie.com, 2018)

2.2.6.1 Hector Simultaneous Localization and Mapping (Hector SLAM)

Mengutip dari (Manuel, dkk., 2018), dijelaskan bahwa Hector SLAM adalah algoritma SLAM yang dikembangkan oleh tim HECTOR *(Heterogeneous Cooperating Team of Robots)* di Universitas *Darmstadt* untuk digunakan pada Robot Darat Tak Berawak, Kendaraan Permukaan Tak Berawak, Perangkat Pemetaan genggam dan data log dari UAV *quadrotor*. Menurut (Kohlbrecher, dkk., 2014) Hector SLAM berisi sekumpulan paket ROS yang digunakan untuk melakukan komputasi SLAM dengan mengkorelasikan perkiraan posisi robot dan peta di lingkungan yang tidak terstruktur.

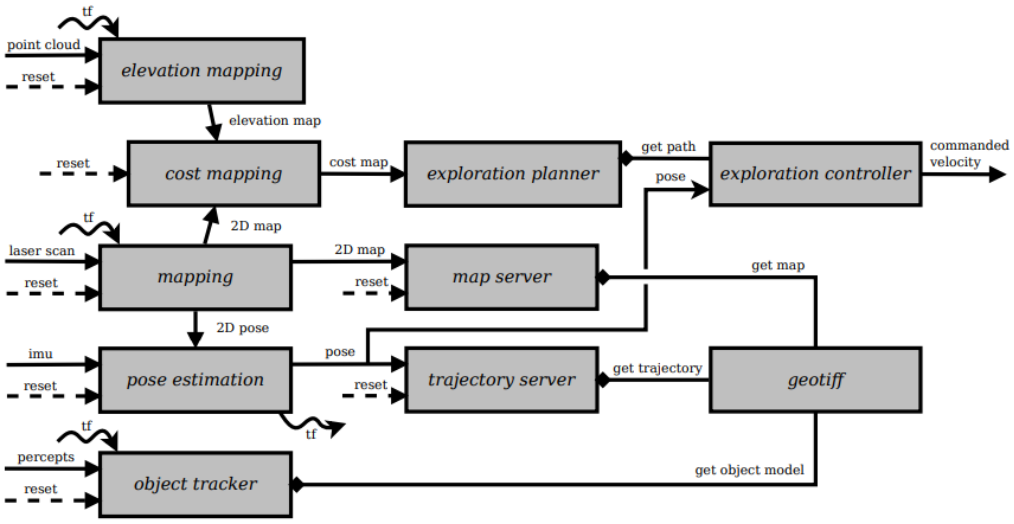
(Manuel, dkk., 2018) menambahkan bahwa Hector SLAM bertentangan dengan solusi SLAM lainnya karena tidak dibagi menjadi *front-end* dan *back-end*. Metode hanya berfungsi sebagai *front-end* dan tidak didasarkan pada grafik atau tidak menyediakan teknik pengoptimalan global apa pun. Pendekatan yang digunakan untuk pencocokan pemindaian didasarkan pada optimasi penyelarasan titik laser dengan peta *grid* dengan menggunakan pengoptimalan berbasis Gauss-Newton dan data baru dari laser ditulis langsung ke peta *grid* setelah diterima. Hector SLAM tidak menyediakan kemampuan penutupan loop eksplisit, sehingga bergantung pada penyelarasan dan pengoptimalan data yang berkelanjutan.

(Manuel, dkk., 2018) berpendapat untuk mengurangi risiko terjebak dalam minimum lokal selama pengoptimalan gradien naik, Hector SLAM menggunakan pendekatan peta *multi-resolution.* Beberapa peta dibuat dengan resolusi berbeda yang diatur dalam piramida, setiap tingkat piramida memiliki setengah resolusi dari tingkat sebelumnya. Gambar 2.11 menyajikan hasil pemetaan simultan *multiresolution* Hector SLAM.

Gambar 2.11 Representasi *Multiresolution* Dari Sel Petak Peta Hector SLAM

Sumber: (Kohlbrecher, dkk., 2011)

Mengutip dari (Kohlbrecher, dkk., 2011) menjelaskan bahwa pada Hector SLAM terdapat proses utama untuk pembuatan peta, yaitu Hector *Mapping*. *Hector Mapping* memanfaatkan sistem LiDAR dengan frekuensi tinggi tanpa menggunakan informasi *Odometry*. *Hector Mapping* menggunakan algoritma *scan matching* yang dikembangkan dengan cara menyelaraskan data dari sensor untuk membangun peta. *Scan matching* adalah proses menyelaraskan hasil pemindaian (pengukuran jarak) pada waktu t dengan LIDAR dengan hasil pemindaian pada waktu t - 1 yang bertujuan untuk memperoleh posisi robot untuk kemudian memperbarui peta. Hector SLAM, *hector mapping*, *hector geotiff*, dan lain-lain beserta alur proses yang terjadi di dalamnya pada skema yang terdapat pada Gambar 2.12.



Gambar 2.12 Skema Sistem Hector SLAM

Sumber : (Kohlbrecher, dkk., 2014)

Seperti yang dapat dilihat pada Gambar 2.12 yang merupakan skema sistem hector SLAM yang dibuat langsung oleh pengembangnya, (Kohlbrecher, dkk., 2014), menjelaskan bahwa skema Hector SLAM selalu melakukan reset pada awalnya. Diagram persegi panjang melambangkan semua *node* yang memegang semacam informasi status berlangganan topik perintah yang terutama digunakan untuk mengatur ulang sistem kapan pun diperlukan. Pada proses *mapping*

terdapat tiga masukkan berupa tf (*transformation*), laser scan dan reset. Data laser scan didapat dari sensor laser lidar, misalnya sensor RPLIDAR, sedangkan tf didapat dari *node* tf yang terdapat pada *file launch* atau konfigurasi EKF. Output dari *mapping* menghasilkan 2D map yang langsung diproses oleh map server. Output yang disimpan dari map server akan disimpan pada geotiff dan menghasilkan *object tracker*.

(Kohlbrecher, dkk., 2014) menjelaskan bahwa hasil dari *mapping* juga menghasilkan 2D pose yang dikirimkan ke proses pose *estimation* yang nantinya digunakan *trajectory* server untuk menyimpan jalur *path* yang sudah dilalui robot. Jika ingin melakukan navigasi, bisa dengan menambahkan paket move_base yang nantinya membuat robot memperhitungkan *cost mapping* dan mampu membuat *exploration planner* sehingga akhirnya mendapatkan *path* yang bebas dari halangan dan memerintahkan motor untuk berputar. *Node* yang terpenting pada proses ini adalah *mapping* dan geotiff yang mana dua *node* ini merupakan proses utama untuk membuat peta menggunakan Hector SLAM. Maka dari itu pada subbab selanjutnya akan dijelaskan *hector mapping* dan *hector geotiff*.

a. Hector Mapping

Seperti yang telah dijelaskan sebelumnya pada jurnal (Kohlbrecher, dkk., 2011), bahwa *Mapping* dengan hector adalah pendekatan SLAM yang dapat digunakan tanpa *odometry* serta pada platform yang menunjukkan gerakan roll/pitch (dari sensor, platform, atau keduanya). Dikombinasikan dengan *attitude estimation system* dan opsi unit pitch/roll untuk menstabilkan pemindai laser, sistem ini dapat menyediakan peta lingkungan bahkan jika tanahnya tidak datar seperti yang ditemui di arena Penyelamatan RoboCup.

Dalam jurnalnya (Kohlbrecher, dkk., 2011) terdapat beberapa persamaan yang menjadi formula utama pada proses pemetaan dengan *Hector Mapping*. Dijelaskan bahwa metode *scan matching* pada Hector *Mapping* berbasis pada pendekatan Gauss- Newton. Perpindahan posisi robot dihitung dengan persamaan 2.14 dan nilai H dihitung dengan persamaan 2.15. Untuk mengetahui penjelasan lebih lanjut mengenai kegunaan atau fungsi dari setiap variabel dapat dilihat pada Tabel 2.3.

$$\Delta\xi = H^{-1} \sum_{i=1}^{r} \left[\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi}\right]^T [1 - M(S_i(\xi))] \quad (2.14)$$

$$H = \sum_{i=1}^{r} \left[\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi}\right]^T \left[\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi}\right] \quad (2.15)$$

Tabel 2.3 Penjelasan Variabel Persamaan Fungsi Hector *Mapping*

Sumber: (Kohlbrecher, dkk., 2011)

No	Notasi	Keterangan
1	$\Delta\xi$	Perpindahan posisi robot dalam koordinat Kartesian (x, y, θ)

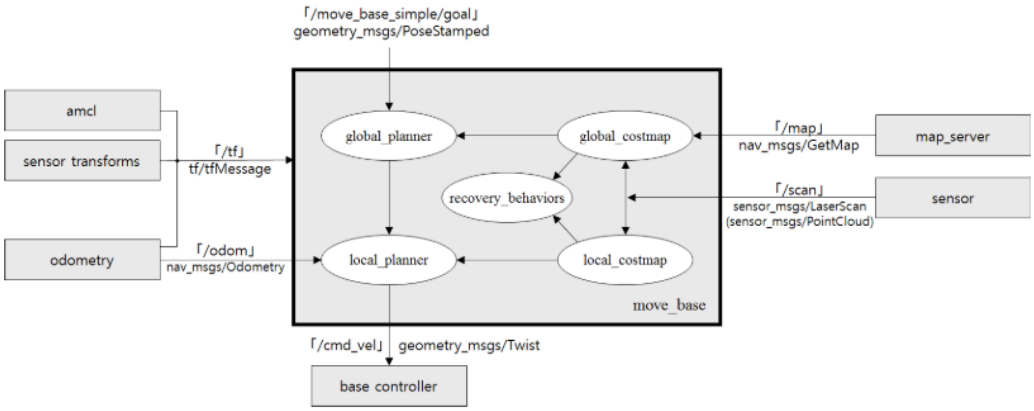
2	r	Jumlah titik pemindaian LIDAR
3	$S_i(\xi)$	Koordinat-koordinat hasil pemindaian LiDAR berdasarkan posisi robot $\xi(x, y, \theta)$
4	$M(S_i(\xi))$	Probabilitas adanya halangan pada $S_i(\xi)$

b. Hector Geotiff

Hector Geotiff adalah *node* yang dapat digunakan untuk menyimpan peta jaringan hunian, lintasan robot, dan data objek yang diinginkan ke peta format Geotiff dengan informasi georeferencing. Sedangkan peta Geotiff adalah gambar peta dengan bentuk berupa data *occupancy grid* berisikan data *free space* dan *obstacle*. Dalam ROS Geotiff digunakan sebagai format standar untuk peta, jadi bukan format gambar biasa seperti JPEG atau PNG melainkan Geotiff dengan extensi format .pgm. *Node* ini dapat berjalan di atas sistem robot dan menyimpan peta ke penyimpanan permanen berdasarkan pengatur waktu, mengurangi kemungkinan hilangnya peta jika terjadi masalah konektivitas.

2.2.7 Navigasi

Menurut (Pyo, dkk., 2017) navigasi adalah memindahkan robot dari satu lokasi ke tujuan yang ditentukan dalam lingkungan tertentu. Untuk tujuan ini, peta yang berisi informasi geometri furnitur, benda, dan dinding dari lingkungan yang diberikan diperlukan. Seperti yang dijelaskan pada bagian SLAM sebelumnya, peta dibuat dengan informasi jarak yang diperoleh sensor dan informasi pose robot itu sendiri. Sedangkan untuk pengertian navigasi dikutip dari penelitian (Utomo, 2015) dalam tesisnya mengatakan navigasi diartikan sebagai proses atau aktivitas untuk merencanakan atau menuju jalur secara langsung dalam sebuah misi yang diberikan pada sebuah *autonomous mobile robot* dari satu tempat ke tempat yang lain tanpa kehilangan arah atau mengalami tabrakan dengan objek. Navigasi memungkinkan robot untuk berpindah dari pose saat ini ke pose tujuan yang ditentukan dengan menggunakan peta, *encoder* robot, sensor inersia, dan sensor jarak.



Gambar 2.13 Hubungan Antara *Node* dan Topik Pada Konfigurasi Navigasi

Sumber: (Pyo, dkk., 2017)

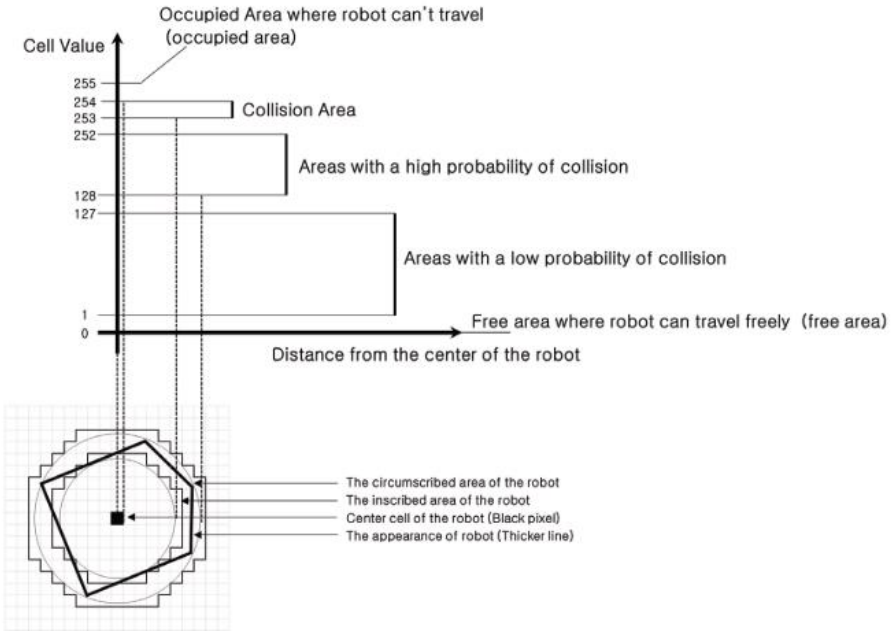
Pada Gambar 2.13 terdapat diagram transformasi yang menyusun proses navigasi pada sebuah robot. Proses utama yang terjadi pada navigasi terletak pada bagian diagram tertutup (kotak) dengan melakukan beberapa pengambilan data dari *node* lain. (Pyo, dkk., 2017) juga menambahkan bahwa, dalam navigasi terdapat suatu peta biaya atau *costmap* yang membuat proses navigasi terhindar dari tabrakan, tidak lupa AMCL untuk lokalisasi robot. Selain itu terdapat pula sistem perencanaan jalur (*path planning)* untuk mencari jalur terpendek dari posisi saat ini menuju posisi tujuan

2.2.7.1 Costmap

Dalam bukunya (Pyo, dkk., 2017) menjelaskan bahwa dalam navigasi, peta biaya atau *costmap* dipakai untuk menghitung area rintangan, kemungkinan area tabrakan, dan robot yang dapat dipindahkan daerah berdasarkan empat faktor. Faktor tersebut adalah peta statis, manfaatkan area yang diduduki, area bebas, dan area yang tidak diketahui untuk navigasi. *Costmap* dapat dibagi menjadi dua, salah satunya adalah 'global_costmap', yang menyiapkan rencana jalur untuk bernavigasi di area global peta tetap. Yang lainnya adalah 'local_costmap' yang digunakan untuk perencanaan jalur dan penghindaran rintangan di area terbatas di sekitar robot. Meskipun tujuannya berbeda, kedua peta biaya direpresentasikan dengan cara yang sama.

(Pyo, dkk., 2017) menjelaskan bahwa peta biaya dinyatakan sebagai nilai antara '0' dan '255'. Nilai digunakan untuk mengidentifikasi apakah robot dapat bergerak atau bertabrakan dengan rintangan. Perhitungan setiap area tergantung pada parameter konfigurasi *costmap* yang dapat diubah oleh pengembang. Nilai tersebut dapat dilihat pada Gambar 2.14 sebagai berikut:

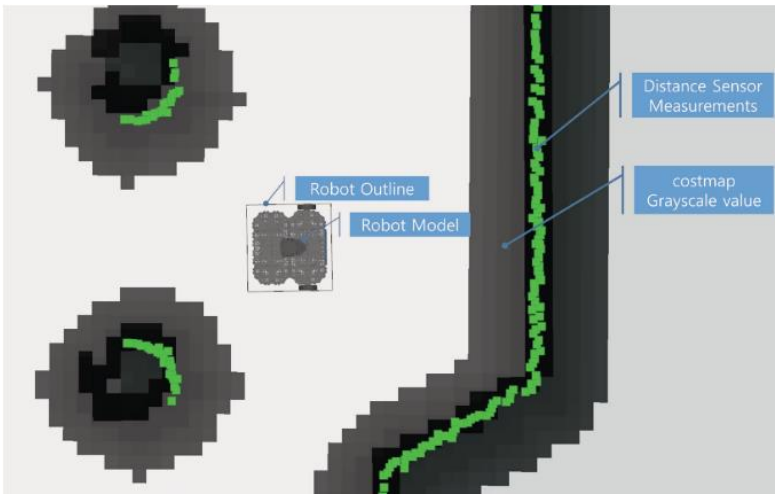
1. 000: Area bebas di mana robot dapat bergerak bebas
2. 001~127: Area dengan kemungkinan tabrakan rendah
3. 128~252: Area dengan kemungkinan tabrakan tinggi
4. 253~254: Area tabrakan
5. 255: Area pendudukan di mana robot tidak bisa bergerak



Gambar 2.14 Hubungan Antara Jarak ke Rintangan dan Nilai *Costmap*

Sumber: (Pyo, dkk., 2017)

Sebagai contoh pada Gambar 2.15, terdapat model robot di tengah dan kotak persegi panjang di sekelilingnya sesuai dengan permukaan luar robot. Ketika garis *outline* ini menyentuh dinding, robot akan menabrak dinding juga (Pyo, dkk., 2017). Warna hijau mewakili hambatan dengan nilai sensor jarak yang diperoleh dari sensor laser. Sebagai *costmap* skala abu-abu menjadi lebih gelap, lebih cenderung menjadi area tabrakan. Jadi robot tidak akan mendekati bagian yang cenderung gelap atau hitam karena *costmap* sudah menganggap bahwa daerah tersebut rawan terjadi tabrakan.



Gambar 2.15 Representasi *Costmap*

Sumber: (Pyo, dkk., 2017)

2.2.7.2 Adaptive Monte Carlo Localization (AMCL)

Mengutip dari (Pyo, dkk., 2017) menjelaskan bahwa AMCL (Adaptive Monte Carlo Localization) adalah algoritme untuk menentukan di mana robot berada di lingkungan tertentu. Metode ini dapat dianggap sebagai versi perbaikan dari estimasi pose Monte Carlo, yang meningkatkan kinerja waktu nyata dengan mengurangi waktu eksekusi dengan jumlah sampel yang lebih sedikit dalam algoritma estimasi pose Monte Carlo. AMCL menggunakan metode *particle filter* yang merupakan metode lokalisasi dengan menggunakan data jarak dari laser.

AMCL menggunakan beberapa persamaan pada prosesnya, seperti posterior *probability* menggunakan Bayes, *generate* dengan particle filter dan *sampling* and *resampling* untuk mendapatkan lokasi dengan presisi. Dengan mengulangi proses SIR (*Sampling Importance weighting Re-sampling*) sambil menggerakkan partikel, perkiraan posisi robot meningkat akurasinya (Pyo, dkk., 2017).

2.2.8 Path Planning

Mengutip dari (Pyo, dkk., 2017), dijelaskan bahwa *path planning* atau perencanaan jalur adalah metode komputasi untuk menciptakan lintasan dari pose saat ini ke pose target yang ditentukan pada peta. Rencana jalur yang dibuat mencakup perencanaan jalur global di seluruh peta dan perencanaan jalur lokal untuk area yang lebih kecil di sekitar robot. Terdapat 2 hal pada perencanaan jalur, yakni global plan dan *local plan*.

2.2.8.1 Global Planning

Menurut (Pyo, dkk., 2017) perencanaan global adalah metode dalam membuat jalur global ke target utama untuk menggerakkan robot. (Marin-Plaza, dkk., 2018) menjelaskan bahwa *global plan* membutuhkan peta lingkungan untuk menghitung rute terbaik. Maka dari itu diperlukan peta sebelum melakukan navigasi dengan global plan

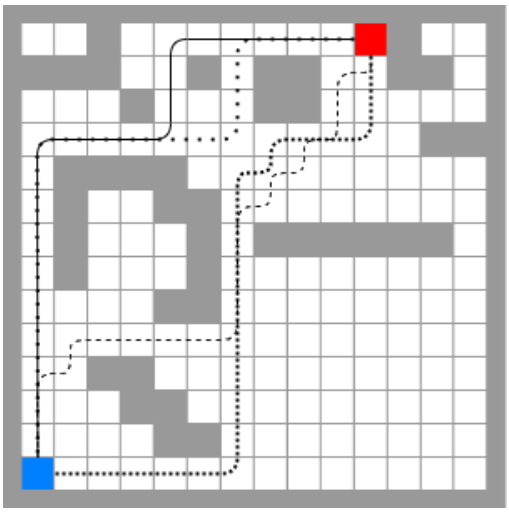
2.2.8.2 Local Planning

Local planner atau perencana lokal merupakan metode untuk mengubah jalur global menjadi perencanaan jalan lokal yang sesuai dengan kondisi sekitar (Marin-Plaza, dkk., 2018). Menurut (Pyo, dkk., 2017) perencanaan lokal menggunakan informasi *odometry* untuk melihat kecepatan robot agar mampu menghindari halangan pada perencanaan jalur global.

2.2.9 Algoritme Navfn

Menurut (Pyo, dkk., 2017) metode perencanaan global yang digunakan pada paket *move base* adalah Navfn dalam menentukan jalur paling optimal dari titik awal hingga titik tujuan. Navfn diciptakan untuk melakukan optimalisasi pada algoritme Dijkstra dikarenakan Dijkstra mengharuskan pembacaan pada seluruh *node* di terdekatnya hingga ke tujuan. Fungsi navigasi dihitung dengan algoritma Dijkstra, tetapi dukungan untuk *heuristic* A* juga dapat ditambahkan. Paket Navfn menyediakan implementasi fungsi navigasi interpolasi yang cepat yang

digunakan untuk membuat rencana untuk basis robot bergerak. Karena merupakan algoritme optimalisasi dari Dijkstra maka Navfn memiliki kesamaan pada caranya dalam mencari tetangga, namun bedanya tidak seperti Dijkstra yang mengunjungi semua tetangga, Navfn hanya mengunjungi tetangga yang memiliki potensi terdekat dengan titik tujuan yang disebut dengan *grid gradient*.



Gambar 2.16 *Grid Map* Hasil Pemetaan

Pada Gambar 2.16 terlihat ilustrasi dari proses *path planning* dari kotak biru menuju kotak merah. Terdapat banyak kemungkinan jalur yang akan dituju, robot tidak mengetahui mana jalur terpendek dan bagaimana mengambil keputusan untuk melewati suatu jalur. Maka dari itu diperlukan sebuah perencanaan global yang menghitung di mana jarak terpendek dari titik awal menuju titik tujuan (Bräunl, 2008).

Perencanaan global selalu berhubungan dengan sebuah area yang sudah diketahui jarak antar *node*, maka dari itu diperlukan sebuah peta yang mampu menyimpan data koordinat. Peta ini merupakan bentuk dari proses *mapping* dan menghasilkan berupa *file Geotiff* dengan bentuk *Grid Map*. Dengan adanya peta kita dapat menentukan jalur dengan menelusuri titik per titik dari peta yang sudah tersedia menggunakan algoritme Navfn. Jika berbicara mengenai Navfn maka sama saja seperti membahas algoritme Dijkstra namun dengan optimalisasi, maka dari itu perlu untuk mengetahui algoritme Dijkstra berikut penjelasan algoritme tersebut

2.2.9.1 Implementasi Algoritme Dijkstra pada Navfn

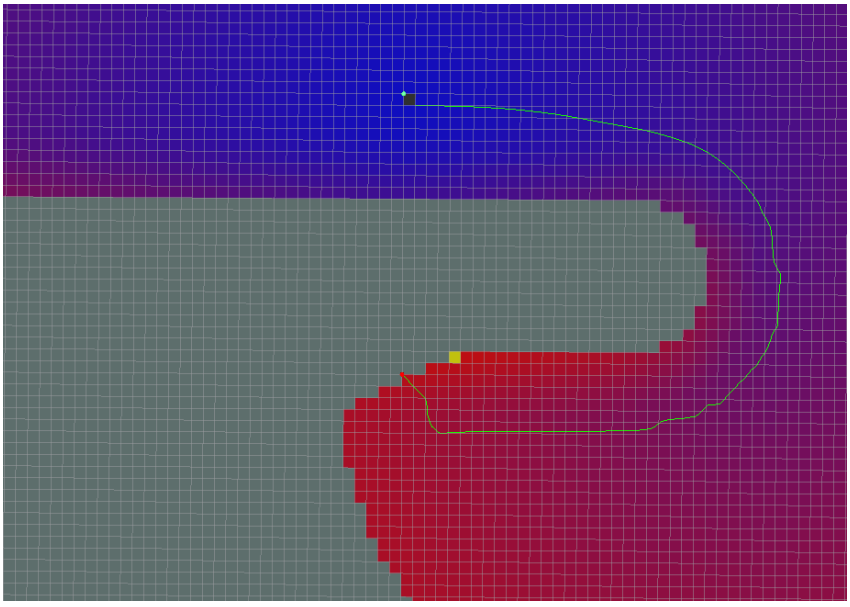
Menurut (Bräunl, 2008) Dijkstra adalah algoritme untuk menghitung semua jalur terpendek dari *node* awal yang diberikan dalam grafik yang terhubung penuh. Algoritme ini ditemukan oleh (Dijkstra, 1959) yang ditujukan untuk menyelesaikan dua masalah dalam suatu hubungan grafik. (Siegwart, dkk., 2004) menjelaskan bahwa dalam aplikasi robot, pencarian Dijkstra biasanya dihitung dari tujuan robot posisi. Akibatnya, tidak hanya jalur terbaik dari *node* awal ke tujuan yang dihitung, tetapi juga semua jalur biaya terendah dari setiap posisi awal dalam grafik ke *node* tujuan. Hasil dari implementasi algoritme ini dijadikan

sebuah alur pemrograman ke dalam *pseudocode* yang dapat dilihat pada Tabel 2.4.

Tabel 2.4 *Pseudocode* Algoritme Dijkstra

```
1   BEGIN function dijkstra(G, S)
2       for each vertex V in G
3           distance[V] <- infinite
4           previous[V] <- NULL
5           If V != S, add V to Priority Queue Q
6       distance[S] <- 0
7
8       while Q IS NOT EMPTY
9           U <- Extract MIN from Q
10          for each unvisited neighbour V of U
11              tempDistance <- distance[U] + edge_weight(U, V)
12              if tempDistance < distance[V]
13                  distance[V] <- tempDistance
14                  previous[V] <- U
15  END return distance[], previous[]
```

(Iovino, 2019) mengatakan bahwa perencana global Navfn hanya mengimplementasikan algoritme Dijkstra dan karena ia bekerja dengan baik, tidak ada waktu tambahan yang perlu dihabiskan untuk menemukan solusi lain yang menggunakan A*. Jadi secara pemrograman algoritme yang digunakan hanyalah Dijkstra dengan optimalisasi, maka dari itu secara *pseudocode* tidak terlalu berbeda. Hasil yang diharapkan dari perencanaan global menggunakan Navfn dapat dilihat pada Gambar 2.17.

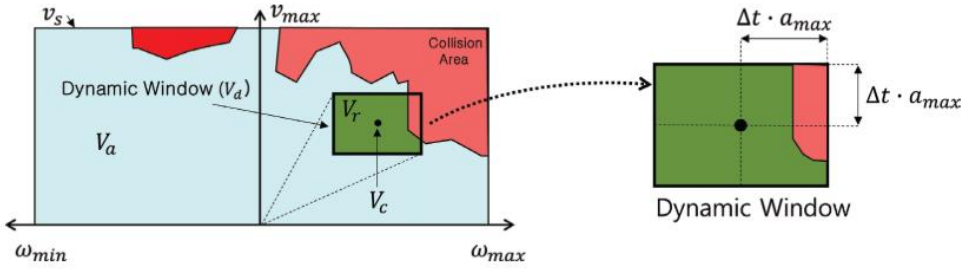


Gambar 2.17 Hasil Perencanaan Jalur Dengan Algoritme Navfn

Sumber: (wiki.ros.org, 2022)

2.2.10 Algoritme Dynamic Window Approach

Menurut (Pyo, dkk., 2017) *Dynamic Window Approach* (DWA) adalah metode populer untuk perencanaan penghindaran rintangan dan menghindari rintangan. Ini adalah metode pemilihan kecepatan yang dapat dengan cepat mencapai titik target sambil menghindari rintangan yang mungkin dapat bertabrakan dengan robot di ruang pencarian kecepatan. Perencana lintasan digunakan untuk perencanaan jalur lokal, tetapi DWA diganti karena kinerjanya yang unggul.



Gambar 2.18 Ruang Pencarian Kecepatan Robot dan *Dynamic Window*

Sumber: (Pyo, dkk., 2017)

Pertama, robot tidak berada pada koordinat sumbu x dan y, tetapi pada ruang pencarian kecepatan dengan kecepatan translasi v dan kecepatan rotasi sebagai sumbunya, seperti terlihat pada Gambar 2.18. Di ruang ini, robot memiliki kecepatan maksimum yang diizinkan karena keterbatasan perangkat keras, dan ini disebut Jendela Dinamis. Berikut ini adalah persamaan yang digunakan pada DWA (Pyo, dkk., 2017):

$$V_m = \{v \in [V_{min}, V_{max}], \omega \in [\omega_{min}, \omega_{max}]\}. \quad (2.16)$$

$$G_{(v,\omega)} = \gamma \times vel(v,\omega) + \alpha \times \mathrm{heading}(v,\omega) + \beta \times distant(v,\omega) \quad (2.17)$$

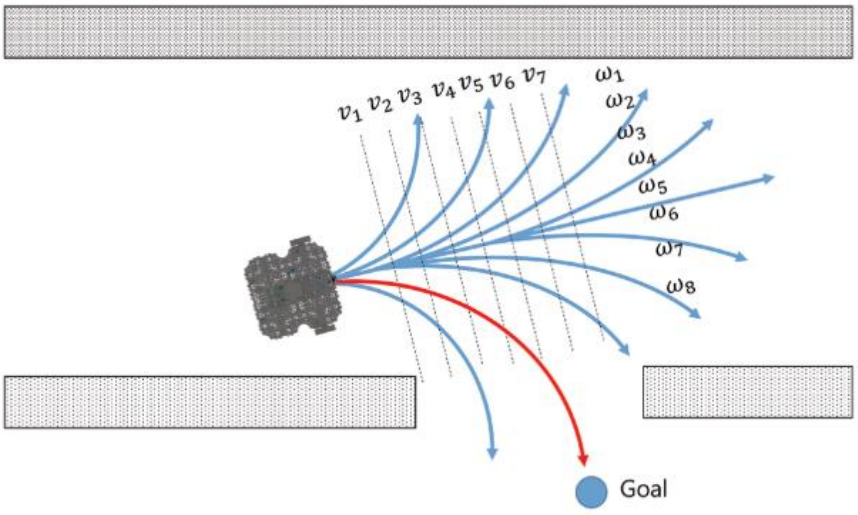
Yang merupakan fungsi utama dari DWA, mengenai penjelasan dari setiap variabel dapat dilihat pada Tabel 2.5.

Tabel 2.5 Persamaan pada Dynamic Window Approach

Sumber: (Pyo, dkk., 2017)

No	Notasi	Keterangan
1	v	Kecepatan translasi (meter/detik)
2	ω	Kecepatan rotasi (radian/dtk)
3	V_s	Area kecepatan maksimum
4	V_a	Area kecepatan yang diizinkan
5	V_c	Kecepatan arus
6	V_r	Area kecepatan di Dynamic Window
7	a_{max}	Tingkat akselerasi / deselerasi maksimum

8	heading(v, ω)	180 - (selisih antara arah robot dan arah titik target)
9	dist(v, ω)	Jarak ke rintangan
10	velocity(v, ω)	Kecepatan yang dipilih
11	α, β, γ	Konstanta pembobotan
12	$\sigma(x)$	Fungsi Halus



Gambar 2.19 Kecepatan translasi v dan kecepatan rotasi w

Sumber: (Pyo, dkk., 2017)

(Pyo, dkk., 2017) mengatakan bahwa dalam DWA, fungsi tujuan $G_{(v,\omega)}$ digunakan untuk menghitung translasi kecepatan v dan kecepatan rotasi ω yang memaksimalkan fungsi objektif yang mempertimbangkan arah, kecepatan dan tubrukan. Jika kita plot, kita dapat menemukan kecepatan optimal di antara berbagai opsi v dan ω ke tujuan seperti yang ditunjukkan pada Gambar 2.19

Akan lebih mudah untuk menjelaskan alur proses algoritma ini jika kita melihat kodenya terlebih dahulu dalam *pseudocode* DWA pada Tabel 2.6 sebagai berikut (Pyo, dkk., 2017):

Tabel 2.6 *Pseudocode* Dynamic Window Approach

Sumber: (Iovino, 2019)

```
1   BEGIN function create_trajectories(robot.pose, robot.vel,
    robot.acc)
2
3     // initialization of each velocity value (vx, vy, ω)
4     max_velocities ← MAX_velocities
5
6     // velocity values that allow the robot to stop in the
7   goal
      if goal_position is valid then
```

```
8       goal_distance ← sqrt(goal_x-robot_x^2 + goal_y-robot_y^2)

9       max_vx ← min(MAX_Vx, goal_distance/sim_time)
10      min_vx ← min(max(MIN_Vx, -goal_distance/sim_time),
    goal_distance/sim_time)
11      max_vy ← min(MAX_Vy, goal_distance/sim_time)
12      min_vy ← -MAX_Vy
13
14    // escaping maneuver: test with slower velocities and
    more samples
15    if escaping then
16      max_vx ← max_vx/2
17      min_vx ← -max_vx
18      max_vy ← max_vx
19      min_vy ← -max_vy
20      max_ω ← max_ω/2
21
22      vx_samples ← vx_samples×2
23      vy_samples ← vy_samples×2
24      ω_samples ← ω_samples×2
25
26    // compute Dynamic Window velocities taking into account
    accelerations
27    max_vx ← max(min(max_vx, robot_vx + robot_accx ×
    sim_period), min_vx)
28    min_vx ← max(min_vx, robot_vx − robot_accx × sim_period)
29    max_vy ← max(min(max_vy, robot_vy + robot_accy ×
    sim_period), min_vy)
30    min_vy ← max(min_vy, robot_vy − robot_accy × sim_period)
31    max_ω ← min(max_ω, robot_ω + robot_accθ × sim_period)
32    min_ω ← max(min_ω, robot_ω − robot_accθ × sim_period)
33
34    // initialize best and comparative trajectory
35    best_traj ← null
36    comp_traj ← null
37    // TRAJECTORY COMPUTATION: begin
38    // sampling interval
39    δ_vx ← (max_vx − min_vx)/(vx_samples − 1)
40    δ_ω ← (max_ω − min_ω)/(ω_samples − 1)
41
42    vx_samp ← min_vx // first tested value for vx
43    for ix = 0 to vx_samples do // loop through all x
    velocities
44      // global y maximum of the ellipses with respect to
    the vx sample
45      if vx_samp ≥ 0 then
46        local_MAX_Vy ← MAX_Vy × sqrt(1-(vx_samp/MAX_Vx)^2)
47      else
48        local_MAX_Vy ← MAX_Vy × sqrt(1-(vx_samp/MIN_Vx)^2)
```

```
49      // local y max and min of the dynamic window with
    respect to the vx sample
50      local_max_vy ← min(MAX_Vy, max_vy)
51      local_min_vy ← max(-MAX_Vy, min_vy)
52
53      δ_vy ← (local_max_vy - local_min_vy)/(vy_samples - 1)
54      vy_samp ← local_min_vy
55      for iy = 0 to vy_samples do
56         ω_samp ← min_ω
57         for iθ = 0 to ω_samples do
58            // if the tested velocity is too small, ignore
    it
59            if |vx_samp| ≤ MAX_Vx/100 and |vy_samp| ≤
    MAX_Vy/100 and |ω_samp| ≤ MAX_ω/100 then
60                 continue
61
62            // generate the trajectory with these velocity
    values
63            generate_trajectory (robot.pose, robot.vel,
    robot.acc, vx_samp, vy_samp, ω_samp,comp_traj)
64
65            // compare the trajectories and take the best 1
66            if comp_traj better than best_traj then
67               best_traj ← comp_traj
68
69            // update velocity samples and close loops
70            ω_samp ← ω_samp + δ_ω
71         vy_samp ← vy_samp + δ_vy
72       vx_samp ← vx_samp + δ_vx
73    // TRAJECTORY COMPUTATION: end
74
75  END return best_traj
```

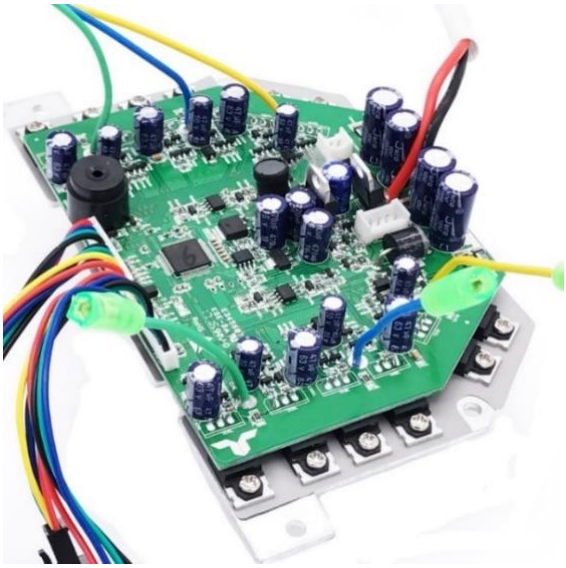
2.2.11 Komponen Utama Robot

Komponen utama robot pengantar barang berisikan alat-alat yang digunakan guna membangun sistem robot ini. Pada subbab ini akan dijelaskan spesifikasi dan pengertian dari setiap komponen yang digunakan. Berikut ini adalah komponen utama yang dibutuhkan dalam membuat robot pengantar barang.

2.2.11.1 Hoverboard Motherboard

Menurut (Joseph, 2018) *driver motor* adalah papan sirkuit elektronik yang menyesuaikan kecepatan motor dengan memasukkan sinyal modulasi lebar-pulsa (PWM) sebagai input. Menurut (Reilly, 2003) *motherboard* adalah papan logika utamanya yang berisi unit pemrosesan pusat dan *chip* memori. Pada penelitian ini, *motherboard hoverboard* digunakan sebagai *mainboard* atau papan pemrosesan utama dari robot sekaligus sebagai *driver* motor dc. Di dalam *motherboard* terdapat *chip* mikroprosesor dengan seri GD32F103RCT6 yang

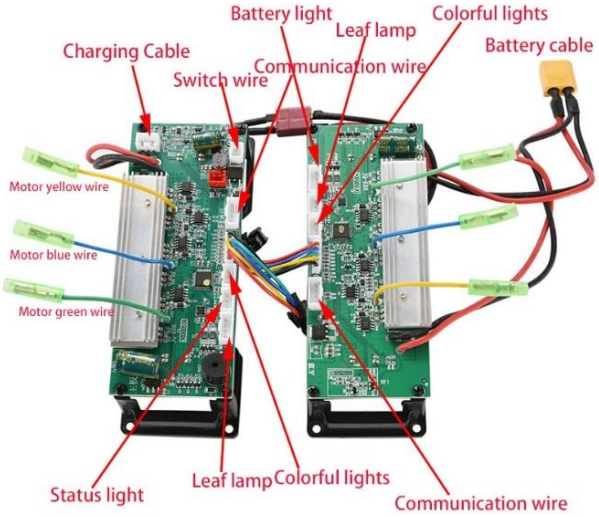
digunakan sebagai otak dari kontroler sekaligus sebagai *driver* untuk mengontrol kecepatan dan arah putaran roda. Gambar 2.20 merupakan gambar *motherboard hoverboard*.



Gambar 2.20 *Motherboard Hoverboard*

Sumber: (github.com/EFeru/*hoverboard*-firmware-hack-FOC, 2021)

Perlu diperhatikan bahwa *motherboard hoverboard* memiliki tipe dan bentuk yang berbeda-beda. Pada penelitian ini digunakan *motherboard hoverboard* dengan tipe *single motherboard* dan berbentuk seperti pada Gambar 2.20. Hal ini perlu diperhatikan karena penggunaan selain *single motherboard* akan sangat menyulitkan proses *flashing firmware* karena perbedaan *port* dan fabrikasi. Misalnya saja *dual board motherboard* pada Gambar 2.21 yang memiliki *port flashing* di kedua *board*, jadi selalu gunakan *single motherboard*.



Gambar 2.21 *Motherboard Hoverboard* Tipe *Dual Board*

Sumber: (aliexpress.com, 2021)

2.2.11.2 Nvidia Jetson Nano

Menurut (Williman, dkk., 1976) sistem komputer mikro berisi komputer mikro ditambah catu daya pendukung, panel kontrol, dll., yang memberi pengguna kemampuan yang hampir sama dengan komputer mini. Hal ini termasuk *microprocessor*, memori dan sirkuit input/output (I/O) yang dipasang pada satu papan sirkuit tercetak (PCB) dan Jetson Nano termasuk salah satu komputer mini tersebut. Penelitian ini menggunakan komputer mini Jetson Nano sebagai pemrosesan utama kedua setelah *microcontroller hoverboard*. Penggunaan dari Jetson Nano berdasarkan saran dari (Brombach, 2021), yang menyarankan untuk menggunakan Jetson Nano 4GB karena jauh lebih kuat dan jauh lebih baik dibanding dengan Raspberry Pi 3. Bentuk fisik dari Nvidia Jetson Nano dapat dilihat pada Gambar 2.22.

Gambar 2.22 Nvidia Jetson Nano

Sumber: (www.nvidia.com, 2021)

2.2.11.3 Aktuator Motor BLDC *Hoverboard* dan Sensor *Rotary Encoder*

Menurut (Wang, 2012) Motor BLDC memiliki beberapa keunggulan dibandingkan motor DC *brushed* konvensional, terutama pada karakteristik kecepatan versus torsi yang lebih baik, efisiensi tinggi, respons dinamis tinggi, masa pakai yang lama, pengoperasian tanpa suara, rentang kecepatan yang lebih tinggi, dan perawatan yang rendah. BLDC menggunakan metode kontrol konduksi berpasangan, maka ketika arah putaran BLDC bolak-balik, hanya ada dua MOSEFT daya yang menyala. Ketika sensor *Hall* mendeteksi posisi baru berikutnya, yang memberi tahu mikroprosesor *chip* GD32 untuk memberikan sinyal kontrol baru.

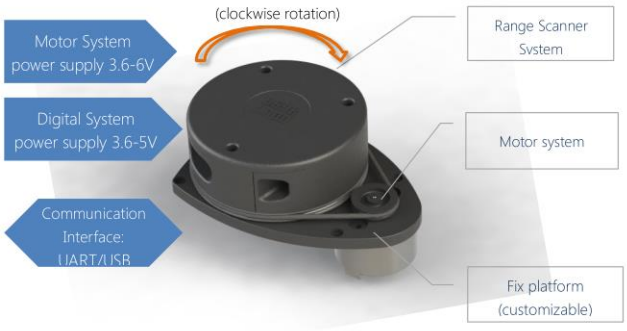
Kontroler menyesuaikan fase dan amplitudo pulsa arus DC untuk mengontrol kecepatan dan torsi motor. Sistem kontrol ini merupakan alternatif dari komutator mekanis (sikat) yang digunakan di banyak motor listrik konvensional (Wang, 2012).

Gambar 2.23 Motor BLDC 6.5” 36V 350W dan *build in Encoder*

Sumber: (www.*hoverboard*forlife.blogspot.com, 2016)

Pada Gambar 2.23 dapat dilihat bahwa di dalam motor BLDC terdapat sensor *encoder* yang berfungsi untuk mengetahui posisi motor berdasarkan data yang dikirimkan dari HALL sensor. Sensor *Encoder* ini akan mengirimkan data berupa *odometry* yang nantinya akan digunakan sebagai input pada sistem. Pembahasan lebih lanjut mengenai *odometry* akan dibahas pada bagian selanjutnya.

2.2.11.4 Sensor RPLIDAR A1



Gambar 2.24 Sensor RPLIDAR

Sumber: (www.RoboPeak.com, 2013)

Mengutip dari (RoboPeak Team, 2014) RPLIDAR A1 (RoboPeak *Light Distance and Ranging*) adalah pemindai laser 2D (*Laser Distance Sensor*) 360 derajat berbiaya rendah yang dikembangkan oleh RoboPeak dengan versi penjualan A1 atau versi LiDAR pertama. Menggunakan teknologi sensor jarak jauh yang memakai laser untuk memindai halangan atau dinding dan menghitung jaraknya. Data titik 2D yang dihasilkan dapat digunakan dalam pemetaan, pelokalan, dan pemodelan objek/lingkungan. Pada Gambar 2.24 dapat dilihat bentuk fisik dari

RPLIDAR dengan keterangan berupa fitur utama dan nilai tegangannya (RoboPeak Team, 2014).

(RoboPeak Team, 2014) menjelaskan bahwa frekuensi pemindaian RPLIDAR mencapai 5,5 hz saat pengambilan sampel 360 titik setiap putaran dan dapat dikonfigurasi hingga maksimum 10 hz. Hasil *scan* atau pemindaian LiDAR ini berupa titik-titik awan (*point clouds*) dalam bidang 2 dimensi, karena sensor ini hanya memindai bidang horizontal. RPLIDAR berisi sistem pemindai jangkauan dan sistem motor. Setelah menyalakan setiap sub-sistem, RPLIDAR mulai berputar dan memindai searah jarum jam. Pengguna bisa mendapatkan data pemindaian jarak jauh melalui antarmuka komunikasi (Port serial/USB). Sistem mengukur data jarak lebih dari 2000 kali per detik dan dengan output jarak resolusi tinggi (<1% dari jarak).

Dalam dokumentasinya (RoboPeak Team, 2014) menjelaskan bahwa RPLIDAR memancarkan sinyal laser inframerah termodulasi dan sinyal laser tersebut kemudian dipantulkan oleh objek yang akan dideteksi. Sinyal kembali di sampel oleh sistem akuisisi visi di RPLIDAR dan DSP yang tertanam di RPLIDAR mulai memproses data sampel dan nilai jarak keluaran dan nilai sudut antara objek dan RPLIDAR melalui antarmuka komunikasi. Hasil dari pemindaian ini akan dikirimkan melalui komunikasi USART melalui adapter USART dan dihubungkan ke Jetson Nano menggunakan Port USB. Data yang didapat berupa koordinat dari setiap titik laser yang terbaca.

2.2.11.5 Sensor IMU GY-521 MPU-6050 6DOF

Gambar 2.25 Sensor IMU GY-521

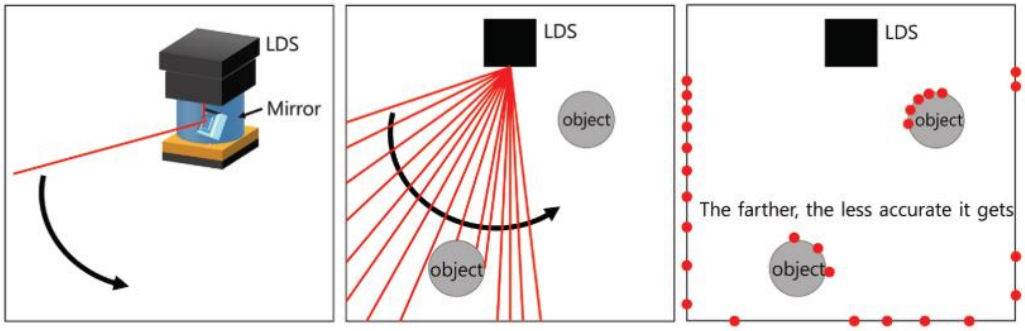
Sumber: (www.digiwarestore.com, 2013)

GY-521 MPU-6050 merupakan modul sensor yang berfungsi sebagai akselerometer dan giroskop yang dikemas dalam bentuk satu modul yang kokoh, seperti yang terlihat pada Gambar 2.25. Menurut (Pyo, dkk., 2017) *Chip* MPU6050 yang terintegrasi dengan *triple-axis gyroscope* dan *triple-axis accelerometer* dalam satu *chip*, dapat digunakan tanpa menambahkan sensor inertial lain. Baca dan tulis mampu melakukannya dengan kecepatan tinggi karena data sensor ditransfer dengan komunikasi I2C atau SPI. Nilai sensor akselerometer dan giroskop dikonversi ke nilai Roll, Pitch, dan Yaw papan dengan

fusi sensor. Saat kelas IMU dibuat sebagai objek dan fungsi update() dipanggil, nilai akselerasi dan *gyro* dibaca secara berkala dari sensor

2.2.12 Light Detection and Ranging

Mengutip dari (Pyo, dkk., 2017) *Laser Distance Sensor* (LDS) disebut dengan berbagai nama seperti *Light Detection and Ranging* (LiDAR), *Laser Range Finder* (LRF) dan *Laser Scanner*. LDS adalah sensor yang digunakan untuk mengukur jarak suatu objek dengan menggunakan laser sebagai sumbernya. Sensor LDS memiliki keunggulan kinerja tinggi, kecepatan tinggi, akuisisi data waktu nyata, sehingga memiliki berbagai aplikasi dalam kaitannya dengan pengukuran jarak. Ini adalah sensor yang banyak digunakan di bidang robot untuk pengenalan objek dan orang, dan sensor jarak berbasis SLAM (sensor berbasis jarak), dan juga banyak digunakan pada kendaraan tidak berawak karena akuisisi data waktu nyata (Pyo, dkk., 2017).



Gambar 2.26 Menghitung Jarak Dengan LDS/LiDAR

(Pyo, dkk., 2017) mengatakan Sensor LDS/LiDAR menghitung perbedaan panjang gelombang ketika sumber laser dipantulkan oleh objek, seperti yang ditunjukkan Gambar 2.26. Jika dianalogikan, ini sama seperti ketika kita mengarahkan cahaya senter ke suatu permukaan. Sebenarnya yang terjadi adalah kita dapat melihat pantulan cahaya senter dari suatu permukaan ke retina, namun karena prosesnya sangat cepat, maka hal tersebut terjadi seolah-olah secara instan. Untuk menghitung jarak, LiDAR menggunakan rumus sebagai berikut:

$$d = \frac{c * t}{2} \tag{2.18}$$

d = jarak antara sensor dan objek yang diukur (m)

c = Kecepatan cahaya (3 x 10^8 m/s)

t = Waktu tempuh cahaya/sinyal

2.2.13 Wheel Odometry

Menurut (Holland, 2004) *odometry* adalah informasi relatif yang memberi tahu kita seberapa banyak kita telah pindah, tetapi bukan di mana kita berada. Menurut (Ben-Ari, dkk., 2018) *Wheel Odometry* merupakan teknik lokalisasi yang menghitung posisi robot berdasarkan perputaran roda (kecepatan putaran roda

atau jumlah perputaran roda). Kecepatan putaran roda atau jumlah perputaran roda dapat diukur menggunakan rotary *encoder*. Jika rotary *encoder* menggunakan tipe *incremental* (menghitung jumlah perputaran roda dalam satuan pulsa / *ticks*), maka perpindahan roda dalam satuan meter menggunakan Persamaan 2.18 dengan i merupakan roda kiri (l) atau roda kanan (r), R adalah jari-jari roda, Δ_{pulsa} adalah perbedaan pulsa pada waktu t dan $t-1$, dan resolusi_sensor adalah jumlah pulsa per revolusi roda (Ben-Ari, dkk., 2018; Putra, dkk., 2020).

Untuk mendapatkan jumlah pulsa setiap satu kali putaran roda digunakan rumus sebagai berikut (Utomo, 2015; Ben-Ari, dkk., 2018):

$$keliling\ roda = 2\,\pi\,r \tag{2.19}$$

$$d_i = \frac{keliling\ roda \times \Delta_{pulsa}}{resolusi\_sensor} \tag{2.20}$$

Menurut (Putra, dkk., 2020), setelah masing-masing perpindahan roda dihitung, perpindahan sudut orientasi robot dapat dihitung menggunakan persamaan 2.21 dengan b adalah jarak antar roda. Perpindahan robot dalam koordinat *x* dapat dihitung menggunakan persamaan 2.22 dan perpindahan robot dalam koordinat *y* dihitung menggunakan persamaan 2.23 dengan variabel *dc* dihitung dengan persamaan 2.24 serta variabel φ merupakan sudut orientasi awal robot (Ben-Ari, dkk., 2018).

$$\theta = (d_r - d_i)/b \tag{2.21}$$

$$dx = d_c\ \cos(\theta + \varphi) \tag{2.22}$$

$$dy = d_c\ \sin(\theta + \varphi) \tag{2.23}$$

$$d_c = (d_r + d_l)/2 \tag{2.24}$$

2.2.14 *Inertial Measurement Unit* (IMU)

Menurut (Siegwart, dkk., 2004) Inertial Measurement Unit (IMU) adalah perangkat yang menggunakan giroskop dan akselerometer untuk memperkirakan posisi relatif, kecepatan, dan percepatan kendaraan yang bergerak. Namun pada penelitian, hanya akselerometer yang dimanfaatkan untuk mendapatkan data inersia dari pergerakan robot, sedangkan giroskop tidak.

2.2.14.1 Akselerometer

Menurut (Siegwart, dkk., 2004) akselerometer adalah perangkat yang digunakan untuk mengukur semua gaya eksternal yang bekerja padanya, termasuk gravitasi. Akselerometer termasuk dalam kelas sensor *proprioceptive*. (Siegwart, dkk., 2004) menambahkan bahwa secara konseptual, akselerometer adalah sistem pegas massa peredam di mana posisi tiga dimensi massa bukti relatif terhadap selubung akselerometer dapat diukur dengan beberapa mekanisme

2.2.15 *Robot Operating System* (ROS)

Mengutip dari buku "ROS *Robot Programming*" (Pyo, dkk., 2017), menjelaskan bahwa ROS *(Robot Operating System)* adalah sistem operasi *meta* yang terbuka untuk umum yang menyediakan layanan seperti sistem operasi, abstraksi perangkat keras, kontrol perangkat tingkat rendah, implementasi fungsionalitas umum, pengiriman pesan antar proses, dan manajemen paket. ROS menyediakan berbagai alat dan *library* untuk dapat membangun, menulis, dan menjalankan kode di banyak komputer. (Pyo, dkk., 2017) menambahkan bahwa sistem operasi meta menggambarkan sebuah sistem yang melakukan proses seperti penjadwalan, pemuatan, pemantauan, dan penanganan kesalahan dengan memanfaatkan lapisan *virtualization* antara aplikasi dan sumber daya komputasi terdistribusi. Pada dasarnya ROS bukanlah sistem operasi seperti Windows, Linux, dan Android, melainkan sebuah sistem operasi yang berjalan pada sistem operasi yang sudah ada, contohnya ROS pada sistem operasi Linux (Pyo, dkk., 2017).

Gambar 2.27 ROS Sebagai Sistem Operasi Meta Pada Sistem Operasi Lain

Sumber: (Pyo, dkk., 2017)

(Pyo, dkk., 2017) menjelaskan bahwa ROS mengembangkan, mengelola, dan menyediakan aplikasi paket untuk berbagai keperluan, dan telah membentuk ekosistem yang mendistribusikan paket yang dikembangkan oleh pengguna. Seperti yang dijelaskan pada Gambar 2.27, ROS adalah sistem pendukung untuk mengendalikan sensor dengan abstraksi perangkat keras dan mengembangkan aplikasi robot berdasarkan sistem operasi konvensional yang telah ada. (Pyo, dkk., 2017) mengatakan untuk mengembangkan robot yang terkait dengan ROS, perlu memahami komponen dan konsep penting dari ROS. Sub bab di bawah ini akan memperkenalkan terminologi yang digunakan dalam ROS dan konsep penting ROS seperti komunikasi pesan, *file* pesan, nama, transformasi koordinat (TF), pustaka klien, komunikasi antar perangkat heterogen, sistem *file*, dan sistem pembangunan.

2.2.15.1 Terminologi Pada Ekosistem ROS

Pada buku "ROS *Robot Programming*" menjelaskan beberapa terminologi-terminologi yang digunakan pada ekosistem ROS sebagai komponen atau konsep penting, berikut ini adalah terminologi yang sering digunakan (Pyo, dkk., 2017).

1. **ROS,** *middleware* yang menyediakan layanan sistem operasi standar seperti abstraksi perangkat keras, *driver* perangkat, implementasi fitur yang umum digunakan termasuk pengindraan, pengenalan, pemetaan, perencanaan gerak, pengiriman pesan antar proses, manajemen paket, visualisasi dan perpustakaan kode untuk pengembangan serta alat *debugging* (Pyo, dkk., 2017).
2. **Master**, sebutan untuk *node* yang bertindak sebagai *node* server dari koneksi *node-to-node* dan komunikasi pesan. Koneksi antara *node* dan komunikasi pesan seperti topik dan layanan tidak mungkin tanpa master. Saat menjalankan ROS, *master* akan dikonfigurasi dengan alamat dan *port* URI dikonfigurasi di ROS_MASTER_URI. Secara *default*, alamat URI menggunakan alamat IP PC lokal, dan nomor *port* 11311, kecuali jika konfigurasi diubah. (Pyo, dkk., 2017).
3. ***Slave*,** sebutan untuk *node* yang terkoneksi dan menerima perintah dari master. Komunikasi Master dengan *slave* menggunakan XML *Remote Procedure Call* yang merupakan protokol berbasis HTTP yang tidak membutuhkan konektivitas (Pyo, dkk., 2017).
4. ***Node,*** merujuk pada unit terkecil dari proses yang berjalan pada ROS, seperti program yang berjalan pada satu program *executable*. ROS merekomendasikan untuk membuat satu *node* tunggal untuk setiap tujuan, dan direkomendasikan untuk dikembangkan agar dapat digunakan kembali dengan mudah. Saat pertama kali dijalankan, *node* mendaftarkan informasi berupa nama, tipe pesan, alamat URI, dan nomor *port*. *Node* yang sudah terdaftar bisa bertindak sebagai *publisher, subscriber,* dan *service* berdasarkan informasi yang terdaftar. Pada implementasinya *node* digunakan sebagai media pertukaran pesan menggunakan *topic* dan *service* (Pyo, dkk., 2017).
5. ***Package,*** unit dasar dari ROS. Aplikasi ROS dikembangkan berdasarkan paket-paket dasar yang berisi konfigurasi untuk meluncurkan *node* lainnya. *Package* berisikan semua *file* penting yang saling terhubung dengan paket lainnya termasuk dependensi *library* ROS untuk menjalankan beberapa proses, *dataset*, dan konfigurasi (Pyo, dkk., 2017).
6. ***Metapackage,*** bagian dari *package* yang memiliki tujuan umum. Sebagai contoh pada *metapackage* "Navigation" yang berisikan 10 paket termasuk AMCL, DWA, EKF, dan map_server (Pyo, dkk., 2017).
7. ***Message,*** variabel yang digunakan untuk mengirim atau menerima pesan di antara *node-node*. *Message* dapat berupa variabel *integer*, *float*, *Boolean* dan *array*. Protokol komunikasi TCPROS dan UDPROS digunakan

untuk pengiriman pesan. Topik digunakan dalam pengiriman pesan searah sementara layanan digunakan dalam pengiriman pesan dua arah yang melibatkan permintaan dan respons (Pyo, dkk., 2017).

8. ***Topic,*** secara harfiah seperti topik di dalam pembicaraan. *Node publisher* mendaftarkan topiknya ke *master* dan kemudian mulai memublikasikan pesan tentang suatu topik. *Node subscriber* yang ingin menerima informasi permintaan topik dari *node publisher* disesuaikan dengan nama topik yang terdaftar di master. Berdasarkan informasi ini, *node subscriber* langsung terhubung ke *node publisher* untuk bertukar pesan sebagai topik. Sehingga dengan konsep seperti ini sistem dapat mendapatkan pesan dari banyak sensor yang digunakan untuk mengambil keputusan menggerakkan aktuator (Pyo, dkk., 2017).
9. ***Publish and Publisher,*** istilah "*publish* atau terbitkan" merujuk pada tindakan mentransmisikan pesan relatif yang sesuai dengan *topic* yang diminta. *Node publisher* mendaftarkan informasi dan topiknya sendiri ke master, kemudian menerbitkan pesan, untuk nantinya di dapat *request* oleh *node subscriber*. Publisher dideklarasikan di dalam *node* dan dapat dideklarasikan beberapa kali dalam satu *node* (Pyo, dkk., 2017).
10. ***Subscribe and Subscriber,*** istilah "*subscribe* atau berlangganan" merujuk pada tindakan menerima pesan relatif yang sesuai dengan topik. *Node subscriber* mendaftarkan informasi dan topiknya sendiri pada *master* dan menerima informasi *publisher* yang menerbitkan topik relatif dari master. Berdasarkan informasi *publisher* yang diterima, *node subscriber* secara langsung meminta koneksi ke *node publisher* yang terhubung. *Subscriber* dideklarasikan dalam *node* dan dapat dideklarasikan beberapa kali dalam satu *node* (Pyo, dkk., 2017).
11. ***Service,*** komunikasi dua arah yang sinkron antara *service client* yang meminta layanan mengenai tugas tertentu dan *service server* yang bertanggung jawab untuk menanggapi permintaan (Pyo, dkk., 2017).
12. ***Service Server,*** server dalam komunikasi pesan layanan yang menerima permintaan sebagai input dan mengirimkan respons sebagai output. Baik permintaan maupun tanggapan adalah berupa pesan. Server layanan diimplementasikan di *node* yang menerima dan menjalankan permintaan yang diberikan (Pyo, dkk., 2017).
13. ***Service Client,*** klien dalam komunikasi pesan layanan yang meminta layanan ke server dan menerima respons sebagai input. Klien mengirimkan permintaan ke server layanan dan menerima respons. *Service client* diimplementasikan di *node* yang meminta perintah tertentu dan menerima hasil (Pyo, dkk., 2017).
14. ***Bag,*** data dari ROS *message* dapat disimpan dan hasil penyimpanan pesan tersebut disebut sebagai *bag*. Pada ROS, *bag* digunakan untuk merekam dan menjalankan ulang pesan untuk menciptakan ulang

lingkungan sebelum perekaman. Fungsi rekam dan jalankan sangat berguna ketika proses pengembangan algoritme dengan beberapa pengulangan ulang program untuk dimodifikasi (Pyo, dkk., 2017).

15. ***Action,*** metode komunikasi pesan lain yang digunakan untuk komunikasi dua arah asinkron. *Action* digunakan ketika dibutuhkan waktu lebih lama untuk merespons perantara hingga hasilnya dikembalikan (Pyo, dkk., 2017).
16. ***Parameter,*** merujuk pada parameter yang dipakai pada *node*. Nilai *default* diatur dalam parameter dan dapat dibaca atau ditulis jika perlu, hal ini sangat berguna ketika nilai yang dikonfigurasi ingin dimodifikasi secara real-time (Pyo, dkk., 2017).
17. ***Catkin,*** merujuk pada *build system* dari ROS. Sistem *build* pada dasarnya menggunakan Cmake (*Cross Platform Make*) yang terdefinisi tiap proses *build* pada *file* 'CMakeLists.txt'. Sistem *build Catkin* memudahkan penggunaan *build* terkait ROS, manajemen paket, dan dependensi di antara paket (Pyo, dkk., 2017).
18. ***URI,*** memiliki kepanjangan *Uniform Resource Identifier* merupakan alamat unik yang mewakili sumber daya di internet. URI adalah salah satu komponen dasar yang memungkinkannya terjadi interaksi dengan internet dan digunakan sebagai pengenal dalam protokol internet (Pyo, dkk., 2017).

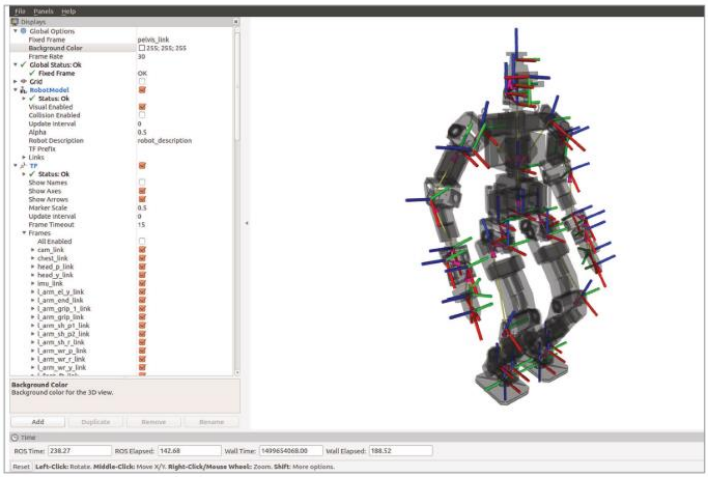
2.2.15.2 Perintah Dasar Pada *ROS*

ROS memiliki beberapa perintah operasi dasar yang sering digunakan pada imbuhan awal setiap perintah yang dilakukan, antara lain sebagai berikut (Pyo, dkk., 2017):

1. ***roscore*****,** merupakan perintah yang menjalankan *master* ROS. Jika beberapa komputer berada di dalam yang sama jaringan, dapat dijalankan dari komputer lain dalam jaringan. Namun, kecuali untuk kasus khusus yang mendukung banyak roscore, hanya satu roscore yang harus berjalan di jaringan. Ketika ROS *master* sedang berjalan, alamat URI dan nomor *port* ditetapkan untuk ROS_MASTER_URI variabel lingkungan yang digunakan. Jika pengguna belum mengatur variabel lingkungan, arus alamat IP lokal digunakan sebagai alamat URI dan nomor *port* 11311 digunakan yang merupakan *port default* nomor untuk *master* (Pyo, dkk., 2017).
2. ***rosrun*****,** merupakan perintah eksekusi dasar pada ROS. Rosrun digunakan untuk menjalankan satu *node* dalam paket. *Node* menggunakan variabel *environment* ROS_HOSTNAME yang disimpan di komputer tempat *node* dijalankan sebagai jalan URI dan *port* disetel ke nilai unik yang berubah-ubah (Pyo, dkk., 2017).

3. ***roslaunch***, sementara rosrun adalah perintah untuk mengeksekusi satu *node*, maka sebaliknya roslaunch digunakan untuk mengeksekusi banyak *node* sekaligus. roslaunch adalah perintah ROS khusus dalam eksekusi *node* dengan fungsi tambahan seperti mengubah parameter paket atau nama *node*, mengkonfirmasi *namespace node*, mengatur ROS_ROOT dan ROS_PACKAGE_PATH, serta mengubah variabel *environment* saat mengeksekusi *node* (Pyo, dkk., 2017).
4. ***rostopic*,** perintah untuk mengakses topik yang ditentukan dalam grafik komputasi (Pyo, dkk., 2017).
5. ***rosparam*,** perintah untuk mendapatkan, mengatur, menghapus parameter dari ROS Parameter Server (Pyo, dkk., 2017).
6. ***rosservice*,** perintah untuk mengakses layanan yang didefinisikan dalam grafik komputasi (Pyo, dkk., 2017).
7. ***rosbeg,*** perintah untuk menyimpan pesan dengan melakukan perekaman dari kondisi lingkungan sebelumnya untuk memudahkan proses pengembangan, sehingga hanya perlu memuat ulang kondisi yang sudah dicapai sebelumnya (Pyo, dkk., 2017).

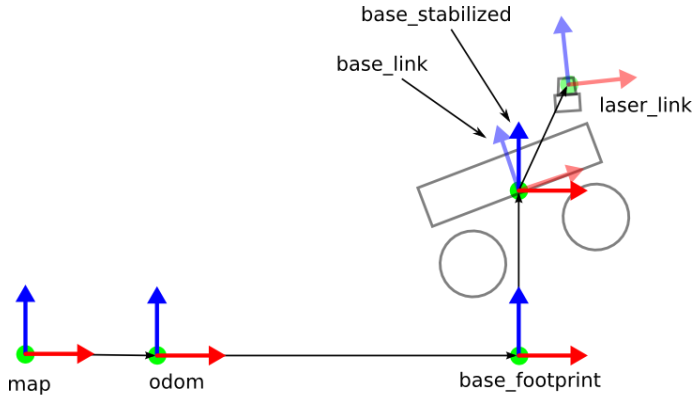
2.2.15.3 *Coordinate Transformation* (TF)



Gambar 2.28 Koordinat Pada Rangkaian Robot

Sumber: (Pyo, dkk., 2017)

Mengutip dari (Pyo, dkk., 2017), pada Gambar 2.28 memperlihatkan pose lengan robot sebagai transformasi koordinat relatif dari masing-masing sendi. Misalnya tangan robot *humanoid* yang terhubung ke pergelangan tangan, pergelangan tangan terhubung ke siku, dan siku terhubung ke bahu. Robot dapat menghitung posisi objek relatif terhadap posisinya berdasarkan data dari TF. Dalam pemrograman robotika, sendiri robot dan posisi masing-masing robot sangat penting, sehingga pada ROS hal ini diwakili oleh TF (*transform*) (Pyo, dkk., 2017).



Gambar 2.29 Frame Transformasi Robot Beroda

Sumber: (wiki.ros.org, 2022)

Pada Gambar 2.29, merupakan transformasi koordinat relatif dari masing-masing titik koordinat pada pose robot beroda. Penelitian ini menggunakan robot beroda sebagai platform dari implementasi sistem pemetaan dan navigasi. Mengutip dari (Pyo, dkk., 2017) dengan menggunakan dua frame di antara frame odom dan base_footprint dapat digunakan sebagai representasi dari pose 2D robot. Frame base_stabilized menambahkan informasi tentang tinggi relatif robot terhadap map atau *odometry*. Frame base_link melekat erat pada robot dan menambahkan sudut roll dan pitch dibandingkan dengan frame base_stabilized (Pyo, dkk., 2017).

(Pyo, dkk., 2017) juga mengatakan bahwa dalam ROS, transformasi koordinat TF adalah salah satu konsep yang paling berguna saat mendeskripsikan bagian robot serta rintangan dan objek. Pose dapat digambarkan sebagai kombinasi posisi dan orientasi. Untuk posisi dinyatakan oleh tiga vektor x, y, dan z sedangkan orientasi dinyatakan oleh empat vektor, yakni vektor x, y, z, dan w atau yang disebut dengan *quaternion*. Bentuk *quaternion* bebas dari *gimbal lock* atau masalah kecepatan yang ada pada vektor roll, pitch, dan yaw metode Euler. Maka dari itu tipe *quaternion* lebih banyak digunakan pada robotika dan ROS menggunakan *quaternion* karena alasan ini.

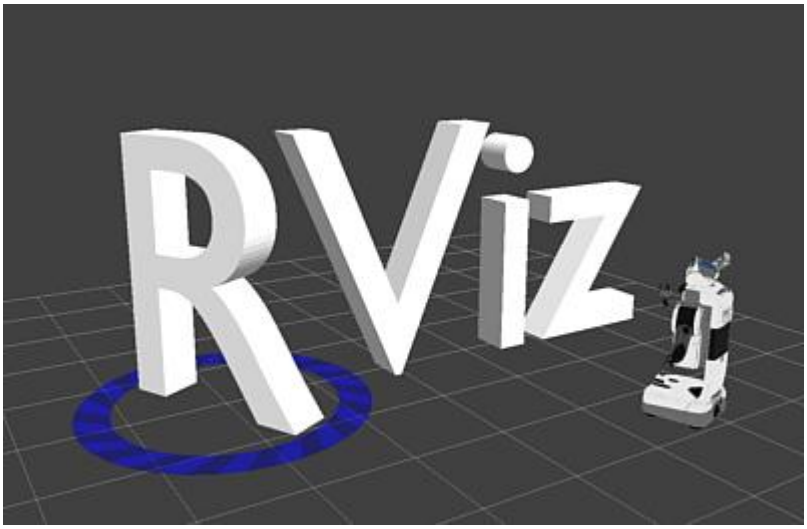
2.2.16 ROS Tools

Mengutip dari (Pyo, dkk., 2017) mengatakan bahwa ada cukup banyak *tool* atau alat pada ROS, termasuk alat yang telah dirilis oleh pengguna ROS secara pribadi. Ini adalah alat tambahan yang sangat berguna untuk pemrograman dengan ROS, salah satunya adalah aplikasi visualisasi 3D yang lengkap dengan alat untuk memberikan koordinat tujuan, yaitu RViz dan aplikasi dengan segudang fitur untuk *development* dan *monitoring*, yaitu rqt.

2.2.16.1 RViz (3D Visualization Tool)

Menurut (Pyo, dkk., 2017) RViz adalah alat visualisasi 3D dari ROS yang tujuan utamanya adalah untuk menampilkan pesan ROS dalam 3D dan memungkinkan untuk memverifikasi data secara visual. Misalnya, untuk memvisualisasikan jarak

dari sensor Laser Distance Sensor (LDS) ke rintangan, Point Cloud Data (PCD) dari sensor jarak 3D seperti *RealSense*, Kinect, atau *Xtion*, nilai gambar yang diperoleh dari kamera, dan banyak lagi tanpa harus mengembangkan perangkat lunak secara terpisah. Tampilan dari RViz dapat dilihat pada Gambar 2.30.



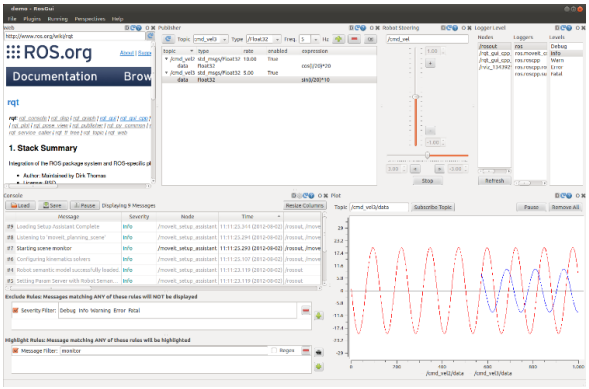
Gambar 2.30 Tampilan *Loading Screen* RViz

Sumber: (github.com/ros-visualization/rviz, 2021)

(Pyo, dkk., 2017) juga menjelaskan bahwa RViz Ini juga mendukung berbagai Penanda Interaktif memungkinkan pengguna untuk melakukan gerakan interaktif dengan perintah dan data yang diterima dari *node* pengguna. Selain itu, ROS memiliki fitur untuk menampilkan visualisasi robot dalam bentuk *Unified Robot Description Format* (URDF), yang dinyatakan sebagai model 3D di mana setiap model dapat dipindahkan atau dioperasikan sesuai dengan derajat kebebasannya, sehingga dapat digunakan untuk simulasi atau kontrol.

2.2.16.2 rqt (ROS GUI Development Tool)

rqt adalah alat yang dapat digunakan sebagai alat GUI yang memiliki banyak fitur untuk keperluan pengembangan. Contohnya seperti *graphical tool* yang menunjukkan hierarki dari setiap *node* sebagai diagram, hierarki ini menunjukkan status *node* dan topik yang saat ini. Kemudian ada alat plot yang mampu membuat skema pesan sebagai grafik 2D dan masih *plugins* lainnya seperti rqt_plugins, rqt_image_view, rqt_graph, rqt_plot, dan rqt_bag seperti yang terlihat pada Gambar 2.31.

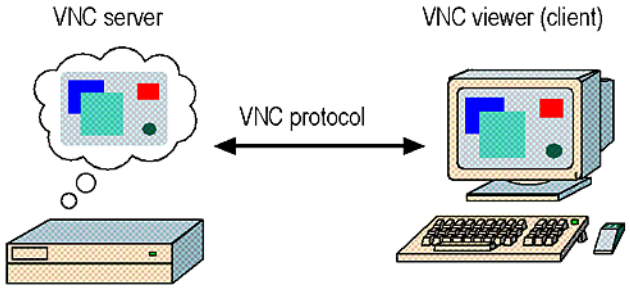


Gambar 2.31 Tampilan Antarmuka Aplikasi rqt Dan Semua *Plugin*

Sumber: (www. ros.guru, 2021)

2.2.17 *Virtual Network Computing* (VNC)

Menurut (Richardson, dkk., 1998) Virtual Network Computing (VNC) atau Komputasi Jaringan Virtual adalah sistem klien ultra tipis berdasarkan protokol tampilan sederhana yang tidak bergantung pada platform. Mesin server tidak hanya menyediakan aplikasi dan data tetapi juga seluruh lingkungan desktop yang dapat diakses dari mesin yang terhubung ke Internet menggunakan perangkat lunak sederhana NC (Richardson, dkk., 1998).



Gambar 2.32 Komunikasi Dua Komputer Dengan VNC

Sumber: (www.computer.org, 1998)

(Richardson, dkk., 1998) mengatakan teknologi yang mendasari sistem VNC adalah protokol sederhana untuk akses jarak jauh ke antarmuka pengguna grafis. Sistem ini bekerja pada tingkat *frame buffer* dan oleh karena itu berlaku untuk semua sistem operasi, sistem *windowing*, dan aplikasi untuk perangkat dengan beberapa bentuk tautan komunikasi. Protokol VNC akan beroperasi melalui semua transportasi yang dapat diandalkan seperti TCP/IP. (Richardson, dkk., 1998) menambahkan bahwa titik akhir di mana perubahan pada *frame buffer* berasal (yaitu, sistem *windowing* dan aplikasi) dikenal sebagai server VNC seperti yang terlihat pada Gambar 2.32.

2.2.18 *Firmware*

Mengutip dari (Pyo, dkk., 2017) dikatakan robot sebagian besar diklasifikasikan ke dalam perangkat keras dan perangkat lunak. Mekanisme, motor, roda gigi, sirkuit, sensor dikategorikan sebagai perangkat keras. (Pyo,

dkk., 2017) menjelaskan bahwa *firmware microcontroller* yang menggerakkan atau mengontrol perangkat keras robot, dan perangkat lunak aplikasi yang membuat peta, menavigasi, membuat gerakan, dan memersepsikan lingkungan berdasarkan data sensor diklasifikasikan sebagai perangkat lunak.

Pada penelitian ini pemrograman *firmware* dibutuhkan untuk mengubah *firmware default* dari *motherboard hoverboard* dengan menggunakan *firmware* yang dapat di *download* pada *link* berikut, https://github.com/EFeru/hoverboard-firmware-hack-FOC. *Firmware* ini ditulis untuk *chip* STM32 dan GD32 yang banyak digunakan sebagai *microcontroller* perangkat mainan *hoverboard*

2.2.19 *Metode Statistik*

Metode statistik merupakan prosedur-prosedur yang digunakan dalam penyajian data yang meliputi pengumpulan, pengorganisasian, peringkasan, dan penyajian data. Metode statistik digunakan untuk mengukur data dari suatu metode untuk mengetahui tingkat keberhasilan pengujiannya.

1. *Weighted Absolute Percentage Error* (WAPE)

   (Chase, 2013; Ragnerstam, 2016) menjelaskan bahwa untuk mengukur kesalahan perkiraan rata-rata di seluruh kelompok item untuk periode tertentu, gunakan metode yang memperhitungkan bobot setiap item dalam hubungannya dengan total. Salah satu metode yang memperhitungkan hal ini adalah WAPE. (Chase, 2013; Ragnerstam, 2016) menggambarkan bahwa rumus dari WAPE adalah seperti yang terlihat pada persamaan 2.25.

$$WAPE = \frac{\sum_{t=1}^{n} |A_t - F_t|}{\sum_{t=1}^{n} |A_t|} \quad (2.25)$$

   (Chase, 2013; Ragnerstam, 2016) mengatakan bahwa WAPE memperhitungkan setiap item dengan menimbang permintaan aktual setiap item dalam kaitannya dengan total permintaan aktual. Dengan demikian, WAPE menghubungkan perbedaan antara nilai aktual dan nilai perkiraan. (Louhichi, dkk., 2012) juga menyebutkan bahwa WAPE adalah metode yang bermanfaat untuk digunakan untuk menghitung akurasi ramalan dan paling optimal ketika variabel mendekati nol.

2. Persentase Keberhasilan

   Persentase keberhasilan merupakan metode untuk menghitung nilai persentase keberhasilan dalam suatu pengujian. Untuk menghitung tingkat keberhasilan akan digunakan persamaan 2.26.

$$\text{Persentase Keberhasilan} = \frac{\text{Keberhasilan}}{\text{Total}} \times 100\% \quad (2.26)$$

BAB 3 METODOLOGI PENELITIAN

Dalam bab metodologi penelitian akan dijelaskan tahapan-tahapan yang akan dilakukan untuk mendukung penelitian dengan judul "Implementasi Pemetaan dan Navigasi Dalam Ruangan Pada Autonomous Mobile Robot Menggunakan Algoritme Hector SLAM Dan Navfn Dengan Sensor RPLIDAR" dan pada bagian metodologi akan memaparkan lebih lanjut terkait tipe penelitian, metode penelitian, subjek penelitian, lokasi penelitian, teknik pengumpulan data, teknis analisis data, dan peralatan pendukung yang digunakan.

3.1 Tipe Penelitian

Tipe penelitian yang digunakan pada penelitian ini adalah tipe implementatif pengembangan. Implementatif menunjukkan bahwa penelitian ini memerlukan pengujian melalui perangkat lunak dan perangkat keras, tidak hanya analisis data saja. Hasil dari penelitian ini adalah sebuah purwarupa *autonomous mobile robot* dari *hoverboard* yang mampu memetakan area sekitar dan mampu menghindari halangan.

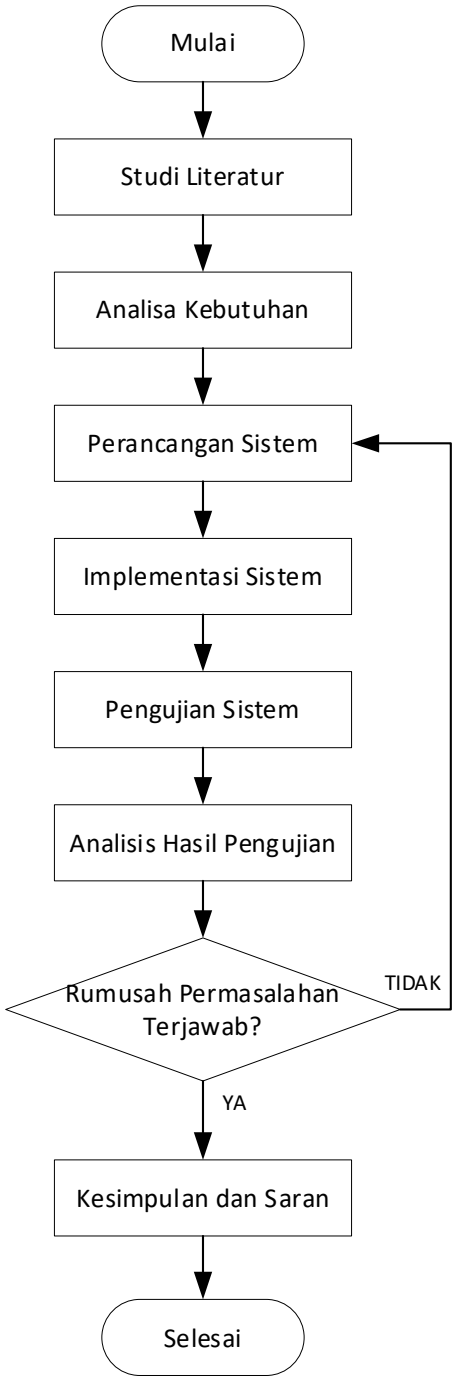
3.1.1 Strategi dan Rencana Penelitian

Hal-Pada penelitian ini, dibutuhkan strategi dan rencana yang jelas agar memenuhi tujuan yang hendak dicapai. Untuk memenuhi hal tersebut penulis akan menjelaskan secara umum setiap perangkat keras dan perangkat lunak yang digunakan sebagai sistem dari robot bergerak. Untuk memenuhi tujuan yang telah dirumuskan pada bab sebelumnya maka disusun perancangan sistem yang menjelaskan peran dari perangkat keras sebagai penyedia sumber pemrosesan untuk melakukan komputasi dan perangkat lunak yang berperan sebagai pengolahan data dan pemrosesan *mapping, localization and navigation*.

Pemecahan masalah pada penelitian ini dilakukan dengan menjalankan 3 tahap, yaitu pengambilan data untuk *mapping*, pengolahan data untuk *localization,* dan hasil pemrosesan peta-lokasi untuk proses *navigation*. Pada tahap awal, pengambilan data berdasarkan perhitungan jarak oleh hasil pembacaan 2 sensor, yakni RPLIDAR dalam bentuk sinar inframerah yang ditembakkan dan *Inertial Measurement Unit* atau IMU. Perhitungan dari kedua sensor ini nantinya akan menghasilkan lokasi robot berupa posisi x, y, z dan benda di sekitarnya seluas 6-6 meter yang menghasilkan bentuk visual pada perangkat lunak yang ditampilkan dalam monitor komputer/Jetson Nano berupa partikel titik merah yang sangat banyak, sehingga membentuk pola berdasarkan area sekitar seperti air yang di wadah maka air akan berbentuk seperti wadah. Kemudian setelah peta sudah dibangun, selanjutnya adalah tahap akhir yaitu navigasi, dengan memanfaatkan sensor RPLIDAR, IMU, peta dan gabungan algoritme navigasi, kita dapat melakukan proses navigasi dengan memasukkan titik tujuan pada visualisasi peta. Dengan begitu robot akan berjalan ke dari titik awal (posisi robot saat ini) menuju titik tujuan (posisi input pengguna.

3.1.2 Metode Penelitian

Demi sepenuhnya mencapai tujuan, maka pada penelitian ini dilakukan secara struktur dan sistematis dalam metode penelitian. Metode yang digunakan untuk mengembangkan suatu karya harus sesuai dengan kebutuhan pembuatan karya. Kebutuhan awal yang diperlukan untuk merancang robot ini yaitu pendalaman informasi terkait permasalahan, pengumpulan, pengkajian teori, dan penelitian sebelumnya yang telah dilakukan dengan tujuan mendukung pengembangan robot ini sehingga dapat ditemukan solusi dari permasalahan dalam pengembangan karya ini. Jurnal ilmiah, artikel ilmiah, situs web, buku, dan lain sebagainya digunakan sebagai studi literatur pada penelitian ini. Setelah melakukan studi literatur, maka langkah selanjutnya adalah melakukan definisi masalah yang ingin diselesaikan, misalnya seperti perangkat lunak (*software*), perangkat keras (*hardware)*, algoritme yang digunakan, dan lain sebagainya. Setelah itu dirumuskan dasar teori yang berisi penelitian secara ilmiah sangat diperlukan sebagai referensi penunjang dalam pengembangan robot ini sehingga menghasilkan karya dari permasalahan yang ada dan bisa dipertanggungjawabkan. Hasil pengkajian teori dari studi literatur yang relevan digunakan dalam mengaplikasikan proses rekayasa kebutuhan sistem yang sesuai lalu melakukan perancangan sistem atau gambaran sistem, implementasi sistem, dan pengujian sistem. Sistem yang telah dilakukan pengujian kemudian dilakukan analisis. Dari hasil analisis pengujian akan digunakan untuk menginformasikan bahwa sistem ini belum atau telah memenuhi kebutuhan dalam permasalahan yang diajukan. Diagram alir tahapan penelitian dapat dilihat pada Gambar 3.1.



Gambar 3.1 Diagram Alir Penelitian

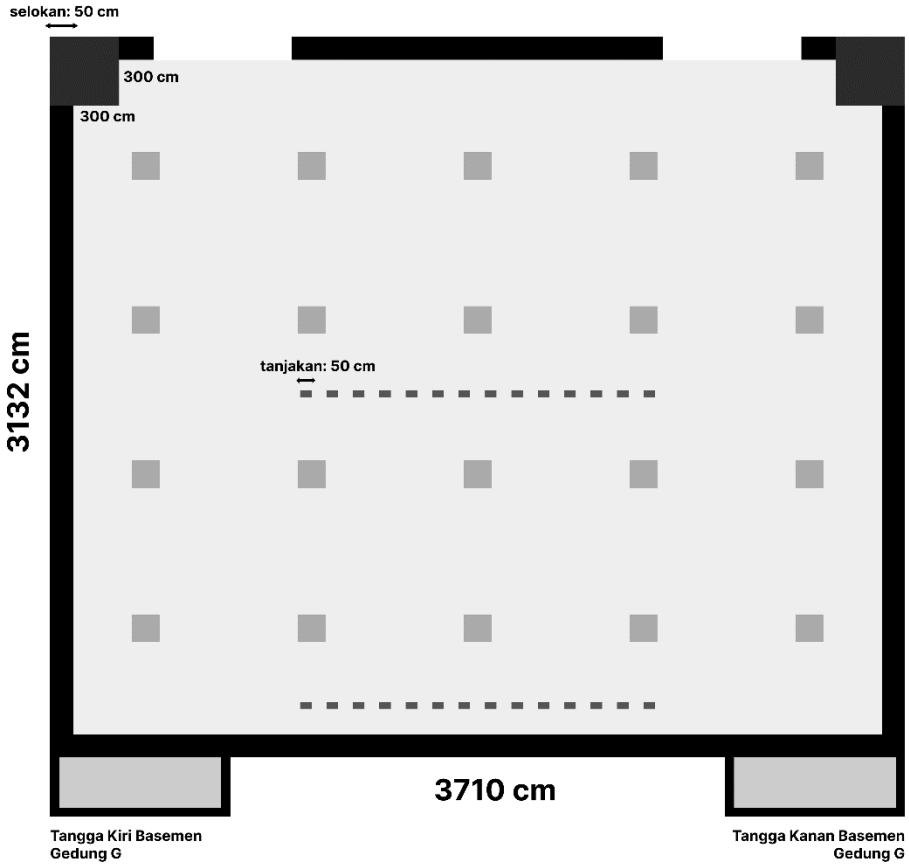
Penggunaan diagram alir penelitian seperti Gambar 3.1 akan membuat proses lebih tepat arah dan teratur. Setelah analisis hasil pengujian apabila penulis merasa bahwa rumusan permasalahan tidak terjawab, maka penulis akan memperbaiki sistem pada perancangan sistem, kemudian melakukan implementasi sistem, pengujian sistem, dan kembali melakukan analisis hasil pengujian. Setelah analisis hasil berhasil menjawab rumusan permasalahan, maka penulis akan mendapatkan kesimpulan dan saran yang dibutuhkan dalam penelitian

3.1.3 Subjek Penelitian

Subjek dari penelitian ini memiliki hubungan dengan sebuah tempat penyimpanan paket pengiriman yang merupakan sebuah gudang ekspedisi yang memiliki tingkat intensitas aktivitas tinggi dalam penyortiran barang dan membutuhkan sistem yang terotomatisasi untuk meningkatkan efisiensi waktu dan tenaga kerja.

3.1.4 Lokasi Penelitian

Untuk lokasi pengujian dilakukan pada bangunan yang tempatnya dapat diukur. Lokasi penelitian dilakukan pada basemen parkir gedung G Fakultas Ilmu Komputer dengan desain denah seperti yang terlihat pada Gambar 3.2.



Gambar 3.2 Denah Basemen Gedung G FILKOM UB

3.2 Teknik Pengumpulan Data

Pengambilan data sensor yang telah dikalibrasi, observasi parameter berdasarkan dokumentasi dan data dari *terminal* Linux digunakan sebagai teknik pengumpulan data dalam penelitian ini. Dengan menggunakan data yang diperoleh dari hasil proses observasi langsung yang dilakukan pada semua sensor, diharapkan penulis mengetahui betul bahwa data semua sensor telah benar berdasarkan pengukuran asli dan dokumentasi *datasheet* setiap sensor. Hal ini dilakukan agar pada pengujian, hasil bisa lebih objektif karena sensor

sudah berjalan sebagaimana mestinya dan menghindari kesalahan sehingga berbeda dari penelitian sebelumnya. Observasi yang dilakukan bertujuan untuk mengetahui apakah sistem yang dibuat berjalan sesuai dengan fungsi dan kebutuhannya. Proses observasi meliputi percobaan setiap parameter pada algoritme yang telah ditentukan dan dicari nilai yang optimal. Pencarian nilai optimal ini didasari pada hasil riset pada penelitian yang serupa. Hal ini dilakukan untuk mengetahui apakah parameter sebagai perhitungan dipengaruhi oleh lingkungan yang dilihat dari hasil *log* pada *terminal* Linux dan hasil visualisasi bentuk peta dan navigasi pada aplikasi RViz yang merupakan salah satu *tool* di dalam ROS. Selanjutnya sampel data hasil observasi digunakan sebagai data untuk proses analisa lebih lanjut

3.3 Teknik Analisis Data

Analisa data dilakukan menggunakan data yang telah dikumpulkan sebelumnya. Hal ini dilakukan untuk menguji kemampuan sistem yang akan menjawab rumusan permasalahan yang ada untuk dievaluasi agar berguna untuk penelitian ke depannya. Berikut adalah aspek yang diuji pada sistem.

1. Pengujian tingkat keakuratan data sensor IMU, Odometer, dan RPLIDAR.
2. Pengujian tingkat ketepatan bentuk peta berdasarkan denah bangunan asli dari hasil pemetaan yang dihasilkan oleh ROS melalui pemindaian dari data sensor RPLIDAR dan IMU.
3. Pengujian tingkat keakuratan navigasi robot terhadap halangan menggunakan algoritme Navfn dan DWA yang diperoleh dari data pemetaan yang telah dibuat.

Pengujian keakuratan pergerakan robot serta waktu komputasi yang dibutuhkan sistem untuk berjalan pada posisi awal menuju posisi yang telah ditentukan yang diperoleh dari masukkan titik koordinat RViz.

3.4 Peralatan Pendukung

Peralatan pendukung diperlukan untuk menunjang penelitian ini agar bisa berjalan dengan lancar dan sistem bisa berjalan sesuai kebutuhan. Peralatan ini terdiri dari perkakas atau peralatan yang berfungsi untuk membantu penulis dalam pembuatan robot. Di bawah ini akan disebutkan peralatan pendukung terkait perangkat keras serta perangkat lunak yang digunakan pada penelitian ini, sebagai berikut.

1. Perangkat keras
    - *Motherboard Hoverboard*
    - Nvidia Jetson Nano
    - Motor Brushless DC dan *encoder*
    - Baterai 10S2P 4000mAh
    - Power bank

- Sensor RPLIDAR
- Sensor IMU GY-521 MPU6050
- *Dongle* Wi-Fi
- *Push button on-off*
- Adapter dan *port charger hoverboard*
- ST-LINK V2 Programmer
- FTDI to mini USB adapter
- Komputer (Personal Komputer)
- *Chassis* papan kayu
- 3D Printer dan filamen
- *Keyboard* dan Mouse
- Kabel *Jumper*
- Kabel *micro* USB

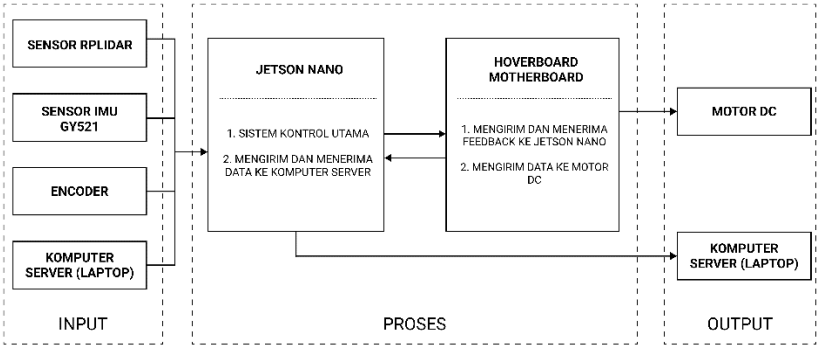
2. Perangkat lunak
    - *Hoverboard Firmware*
    - *STMCube Programmer*
    - *Catkin workspace*
    - *Robot Operating System* (ROS)
    - Terminator
    - Visual Studio Code
    - Platform IO
    - *Virtual Network Computing* (VNC)
    - RViz
    - rqt
    - *Hoverboard driver*
    - RPLIDAR *driver*
    - MPU6050 dan I2C *driver*
    - Hector SLAM *library*
    - Robot localization (EKF & AMCL) *library*
    - *Move base* (Global & *local planner*) *library*

BAB 4 REKAYASA KEBUTUHAN

Dalam bab rekayasa kebutuhan, penulis akan menjelaskan kebutuhan apa saja yang diperlukan untuk mengimplementasikan sistem dalam penelitian ini. Rekayasa kebutuhan mencakup gambaran umum sistem, analisis kebutuhan fungsional dan kebutuhan non-fungsional, kebutuhan perangkat keras, kebutuhan perangkat lunak dan batasan desain sistem. Diharapkan dengan rekayasa kebutuhan ini, sistem yang dibuat bisa berfungsi sesuai dengan tujuan penelitian.

4.1 Gambaran Umum Sistem

Sistem yang dibuat pada penelitian ini merupakan sistem yang mempunyai fungsi untuk membawa barang dari titik awal menuju titik tujuan yang bergerak menggunakan roda serta mendeteksi area sekitar dengan sensor-sensornya. Barang yang dibawa oleh sistem dianalogikan sebagai paket ekspedisi pengiriman asli dengan berat beragam yang nantinya akan dibawa mengelilingi area gudang. Mekanisme yang dilakukan oleh robot dianalogikan sebagai pekerja penyortiran barang. Barang yang disortir berupa paket yang akan dipindahkan dari gudang penyimpanan menuju area distribusi. Setiap informasi yang sudah diselesaikan akan mengirimkan sebuah pesan menuju komputer server. Sesuai yang ada pada Gambar 4.1.



Gambar 4.1 Blok Diagram Sistem

Dari Gambar 4.1 menunjukkan Blok Diagram dari sistem yang memiliki tiga bagian yaitu input, proses, dan output. Bagian input dari sistem akan menggunakan sensor RPLIDAR, IMU dan Odometer. Ketiga sensor ini digabung menjadi satu kesatuan untuk menjalankan proses lokalisasi. Hasil dari sensor RPLIDAR akan digunakan untuk lokalisasi dengan metode AMCL di mana data laser dan map akan digabungkan sehingga menghasilkan perkiraan pemosisian. Untuk sensor IMU dan Odometer akan digabungkan sekaligus menggunakan algoritme EKF sehingga menghasilkan odom_combined yang membuat posisi robot selalu terkoreksi dan terfilter. Sedangkan input dari komputer server berupa perintah-perintah *command line* dan input letak koordinat tujuan melalui RViz atau terminal. Semua bagian input ini akan diteruskan ke bagian proses untuk nantinya diolah oleh Jetson Nano sebagai sistem kontrol utama. Pada

bagian proses sistem menggunakan dua alat yaitu Jetson Nano dan *Hoverboard Motherboard*. Jetson Nano akan menjadi sistem kendali utama dari sistem yang bertugas untuk mencakup seluruh proses pengolahan data sensor, data odometer, dan proses SLAM serta navigasi. Setiap data dari sensor akan dikirimkan ke Jetson Nano yang kemudian akan diproses untuk lokalisasi robot. Hanya sensor RPLIDAR yang mengirimkan data berupa laser scan dengan output yang sudah jadi dan siap digunakan. Jadi tidak perlu melakukan konfigurasi maupun menghitung manual data *raw* sensor. *Hoverboard motherboard* juga mengirimkan data yang sudah jadi berupa *odometry*, data ini sudah diolah di dalam *motherboard*. Hal ini bisa dilakukan karena kita melakukan *custom firmware* dan memproses data odometer menjadi data *odometry* yang bisa langsung digunakan tanpa perlu di filtrasi.

Untuk menggerakkan robot, kita hanya perlu memutar motor BLDC. Untuk memutar motor kita perlu memanipulasi nilai pada variabel cmd_vel. Variabel cmd_vel merupakan variabel yang digunakan untuk *trigger* nilai voltase pada *motherboard hoverboard* sehingga motor dapat berputar, hal ini terjadi karena kode biner yang ditanam ke dalam *chip motherboard*. Untuk memanipulasi cmd_vel kita perlu membuat kode di dalam Jetson Nano. Setelah Jetson Nano menentukan nilai cmd_vel maka nilai cmd_vel akan berubah dari kondisi sebelumnya, hal ini juga berlaku pada variabel cmd_vel *motherboard* karena Jetson Nano saling berkomunikasi dengan *motherboard* melalui *serial* USART. Dengan begitu akan menghasilkan output berupa pergerakan motor, sehingga robot bergerak. Setelah data berubah *motherboard* juga mengirimkan balik nilai Odometer yang terdeteksi ke Jetson Nano untuk mengetahui secara simultan lokasi dari robot. Kemudian Jetson Nano akan mengirimkan semua data pesan yang diterimanya ke dalam RViz, sehingga kita dapat melihat perubahan yang terjadi.

4.2 Analisis Kebutuhan Sistem

Pada subbab terkait analisis kebutuhan sistem, proses ini bertujuan agar peneliti tahu segala kebutuhan yang diperlukan oleh sistem pada perangkat keras dan perangkat lunak, baik secara fungsional dan non-fungsional.

4.2.1 Kebutuhan Fungsional

Kebutuhan fungsional merupakan kebutuhan sistem yang di dalamnya berisi proses apa saja yang harus disediakan oleh sistem, seperti bagaimana sistem merespons input dan bagaimana respons sistem pada kondisi tersebut. Kebutuhan fungsional disajikan ke dalam tabel yang dapat diliat pada Tabel 4.1.

Tabel 4.1 Kebutuhan Fungsional Sistem

No	Kebutuhan Sistem	Skenario Pengujian
1	Sensor RPLIDAR dapat membaca jarak benda sekitar robot dengan pemindaian pada area seluas 360	Dapat mengirimkan data ke *microcontroller* dan pengujian hasil data berupa pengukuran jarak

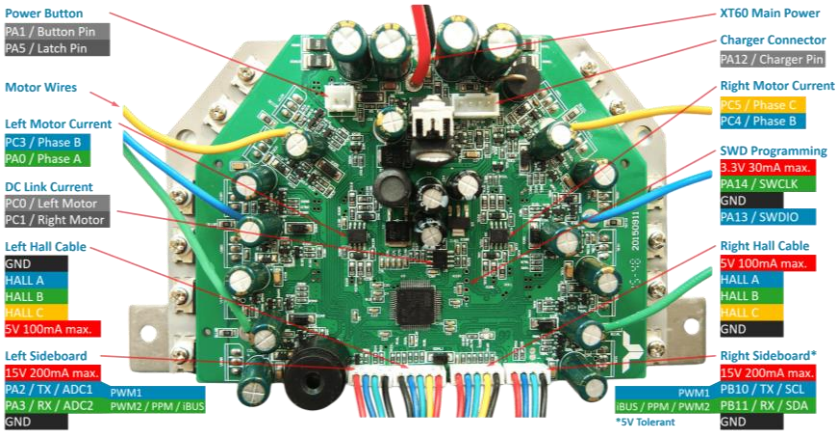
	derajat. Nilai yang dihasilkan berupa nilai *distance* atau jarak dari *range* 0.15 meter hingga 6 meter	antara robot dengan halangan menggunakan alat pengukur sebenarnya seperti meter ukur
2	Sensor Inertial measurement unit (IMU) dapat membaca nilai derajat kemiringan dan gerak akselerasi pada robot. Berfungsi untuk melakukan Sensing proyeksi terhadap kemiringan dan gerak akselerasi robot	Dapat mengirimkan posisi robot pada *microcontroller* terhadap posisi sebenarnya robot menggunakan pemantauan secara langsung dan proyeksi melalui 3D visual RViz
3	Motor DC Brushless atau BLDC dapat menggerakkan robot berdasarkan nilai PWM yang diberikan dari *motherboard hoverboard*	Diujinya jumlah putaran motor DC menggunakan *odometry* dan perhitungan secara manual sehingga dipastikan data *odometry* adalah putaran sebenarnya dan robot bergerak sesuai dengan yang didesain
4	Sistem *microcontroller* di dalam robot mampu mengirim dan menerima data sensor IMU untuk dilaksanakan	Diujinya pengiriman data dan penerimaan data ketika diam dan bergerak
5	Sistem *microcontroller* di dalam robot mampu mengirim dan menerima data sensor RPLIDAR untuk dilaksanakan	Diujinya pengiriman data ketika robot melakukan proses pembuatan peta atau *mapping*
6	*Microcontroller* dapat memproses data dari sensor menjadi kesimpulan arah gerak dan kecepatan sistem pada aplikasi ROS	Diujinya navigasi robot menggunakan data yang diberikan oleh sensor
7	Sistem *Motherboard Hoverboard* dapat menggerakkan kedua motor serta memberi tegangan pada semua komponen elektronik.	Diujinya kecepatan berdasarkan perputaran roda melalui *encoder* motor BLDC menggunakan metode *debugging*
8	Aplikasi RViz dapat memantau pengontrolan sistem secara otomatis ketika bergerak menggunakan sistem otonom *robot operating system*	Diujinya penggunaan RViz untuk melihat hasil pengontrolan otonom secara real-time

4.2.2 Kebutuhan Non-Fungsional

Kebutuhan non-fungsionalitas pada sistem ini terbagi menjadi dua kebutuhan yaitu perangkat keras dan perangkat lunak yang diperlukan untuk menunjang sistem.

4.2.2.1 Kebutuhan Non-Fungsional Perangkat Keras

1. *Motherboard Hoverboard*



Gambar 4.2 Perangkat *Motherboard Hoverboard*

Sumber: (aliexpress.com, 2021)

Pada Gambar 4.2 dapat dilihat pinout dari *motherboard hoverboard* yang memiliki banyak pin di bagian bawah. *Motherboard Hoverboard* digunakan sebagai *driver* dan *controller* pada motor BLDC. *Hoverboard motherboard* menjadi komponen penting yang menyimpan data *firmware* yang telah di *flash* sebelumnya. *Motherboard* merupakan komponen wajib yang ada pada penelitian, ini dikarenakan tanpa adanya *motherboard hoverboard* maka tidak akan bisa mengendalikan motor dan akan sangat sulit jika menggunakan motor BLDC *driver* eksternal karena permasalahan biaya dan kompatibilitas dengan *firmware* yang ada. Berikut adalah spesifikasi yang diambil dari *datasheet motherboard hoverboard* yang ditunjukkan pada Tabel 4.2.

Tabel 4.2 Spesifikasi *Motherboard Hoverboard*

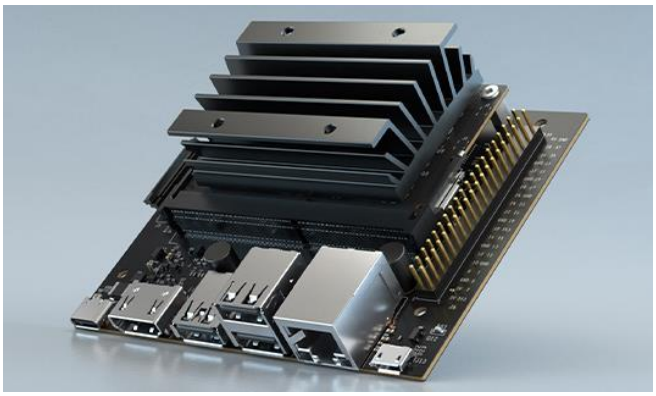
Sumber: (beta.ivc.no dan www.st.com, 2021)

Keterangan	Spesifikasi
Prosesor	GD32F103RCT6 ARM Cortex-M3 32-bit MCU
Flash memory	256 KB sampai 512 KB
SRAM	64 KB
Clock speed	108MHz
Flasher	ST-Link V2 *Programming* Unit

Tegangan operasi	36V
Pengisian daya	42V 2A
Port *Fast* I/O	80 Port (45P, 9ADC, 16PWM, 6RXTX, 4SC, 3.3V, 5V, GND)
Kekuatan putaran maksimal	1000 r.p.m. 1-25A 350W
Tegangan dan arus DC Pin I/O motor	5V 100mA
Tegangan dan arus DC Pin I/O sensor	15V 200mA
Komunikasi	USART, SPI, I2C, I2S, USB

2. Nvidia Jetson Nano

Nvidia Jetson Nano merupakan pemrosesan inti dari sistem pada penelitian ini, fungsinya sebagai tempat pemrosesan input dari ketiga sensor dan menjalankan fungsi mulai dari lokalisasi, pemetaan dan perencanaan jalur serta komunikasi VNC pada slave (komputer). Semua metode, algoritme dan komunikasi yang digunakan akan diproses di dalam sini. Alasan dipilihnya Jetson Nano sebagai *microcomputer* dan pemrosesan utama adalah pertimbangan kemampuan dari perangkat ini yang memiliki CPU Quad-core dengan *clock speed* 1.43 GHz dan RAM sebesar 4 GB LPDDR. Gambar 4.3 merupakan bentuk fisik dari Nvidia Jetson Nano



Gambar 4.3 Nvidia Jetson Nano

Sumber: (www.nvidia.com, 2022)

Dengan spesifikasi seperti ini diharapkan Nvidia Jetson Nano mampu melakukan pemrosesan *mapping* dan *navigation.* Proses pemetaan menggunakan sumber daya yang cukup banyak karena terdapat banyak sistem ROS yang berjalan di belakang layar ditambah dengan sistem perencanaan gerak menggunakan Navfn yang banyak menguras CPU untuk menemukan jalur terpendek. Memori RAM sebesar 4 GB dirasa sudah mumpuni untuk menjalankan semua proses ini. Walaupun pemrosesan pada

RViz dilakukan di Slave atau Komputer, akan tetapi untuk melakukan *load data* hasil pemetaan juga dibutuhkan sumber daya CPU dan RAM yang besar. Oleh karena itu, kesimpulannya adalah Nvidia Jetson Nano mampu menjalankan sistem secara baik pada penelitian ini. Berikut adalah *datasheet* dari *microcontroller* Jetson Nano 3B yang ditunjukkan pada Tabel 4.3.

Tabel 4.3 Spesifikasi *Microcontroller* Jetson Nano 3B

Sumber: (developer.nvidia.com, 2021)

Keterangan	Spesifikasi
CPU	Quad-core ARM A57 @ 1.43 GHz
GPU	NVIDIA Maxwell *architecture with* 128 NVIDIA CUDA® *cores*
Memory	4 GB 64-bit LPDDR4, 1600MHz 25.6 GB/s
Storage	*microSD*
Video Encode	4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 18x 720p @ 30 (H.264/H.265)
Kamera	2x MIPI CSI-2 DPHY
Konektivitas	*Gigabit Ethernet*, M.2 Key E
Display	HDMI 2.0 *and display port*
USB	4x USB 3.0, USB 2.0 Micro-B
Lainnya	GPIO, I2C, I2S, SPI, USART
Mechanical	69.6 mm x 45 mm 260-pin *edge connector*
Memory External	64GB Mikro SD
Power	*Micro* USB 5V 2.5A *and* DC Barrel Jack 5V 4A

3. Motor DC Brushless dan *build-in encoder*

Gambar 4.4 Perangkat Motor BLDC *Hoverboard*

Sumber: (www.jaxlec.top, 2021)

Motor BLDC menggunakan pengontrol loop tertutup elektronik untuk mengalihkan arus DC ke putaran kawat motor yang menghasilkan medan magnet yang secara efektif berputar di ruang angkasa dan yang diikuti oleh rotor magnet permanen. Kontroler menyesuaikan fase dan amplitudo pulsa arus DC untuk mengontrol kecepatan dan torsi motor. Sistem kontrol ini merupakan alternatif dari komutator mekanis (sikat) yang digunakan di banyak motor listrik konvensional. Motor BLDC digunakan sebagai aktuator utama penggerak *autonomous mobile robot*, berikut adalah *datasheet* motor dc *brushless* yang ditunjukkan pada Tabel 4.4 dan bentuk fisik dari motor BLDC *hoverboard* ditunjukkan pada Gambar 4.4.

Tabel 4.4 Spesifikasi Motor DC Brushless

Sumber: (www.aliexpress.com, 2021)

Keterangan	Spesifikasi
Komunikasi	Brushless
Kecepatan	5 - 1500 RPM
Tegangan	36V
Arus	1A - 25A
Power Motor	250 - 350W
Kecepatan Maksimal	26 Km/h
Kecepatan rata-rata	800 RPM
Diameter roda	6.5”

4. Catu Daya

Gambar 4.5 Baterai *Hoverboard*

Sumber: (ubuy.com, 2020)

Catu daya merupakan hal penting dari sistem yang digunakan untuk penyedia sumber tegangan untuk mengaktifkan keseluruhan sistem. Selain itu, catu daya tergolong jenis yang portabel dengan daya tampung energi listrik yang dapat diisi ulang kembali. Untuk penggunaan catu daya harus dapat menyesuaikan tegangannya yang dibutuhkan oleh sistem (Cahyadi, et al., 2016). Ada dua catu daya yang digunakan pada robot ini, pertama yaitu baterai li-ion dengan besar

tegangan 32V dan kuat arus 2A. Catu daya ini digunakan untuk memberikan *power* ke *motherboard hoverboard* dan yang nantinya akan disalurkan lagi ke motor BLDC. Bentuk fisik dari baterai ini dapat dilihat pada Gambar 4.5. Kemudian catu daya yang kedua adalah *power* bank ke Jetson Nano dengan besar tegangan 5V dan kuat arus 1A. Berikut adalah tabel spesifikasi Baterai Li-ion 10S2P dan Power bank yang masing-masing ditunjukkan pada Tabel 4.5 dan Tabel 4.6.

Tabel 4.5 Spesifikasi Catu Daya untuk Suplai *Motherboard Hoverboard*

Sumber: (www. hovertech.co.za, 2021)

Keterangan	Spesifikasi
Tipe Baterai	Litium-Ion
Ukuran Cell	18650 Litium Ion
Tegangan ke *motherboard*	25.2V - 36V
Arus sistem kerja	1A - 20A
Maksimal pemakaian arus	15A
Tegangan Pengisian	43.2V
Kapasitas	4400 mAh
Struktur	10S2P

Tabel 4.6 Spesifikasi Catu Daya untuk Suplai Jetson Nano

Sumber: (www.shopee.co.id, 2021)

Keterangan	Spesifikasi
Tipe Baterai	*Lithium Polymer*
Output 1	USB A DC 5V/3A, 9V/2A, 12V/1,5A
Output 2	USB A DC 5V/3A, 9V/2A, 12V/1,5A
Output 3	*Type* C 5V/3A, 9V/2,33A, 12V/1,7A
Kapasitas	10000 mAh

5. Sensor RPLIDAR A1

Gambar 4.6 Sensor RPLIDAR A1

Sumber: (amazon.com, 2022)

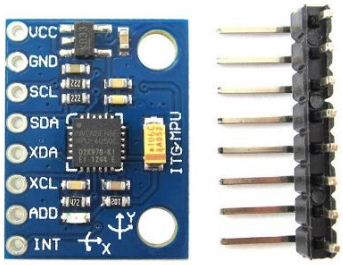
Pada Gambar 4.6 ditunjukkan bentuk fisik dari sensor RPLIDAR yang digunakan pada penelitian ini. Terdiri dari sensor RPLIDAR, kabel *socket*, dan *converter port* lidar ke mikro USB. RPLIDAR A1 digunakan sebagai input utama dari sistem berupa tangkapan jarak dari tembakan laser yang nantinya akan diproses oleh sistem. Penggunaan RPLIDAR didasarkan dari kemampuannya mendeteksi objek seluas 360 derajat dan sejauh 6 meter. Selain itu RPLIDAR juga sangat praktis untuk digunakan karena kita tidak perlu menghitung nilai ataupun kalibrasi data karena RPLIDAR memiliki prosesornya sendiri dan mampu mengirimkan data setengah jadi berupa koordinat-koordinat data jarak. Atas semua pertimbangan dan kemampuannya yang *powerful* maka dari itu RPLIDAR dipilih menjadi sensor utama pada penelitian ini. Berikut adalah spesifikasi sensor RPLIDAR yang ditunjukkan pada Tabel 4.7.

Tabel 4.7 Spesifikasi RPLIDAR A1

Sumber: (www.slamtec.com, 2021)

Keterangan	Spesifikasi
Measuring Range	0.15 – 12 meter
Angular Field of View	0 - 360°
Angular Resolution:	≤1°
Sampling frequency	4000 - 8000 Hz
Rotational Speed	5.5Hz
Laser wavelength	775 – 795nm
Pulse width	110 us
System Voltage	5V
System Current	100mA
Power Consumption	0.5W
Output	USART Serial 3.3 *Voltage level*
Range Resolution	*≤1% of the range （≤12m)* *≤2% of the range （12m～16m)*
Accuracy	*1% of the range （≤3 m）* *2% of the range （3-5 m）* *2.5% of the range （5-25m）*

6. Sensor IMU GY-521



Gambar 4.7 Sensor GY-521 MPU 6050

Sumber: (geekbuying.com, 2022)

Sensor GY-521 memiliki 3 DOF akselerometer dan 3 DOF giroskop yang dapat mendeteksi nilai akselerasi dan gerakan roll, pitch atau yaw pada robot. Sensor GY-521 digunakan sebagai input dari sistem berupa mendeteksi posisi robot secara lokal. Hal ini sangat berguna dalam proses lokalisasi, walaupun proses lokalisasi dapat dilakukan hanya dengan menggunakan data dari *Odometer*, akan tetapi masih perlu dilakukan validasi lebih lanjut lagi menggunakan sensor yang mampu membaca akselerasi robot. Ini dikarenakan terkadang ketika motor BLDC tersangkut atau berputar di tempat maka data yang dikirimkan akan dianggap sedang melakukan suatu gerakan. Tentunya hal ini tidak diinginkan karena dapat membuat robot tidak bergerak semestinya dan posisinya akan berbeda ketika di aplikasi RViz dan keadaan di dunia riil. Maka dari itu atas pertimbangan ini, sensor GY-521 dipilih karena sangat dibutuhkan untuk memvalidasi nilai dari *Odometer* dan membuat robot untuk tetap di posisi semestinya walaupun motor BLDC berputar di tempat. Berikut adalah spesifikasi sensor IMU GY-512 yang ditunjukkan pada Tabel 4.8 dan bentuk fisik dari sensor GY-521 yang digunakan ditunjukkan pada Gambar 4.7.

Tabel 4.8 Spesifikasi Sensor IMU GY-521 MPU6050

Sumber: (www. berrybase.de, 2022)

Keterangan	Spesifikasi
Chipset	MPU-5060
Tegangan sistem kerja	3.3 - 5V DC
Jalur Komunikasi	I2C *up to* 400kHz
Range Gyroscope	250, 500, 1000, 2000/s
Range Gyroscope	2, 4, 6, 8, 16g
Pin	VCC, RX, TX, GND, RST, B0, SCL, SDA

7. Wi-Fi *Dongle*

Wi-Fi *dongle* digunakan sebagai Wi-Fi *extension* karena Jetson Nano tidak memiliki modul Wi-Fi setelah pabrik. Wi-Fi *dongle* akan menjadi komunikasi utama robot dengan komputer server.

8. Tombol *Power On-Off*

Tombol *power* on-off digunakan untuk menyalakan atau mematikan perangkat robot. Tombol ini berupa *push button* yang kondisinya hanya 1 dan 0.

9. *Adapter dan Port Charger Hoverboard*

Adapter charger baterai digunakan untuk mengisi ulang daya dari baterai *hoverboard*. Adapter ini merupakan adapter AC ke DC yang nantinya ditancapkan pada lubang *port* adapter yang terhubung langsung ke *motherboard hoverboard*.

10. ST-LINK V2

ST-LINK bukanlah bagian dari komponen utama robot, alat ini digunakan dalam proses *flashing motherboard hoverboard*. Tepatnya ST-LINK berfungsi sebagai perantara USB *Flashing* antara komputer dengan *motherboard*. *Flashing* dilakukan untuk *reset* semua kode yang ada di *motherboard* sekaligus menuliskan ulang *motherboard* dengan kode baru yang sudah dimodifikasi. Spesifikasi dari ST-LINK yang digunakan pada penelitian ini dapat dilihat pada Tabel 4.9.

Tabel 4.9 *Datasheet* ST-LINK V2 mini USB

Sumber: (www.waveshare.com, 2022)

Keterangan	Spesifikasi
SWD voltage range	1.65V - 3.6V
SWIM *voltage range*	1.65V - 5.5V
Supports SWV	No
Debug interfaces	2
LED indicator	Dual color LED

11. FTDI mini USB F23RL

FTDI merupakan kepanjangan dari Future Technology Devices International yang merupakan nama perusahaan semiconductor yang mengkhususkan diri dalam teknologi Universal Serial Bus. FTDI digunakan untuk komunikasi *serial* Jetson Nano dengan *motherboard hoverboard* menggunakan protokol *serial* USART. FTDI akan terhubung dengan kabel sensor kanan dari *hoverboard* yang memiliki pin RX/TX dan akan ditancapkan dengan *port* USB pada Jetson Nano.

12. Komputer (*Personal* komputer)

Komputer yang dimaksud merupakan sebuah komputer pribadi fleksibel yang dapat dijinjing dengan mudah atau dengan kata lain adalah laptop. Pada penelitian ini, komputer pribadi memiliki peran yang sangat penting dan banyak berkontribusi pada penelitian ini. Personal komputer digunakan sebagai server dan seterusnya penulis akan menggunakan kata komputer server sebagai maksud dari komputer pribadi penulis. Komputer server digunakan sebagai *setting up Master - Slave* pada *robot operating system*. Robot bergerak otonom merupakan *slave* sedangkan komputer berperan sebagai *master*. Fungsi *master* (komputer) adalah menerima semua jenis data *feedback* yang dikirim atau di ekspor dari *slave* (robot) yang kemudian data-data tersebut dapat digunakan untuk proses visualisasi menggunakan aplikasi *third party*.

13. *Chassis* papan kayu

Chassis merupakan rangka penyangga suatu struktur. Rangka penyangga pada robot AMR terbuat dari gabungan papan kayu berukuran 40x40 cm sejumlah 4 buah dengan ketebalan 2 cm yang digunakan sebagai bahan pembuatan *chassis* robot. Papan kayu ini nantinya akan menjadi tempat

penyimpanan komponen utama seperti Jetson Nano dan bagian atasnya berperan sebagai tutup sekaligus alas untuk tempat menaruh barang.

14. Filamen 3D *printer*

Filamen 3D printer sebagai bahan utama tinta 3D printer. *Chassis* robot menggunakan desain *custom* maka dari nantinya akan ada beberapa bagian yang hanya bisa dibuat menggunakan mesin 3D printer. Jenis filamen yang digunakan adalah PETG berwarna hitam sebanyak 2 kilogram.

15. 3D *printer*

3D Printer digunakan untuk mencetak bagian utama robot. Jenis 3D printer yang digunakan pada penelitian ini adalah printer dengan jenis FDM bermerek Creality CR-10S Pro dengan spesifikasi minimal panjang 30 cm, lebar 30 cm dan tinggi 20 cm.

16. Kabel *jumper*

Kabel *jumper* digunakan untuk menghubungkan *serial* USART sensor kanan *motherboard* dengan FTDI yang terhubung pada Jetson Nano melalui USB mini.

17. Kabel *micro* USB

Kabel mikro usb digunakan untuk menghubungkan sensor RPLIDAR denganJetson Nano melalui USB. Selain itu juga digunakan untuk menghubungkan Jetson Nano ke catu daya, yakni Power bank.

4.2.2.2 Kebutuhan Non-Fungsional Perangkat Lunak

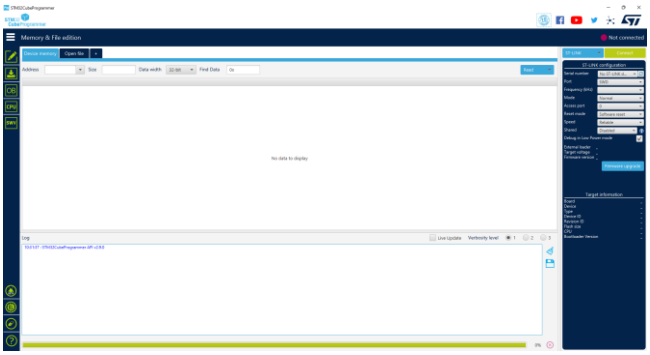
Kebutuhan non-fungsional adalah batasan dari sistem terhadap layanan yang diberikan. Batasan tersebut dapat berupa waktu, batasan proses pengembangan dan batasan terhadap standar tertentu.

1. *Hoverboard* Firmware

Firmware *hoverboard* tersimpan di dalam *chip* GD32 yang menjadi pemrosesan utama *hoverboard*. *Firmware* ini dapat diubah dan dimanipulasi nilainya dengan mengunggah *file* ekstensi biner ke dalam *chip* tersebut. Penelitian ini akan menggunakan *firmware* yang dikembangkan oleh Emanuel Feru dan komunitas *hoverboard* di GitHub pada *link* berikut, https://github.com/EFeru/hoverboard-firmware-hack-FOC.

2. STM32Cube *Programmer*

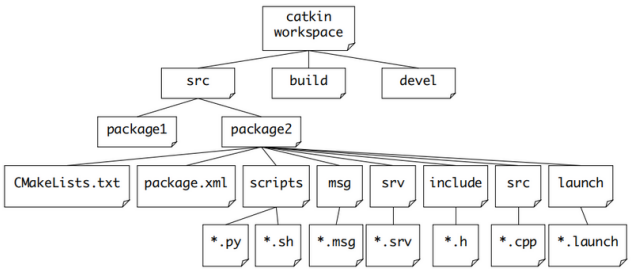
Aplikasi STM32Cube *Programmer* digunakan untuk melakukan *flashing* pada *motherboard hoverboard*. STM32Cube Programmer dikirimkan dalam versi GUI (antarmuka pengguna grafis) dan CLI (antarmuka baris perintah). Gambar 4.8 merupakan tampilan GUI dari aplikasi STM32Cube *Programmer*.



Gambar 4.8 Antarmuka aplikasi STM32Cube Programmer

3. Catkin workspace

Mengutip dari (Pyo, dkk., 2017) mengatakan bahwa ROS menggunakan sistem pembuatan khusus ROS yang disebut *catkin*. *Catkin* mengacu pada sistem *build* ROS yang pada dasarnya menggunakan CMake *(Cross Platform Make)* dan lingkungan *build* dijelaskan dalam *file* 'CMakeLists.txt' di folder paket. CMake dimodifikasi di ROS untuk membuat sistem *build* khusus ROS. Menurut (Pyo, dkk., 2017) *catkin* merupakan salah satu metode *workspace* bertujuan untuk memudahkan proses *developing* dan *compiling*, konsep ini sama dengan konsep *webpack* yang ada pada web *developing*, diagram pohon dari struktur *catkin* seperti yang terdapat pada Gambar 4.9.



Gambar 4.9 Diagram Pohon *Workspace Catkin*

Sumber: (www.medium.com/swlh, 2021)

4. ROS 1 - Melodic Morenia

Gambar 4.10 ROS Melodic

Sumber: (wiki.ros.org, 2018)

Saat ini ROS sudah mempunyai 2 *major version* dengan arsitektur yang berbeda, yaitu ada ROS1 dan ROS2. Pada penelitian ini, penulis menggunakan ROS1 dengan versi Melodic pada *microcomputer* Jetson Nano dan menggunakan versi Noetic pada komputer server. Perbedaan versi ini di karena kan perbedaan versi Ubuntu pada Jetson Nano dan komputer server, akan tetapi ROS mampu melakukan komunikasi antar perangkat walaupun memiliki versi yang berbeda. ROS Melodic Morenia terutama ditargetkan pada rilis Ubuntu 18.04 (Bionic) seperti yang terlihat pada Gambar 4.10.

5. Terminator

Terminator atau *terminal* digunakan sebagai *terminal* perintah utama dalam melakukan penelitian ini, hal ini dikarenakan fiturnya lebih kaya dari pada *terminal default* milik Linux. Kelebihan *terminal* ini adalah mampu membelah atau membagi tampilan tabnya menjadi beberapa bagian. Dikarenakan proses untuk menjalankan *file* terjadi lewat terminal, untuk itu dibutuhkan *terminal* yang mampu membelah diri di dalam satu tampilan *Window.*

6. Visual Studio Code (IDE)

Visual Studio Code atau yang biasa disingkat dengan VSCode merupakan kode editor yang banyak digunakan oleh developer. Kode editor ini digunakan karena kemudahan dalam auto koreksi dan kemudahan navigasi antar *file*. Selain itu fitur *debugging* dalam melakukan *tracking error* akan sangat membantu sekali.

7. Platform IO

Penggunaan *extension* Platform IO ini dikarenakan kemudahannya dalam proses *compiling* dan *flash* ke *motherboard hoverboard*. Dengan cukup melakukan 1 kali klik maka proses *compile* dan *upload* akan sekaligus dilakukan sehingga mempercepat proses pengembangan robot di dalam penelitian ini.

8. Creality Slicer

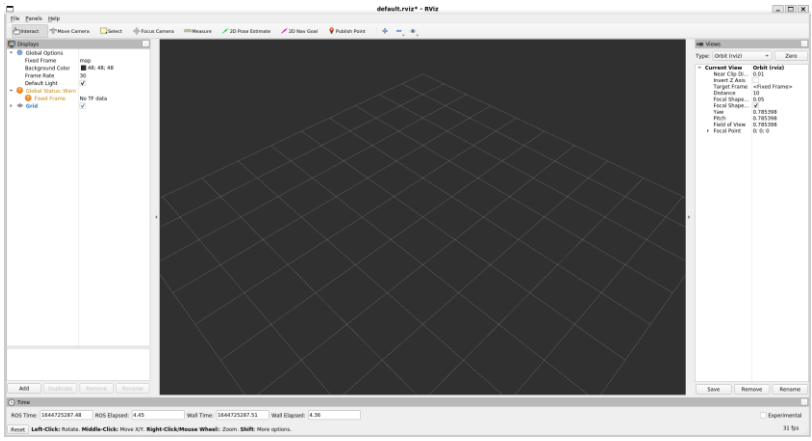
Creality Slicer merupakan perangkat lunak pengiris digunakan dalam pencetakan 3D untuk memotong model 3D menjadi beberapa lapisan (irisan) dan menggambar jalur pencetakan yang dapat diikuti oleh mesin. Creality Slicer adalah alat pengiris milik Creality, yang juga memproduksi banyak printer 3D populer seperti seri Ender 3. Aplikasi ini dipakai sebelum melakukan pencetakan 3D, fungsinya adalah melakukan *convert* dari *file* ekstensi umum menjadi *gcode*, sehingga 3D printer dapat mengenali *file* tersebut

9. VNC *Viewer & VNC Server*

VNC *Viewer* dan VNC Server merupakan software remote-control yang memungkinkan untuk mengontrol komputer lain melalui koneksi *network* (Richardson, dkk., 1998). Komputer server akan di *install* VNC *Viewer* sedangkan Jetson Nano sebagai *microcomputer* akan di *install* VNC Server. Kegunaan perangkat lunak ini adalah untuk melakukan proses yang membutuhkan tampilan user interface pada Jetson Nano.

10. RViz

Menurut (Pyo, dkk., 2017) RViz merupakan aplikasi Linux berbasis GUI yang mendukung visualisasi dari *Robot Operating System* (ROS). Pada penelitian ini RViz digunakan sebagai aplikasi untuk melihat sejauh mana pembuatan peta atau *mapping* bekerja. Hampir setiap pengembangan ROS pasti menggunakan RViz sebagai aplikasi visualisasinya. RViz memiliki fitur pendukung utama untuk melakukan navigasi, yakni 2D *Nav Goal*. (Pyo, dkk., 2017) mengatakan bahwa fitur ini mengirimkan data koordinat pada peta yang dijadikan *fixed* frame, sehingga ketika robot sudah dilengkapi dengan sistem navigasi maka akan terbentuk sebuah *path planning* sehingga robot akan mengikuti arahan dari jalur yang sudah dibuat dan divisualisasikan melalui RViz. Gambar 4.11 merupakan tampilan antar muka aplikasi RViz.



Gambar 4.11 Tampilan Antarmuka RViz

11. Aplikasi rqt

Menurut (Pyo, dkk., 2017) rqt adalah kerangka kerja perangkat lunak ROS yang mengimplementasikan berbagai alat GUI dalam bentuk *plugin*). *Tools* ini masih dapat berjalan dalam metode mandiri tradisional, tetapi rqt memudahkan untuk mengelola semua berbagai jendela di layar pada satu tampilan.

12. *Hoverboard driver*

Hoverboard driver terdiri dari sekumpulan *file* dengan bahasa C++ yang menjembatani antara perangkat lunak dengan *firmware hoverboard*. *Hoverboard driver* berisikan *library-library* yang dibutuhkan untuk mengendalikan *hoverboard* melalui *node* ROS. *Driver* ini tidaklah ditulis dari awal oleh penulis, melainkan menggunakan *driver* dari pengembang lain yang membuat tim untuk fokus mengembangkan *driver hoverboard*, *link* yang digunakan sebagai berikut, https://github.com/alex-makarov/hoverboard-driver.

Dengan menggunakan *driver* ini kita dapat melakukan kontrol pada *hoverboard* dengan mudah, hal ini karena *driver* ini sudah berisikan kode untuk komunikasi *firmware* dengan komputer. *Hoverboard driver* memiliki beberapa parameter pada paketnya yang digunakan sebagai konfigurasi pada robot. Nilai dari parameter ini disesuaikan berdasarkan data nyata pada pengukuran robot

sesungguhnya. Tabel parameter yang digunakan paket *Hoverboard driver* dapat dilihat pada Tabel 4.10.

Tabel 4.10 Parameter *Hoverboard Driver*

Sumber: (wiki.ros.org, 2022)

No	Parameter	Keterangan
1	left_wheel	Nama sambungan roda kiri
2	right_wheel	Nama sambungan roda kanan
3	pose_covariance_diagonal	Diagonal matriks kovarians untuk penerbitan pose *odometry*
4	twist_covariance_diagonal	Diagonal matriks kovarians untuk penerbitan twist *odometry*
5	wheel_separation_multiplier	Ini digunakan untuk menjelaskan perbedaan antara model robot dan robot nyata.
6	cmd_vel_timeout	Periode yang diizinkan (dalam s) diperbolehkan antara dua perintah kecepatan yang berurutan.
7	base_frame_id	Digunakan untuk mengisi child_frame_id dari pesan *Odometry* dan TF
8	enable_odom_tf	Publikasikan ke TF secara langsung atau tidak
9	wheel_radius	Jari-jari roda. Diharapkan mereka semua memiliki ukuran yang sama.
10	wheel_separation	Jarak roda kiri dan kanan.
11	allow_multiple_cmd _vel_publishers	Setel ke true akan mengizinkan lebih dari satu topik masukan ~/cmd_vel. Setel ke false akan menyebabkan pengontrol mengerem jika ada lebih dari satu publisher di ~/cmd_vel.
12	linear/x/min_velocity	Kecepatan linier minimum (dalam m/s).
13	linear/x/max_velocity	Kecepatan linier maksimum (dalam m/s)
14	linear/x/max_acceleration	Percepatan linier maksimum (dalam m/s^2)
15	angular/z/min_velocity	Kecepatan sudut minimum (dalam rad/s). Mengatur ini ke 0,0 akan menonaktifkan rotasi berlawanan arah jarum jam.

16	angular/z/max_velocity	Kecepatan sudut maksimum (dalam rad/s)
17	angular/z/min_acceleration	Percepatan sudut minimum (dalam rad/s^2)

13. RPLIDAR ROS *driver*

Menurut (RoboPeak Team, 2014) RPLIDAR *driver* adalah sekumpulan *library* sebagainya yang menjembatani antara perangkat lunak dengan perangkat RPLIDAR. *Driver* ini diciptakan langsung oleh tim riset dari perusahaan yang menjual produk RPLIDAR A1, yaitu RoboPeak. Untuk melakukan *download driver* bisa di *download* di GitHub resmi RoboPeak, *link* yang digunakan sebagai berikut: https://github.com/robopeak/RPLIDAR_ros. *Driver* ini sudah bisa langsung digunakan tanpa harus melakukan pengaturan yang berarti. Tabel parameter yang digunakan paket RPLIDAR ROS *driver* dapat dilihat pada Tabel 4.11.

Tabel 4.11 Parameter RPLIDAR ROS *Driver*

Sumber: (wiki.ros.org, 2022)

No	**Parameter**	**Keterangan**
1	serial_port	nama *port* serial yang digunakan di sistem Anda.
2	serial_baudrate	kecepatan baud *port* serial
3	frame_id	Frame id untuk perangkat RPLIDAR.
4	inverted	menunjukkan apakah LiDAR dipasang terbalik.
5	angle_compensate	menunjukkan apakah pengemudi perlu melakukan kompensasi sudut.
6	scan_mode	mode pemindaian lidar.

14. MPU6050 dan I2C *driver*

MPU6050 merupakan *driver* sensor yang digunakan untuk menjembatani sensor IMU 6050 dengan *microcomputer* atau *microcontroller*. Sedangkan I2C merupakan *library* untuk komunikasi jalur *serial* I2C antara kontroler dengan sensor IMU. Untuk mendapatkan *driver* ini, kita bisa mengunduh *driver* MPU6050 dan I2C *driver* pada *link* berikut:

- https://github.com/Brazilian-Institute-of-Robotics/mpu6050_driver
- https://github.com/Brazilian-Institute-of-Robotics/i2c_device_ros

Driver ini bertujuan untuk membaca nilai percepatan dari sensor akselerometer, giroskop dan magnetometer. Dengan menggunakan *driver* ini kita mampu mengakses sensor IMU dengan mudah. Tabel parameter yang digunakan paket MPU6050 dan I2C *driver* dapat dilihat pada Tabel 4.12.

Tabel 4.12 Parameter MPU6050 *Driver*

Sumber: (wiki.ros.org, 2022)

No	Parameter	Keterangan
1	bus_uri	I2C Bus URI digunakan untuk berkomunikasi dengan perangkat I2C
2	mpu_address	Alamat I2C dari MPU6050 yang biasanya memiliki alamat default 0x68
3	pub_rate	Frekuensi dalam Hertz yang data IMU diterbitkan
4	frame_id	Bingkai jika pesan IMU
5	axes_offsets	Offset untuk memperbaiki nilai yang salah yang disebabkan oleh misalignment. Nilai ini didapat setelah melakukan kalibrasi
6	ki	Konstanta PID yang digunakan dalam prosedur kalibrasi.
7	kp	Konstanta PID yang digunakan dalam prosedur kalibrasi
8	delta	Proses kalibrasi selesai ketika kesalahan mendekati nol dengan presisi yang ditetapkan oleh delta

15. Hector SLAM *library*

Menurut (Kohlbrecher, dkk., 2014) Hector SLAM merupakan sekumpulan paket fungsi atau *library* yang dipakai untuk membuat pemetaan. *Library* ini bertujuan untuk melakukan pemetaan dan lokalisasi dengan mengandalkan data dari laser. *Library* ini memiliki kemampuan melakukan *mapping* tanpa harus mengetahui kondisi sekitarnya terlebih dahulu (Kohlbrecher, dkk., 2011). Fungsi-fungsi pada paket ini tersedia pada *directory* GitHub pengembangnya yaitu *Technical University* Darmstats, *link* yang digunakan sebagai berikut: https://github.com/tu-darmstadt-ros-pkg/hector_slam. Hector SLAM *library* memiliki dari beberapa *library* lagi di dalamnya, terdiri dari *library* hector *mapping*, hector Geotiff, hector Trajectory, hector SLAM dan masih banyak lagi. Tabel parameter yang digunakan paket robot pose EKF dapat dilihat pada Tabel 4.13.

Tabel 4.13 Parameter Hector *Mapping*

Sumber: (wiki.ros.org, 2022)

No	Parameter	Keterangan
1	base_frame	Nama rangka dasar robot. Ini adalah frame yang digunakan untuk pelokalan dan transformasi data pemindaian laser.

2	map_frame	Nama dari frame peta
3	odom_frame	Nama dari frame *odometry*
4	map_resolution	Resolusi peta [m]. Ini adalah panjang tepi sel *grid*.
5	map_size	Ukuran [jumlah sel per sumbu] peta. Peta berbentuk persegi dan memiliki sel kisi (map_size * map_size).
6	map_start_x	Lokasi asal [0.0, 1.0] bingkai /peta pada sumbu x relatif terhadap peta kisi. 0,5 di tengah.
7	map_start_y	Lokasi asal [0.0, 1.0] bingkai /peta pada sumbu y relatif terhadap peta kisi. 0,5 di tengah.
8	map_update _distance_thresh	Ambang batas untuk melakukan pembaruan peta.
9	map_update _angle_thresh	Ambang batas untuk melakukan pembaruan peta.
10	map_multi_res_levels	Jumlah tingkat petak multi-resolusi.
11	update_factor_free	Pengubah pembaruan peta untuk pembaruan sel gratis dalam perkiraan.
12	update_factor_occupied	Pengubah pembaruan peta untuk pembaruan sel yang ditempati dalam perkiraan
13	laser_z_min_value	Tinggi minimum [m] relatif terhadap bingkai pemindai laser untuk titik akhir pemindaian laser yang akan digunakan oleh sistem
14	laser_z_max_value	Tinggi maksimum [m] relatif terhadap bingkai pemindai laser untuk titik akhir pemindaian laser yang akan digunakan oleh sistem
15	pub_map_odom_transform	Tentukan apakah transformasi map->odom harus dipublikasikan oleh sistem.
16	tf_map_scanmatch _transform_frame_name	Nama frame saat memublikasikan scanmatcher ke transformasi peta seperti yang dijelaskan dalam parameter sebelumnya.

16. Robot Pose EKF Localization library

Menurut (Siegwart, dkk., 2004) algoritme EKF adalah implementasi dari filter Kalman yang diperluas. *Library* termasuk ke dalam paket eksternal dari ROS yang dapat langsung di *install* menggunakan perintah $ sudo apt install ros-melodic-robot-pose-ekf. *Library* EKF digunakan baik dalam proses *mapping* ataupun navigasi, hal ini dikarenakan kedua proses itu membutuhkan metode untuk menggabungkan sensor agar peta yang dihasilkan menjadi lebih baik atau yang disebut dengan teknik *fuse* sensor (Pyo, dkk., 2017). Output dari paket ini adalah sebuah frame bernama odom_combined yang sudah dilakukan koreksi nilai kesalahan data *odometry* dengan menggunakan data IMU. Tabel parameter yang digunakan paket robot pose EKF dapat dilihat pada Tabel 4.14.

Tabel 4.14 Parameter EKF Localization

Sumber: (wiki.ros.org, 2022)

No	Parameter	Keterangan
1	output_frame	Nama frame dari *odometry*, dalam hal ini adalah /odom
2	base_footprint_frame	Nama frame dari base footprint
3	freq	Frekuensi bernilai nyata dalam Hz, di mana filter menghasilkan perkiraan keadaan
4	sensor_timeout	Ketika sensor berhenti mengirimkan informasi ke filter, berapa lama filter harus menunggu sebelum melanjutkan tanpa sensor itu.
5	odom_used	Aktifkan atau nonaktifkan input sensor *odometry*
6	imu_used	Aktifkan atau nonaktifkan input sensor IMU
7	vo_used	Aktifkan atau nonaktifkan input sensor visual *odometry*
8	gps_used	Aktifkan atau nonaktifkan input sensor GPS
9	debug	Parameter yang menentukan apakah akan dijalankan dalam mode debug atau tidak.

17. ROS Navigation Stack: AMCL library

Menurut (Pyo, dkk., 2017) *ROS navigation library* merupakan sekumpulan paket fungsi yang dipakai untuk proses *path planning* robot *Library* termasuk ke dalam paket eksternal dari ROS yang dapat langsung di *install* menggunakan perintah $ sudo apt install ros-melodic-navigation. Sedangkan AMCL atau *Adaptive Monte Carlo Localization* adalah sistem lokalisasi probabilistik untuk robot yang bergerak dalam 2D. Ini menerapkan pendekatan lokalisasi Monte Carlo adaptif yang menggunakan filter partikel untuk melacak pose robot terhadap peta yang diketahui (Pyo, dkk., 2017). AMCL digunakan pada lokalisasi

proses navigasi robot, yaitu dengan memanfaatkan data lidar dan hasil pemetaan untuk mendapatkan lokasi dan pose terkini dari robot. Tabel parameter yang digunakan paket AMCL dapat dilihat pada Tabel 4.15.

Tabel 4.15 Parameter AMCL *Localization*

Sumber: (wiki.ros.org, 2022)

No	Parameter	Keterangan
1	odom_frame_id	Frame mana yang digunakan untuk *odometry*.
2	odom_model_type	Model mana yang akan digunakan, baik "diff", "omni", "diff-corrected" atau "omni-corrected".
3	base_frame_id	Bingkai mana yang digunakan untuk basis robot
4	update_min_d	Jumlah partikel minimum yang diizinkan
5	update_min_a	Jumlah partikel maksimum yang diizinkan
6	global_frame_id	Nama kerangka koordinat yang diterbitkan oleh sistem lokalisasi
7	tf_broadcast	Setel ini ke false untuk mencegah amcl memublikasikan transformasi antara frame global dan frame *odometry*.
8	initial_pose_x	Mean pose awal (x), digunakan untuk menginisialisasi filter dengan distribusi Gaussian.
9	initial_pose_y	Pose awal mean (y), digunakan untuk menginisialisasi filter dengan distribusi Gaussian.
10	initial_pose_a	Pose mean awal (yaw), digunakan untuk menginisialisasi filter dengan distribusi Gaussian.

18. ROS Navigation Stack: Move base library

Menurut (Pyo, dkk., 2017) di dalam paket ROS *navigation* stack terdapat paket *move base* yang merupakan paket utama untuk mengirimkan pesan *twist* yang nantinya di konversi. Di dalam paket ini juga terdapat *base* global *planner* untuk *path planning* secara global, DWA *local planner* untuk *path planning local* serta dan masih banyak lagi (Pyo, dkk., 2017). Semua ini tentunya berisikan fungsi-fungsi yang bertujuan untuk menunjang sistem navigasi robot. Paket *move base* melakukan *subscribe* pada beberapa topik antara lain, move_base/*goal* yang merupakan sebuah koordinat tujuan untuk dicapai robot dan move_base/*cancel* yang merupakan permintaan untuk membatalkan tujuan tertentu. Output dari paket ini adalah sebuah *topic* yang digunakan untuk

menggerakkan robot, yaitu cmd_vel. Parameter yang digunakan pada paket ini dapat dilihat pada Tabel 4.16

Tabel 4.16 Parameter *Move Base*

Sumber: (wiki.ros.org, 2022)

No	Parameter	Keterangan
1	base_global_planner	Nama plugin untuk perencana global untuk digunakan dengan move_base
2	base_local_planner	Nama plugin untuk perencana lokal untuk digunakan dengan move_base
3	recovery_behaviors	Daftar plugin perilaku pemulihan untuk digunakan dengan move_base
4	controller_frequency	Laju dalam Hz untuk menjalankan loop kontrol dan mengirim perintah kecepatan ke pangkalan.
5	planner_patience	Berapa lama perencana akan menunggu dalam hitungan detik dalam upaya untuk menemukan rencana yang valid sebelum operasi pembersihan ruang dilakukan.
6	controller_patience	Berapa lama pengontrol akan menunggu dalam hitungan detik tanpa menerima kontrol yang valid sebelum operasi pembersihan ruang dilakukan.
7	conservative_reset_dist	Jarak dari robot dalam meter di mana rintangan akan dibersihkan dari peta biaya saat mencoba mengosongkan ruang di peta
8	recovery_behavior_enabled	Apakah akan mengaktifkan perilaku pemulihan move_base atau tidak untuk mencoba mengosongkan ruang
9	clearing_rotation_allowed	Menentukan apakah robot akan mencoba melakukan rotasi di tempat atau tidak saat mencoba mengosongkan ruang
10	shutdown_costmaps	Menentukan apakah akan mematikan peta biaya *node* atau tidak saat move_base dalam keadaan tidak aktif
11	oscillation_timeout	Berapa lama dalam detik untuk memungkinkan osilasi sebelum menjalankan perilaku pemulihan
12	oscillation_distance	Berapa jauh dalam meter robot harus

		bergerak agar dianggap tidak berosilasi
13	planner_frequency	Laju dalam Hz untuk menjalankan loop perencanaan global
14	max_planning_retries	Berapa kali untuk memungkinkan perencanaan ulang sebelum menjalankan perilaku pemulihan

Parameter paket *move base* hanya berisikan variabel untuk menentukan *plugin* dari perencanaan jalur global dan lokal. Terdapat beberapa parameter penting yang menjadi konfigurasi pada proses navigasi. Pada dokumentasi ROS hal ini disebut dengan *Component APIs. Node* move_base berisi komponen yang memiliki ROS API sendiri. Komponen ini dapat bervariasi berdasarkan nilai masing-masing yang terdiri dari global *costmap*, *local costmap, global planner*, *local planner*, dan *recovery behaviors*. Global *costmap* adalah semua yang diketahui robot dari kunjungan sebelumnya dan pengetahuan yang tersimpan misalnya peta. Sedangkan *Local costmap* adalah segala sesuatu yang dapat diketahui dari posisi saat ini dengan sensor yang tepat. Misalnya. orang berjalan dan benda bergerak lainnya, serta setiap dinding yang dapat dilihat.

Parameter yang ada pada *costmap* memiliki kesamaan hanya berbeda di penggunaan dan topik yang akan di *subscribe*, maka dari itu penjelasan mengenai parameter global *costmap* dan *local costmap* akan dijadikan satu tabel saja. Berikut ini akan dijelaskan parameter-parameter apa saja yang ada pada setiap komponen yang sudah disebutkan sebelumnya. Untuk parameter komponen 2D *costmap* dapat dilihat pada Tabel 4.17, sedangkan untuk komponen global *planner* dapat dilihat pada Tabel 4.18 dan *local planner* Tabel 4.19.

Tabel 4.17 Parameter Konfigurasi 2D *Costmap*

Sumber: (navigation.ros.org, 2022)

No	**Parameter**	**Keterangan**
1	always_send_full_costmap	Apakah akan *costmap* secara lengkap setiap pembaruan, bukan pembaruan.
2	footprint_padding	Jumlah alas pad footprint (m).
3	footprint	Kumpulan titik jejak yang dipesan yang diteruskan sebagai string, harus disetel tertutup
4	global_frame	Frame referensi global
5	height	Tinggi *costmap*.
6	width	Lebar *costmap*
7	lethal_cost_threshold	Biaya minimum occupancy *grid* dianggap sebagai hambatan yang mematikan.

8	map_topic	Topik peta dari map_server atau SLAM.
9	observation_sources	Daftar sumber sensor sebagai string, untuk digunakan jika tidak ditentukan dalam konfigurasi khusus plugin
10	origin_x	X asal peta biaya relatif terhadap lebar (m).
11	origin_y	Y asal peta biaya relatif terhadap tinggi (m).
12	publish_frequency	Frekuensi untuk mempublikasikan *costmap* ke topik.
13	resolution	Resolusi 1 piksel peta biaya, dalam meter.
14	robot_base_frame	Frame referensi robot base
15	robot_radius	Jari-jari robot untuk digunakan, jika koordinat tapak kaki tidak disediakan.
16	rolling_window	Apakah peta biaya harus digulung dengan bingkai dasar robot.
17	track_unknown_space	Jika salah, perlakukan ruang yang tidak diketahui sebagai ruang kosong, yang lain sebagai ruang yang tidak diketahui.
18	transform_tolerance	Toleransi transformasi TF.
19	trinary_costmap	ika peta kisi hunian harus ditafsirkan sebagai hanya 3 nilai (bebas, terisi, tidak diketahui) atau dengan nilai yang tersimpan.
20	unknown_cost_value	Biaya ruang yang tidak diketahui jika melacaknya.
21	update_frequency	Frekuensi pembaruan peta biaya.
22	use_maximum	apakah saat menggabungkan peta biaya untuk menggunakan biaya maksimum
23	plugins	Daftar nama plugin yang ditujukan untuk ruang nama dan nama parameter
24	filters	Daftar nama filter peta biaya yang ditujukan untuk ruang nama dan nama parameter.
25	plugins/static_layer	Plugin untuk *costmap* static *layer* seperti peta
26	plugins/ obstacle_layer	Plugin untuk *costmap obstacle layer*

		seperti data laser
27	plugins/ inflation_layer	Plugin untuk melipat gandakan ukuran *costmap*

Tabel 4.18 Parameter Konfigurasi Navfn *Global Planner*

Sumber: (wiki.ros.org, 2022)

No	Parameter	Keterangan
1	allow_unknown	Menentukan apakah akan mengizinkan perencana membuat rencana yang melintasi ruang yang tidak diketahui atau tidak
2	planner_window_x	Menentukan ukuran x dari jendela opsional untuk membatasi perencana. Ini berguna untuk membatasi NavFn agar berfungsi di jendela kecil dari besar *costmap*
3	planner_window_y	Menentukan ukuran y dari jendela opsional untuk membatasi perencana. Ini dapat berguna untuk membatasi NavFn agar berfungsi di jendela kecil dari besar *costmap*
4	default_tolerance	Sebuah toleransi pada titik tujuan untuk perencana. *Path planner* akan mencoba membuat rencana yang sedekat mungkin dengan tujuan yang ditentukan tetapi tidak lebih jauh dari default_tolerance.
5	visualize_potential	Menentukan apakah akan memvisualisasikan area potensial yang dihitung melalui PointCloud2 atau tidak.

Tabel 4.19 Konfigurasi DWA *Local Planner*

Sumber: (wiki.ros.org, 2022)

No	Parameter	Keterangan
1	acc_lim_x	Batas percepatan x robot dalam meter/detik^2
2	acc_lim_y	Batas percepatan y robot dalam meter/detik^2
3	acc_lim_th	batas percepatan rotasi robot dalam radian/detik^2
4	max_vel_trans	Nilai absolut dari kecepatan translasi maksimum untuk robot dalam m/s

5	min_vel_trans	Nilai absolut dari kecepatan translasi minimum untuk robot dalam m/s
6	max_vel_x	Kecepatan x maksimum untuk robot dalam m/s.
7	min_vel_x	Kecepatan x minimum untuk robot dalam m/s, negatif untuk gerak mundur.
8	max_vel_y	Kecepatan y maksimum untuk robot dalam m/s
9	min_vel_y	Kecepatan y minimum untuk robot dalam m/s
10	max_rot_vel	Nilai mutlak kecepatan putar maksimum robot dalam rad/s
11	min_rot_vel	Nilai mutlak kecepatan putar minimum robot dalam rad/s
12	yaw_goal_tolerance	Toleransi dalam radian untuk pengontrol dalam yaw/rotasi saat mencapai tujuannya
13	xy_goal_tolerance	Toleransi dalam meter untuk pengontrol dalam jarak x & y saat mencapai tujuan
14	latch_xy_goal_ tolerance	Jika toleransi tujuan terkunci, jika robot pernah mencapai lokasi tujuan xy, ia hanya akan berputar di tempatnya, bahkan jika itu berakhir di luar toleransi tujuan saat melakukannya.
15	sim_time	Parameter untuk simulasi. Jumlah waktu untuk meneruskan simulasi lintasan dalam hitungan detik
16	sim_granularity	Parameter untuk simulasi. Ukuran langkah, dalam meter, untuk mengambil antara titik pada lintasan tertentu
17	vx_samples	Parameter untuk simulasi. Jumlah sampel yang akan digunakan saat menjelajahi ruang kecepatan x
18	vy_samples	Parameter untuk simulasi. Jumlah sampel yang akan digunakan saat menjelajahi ruang kecepatan y
19	vth_samples	Parameter untuk simulasi. Jumlah sampel yang akan digunakan saat menjelajahi ruang kecepatan theta
20	controller_frequency	Parameter untuk simulasi. Frekuensi di mana

		pengontrol ini akan dipanggil dalam Hz.
21	path_distance_bias	Pembobotan untuk seberapa banyak pengontrol harus tetap dekat dengan jalur yang diberikan
22	goal_distance_bias	dia menimbang berapa banyak pengontrol harus berusaha mencapai tujuan lokalnya, juga mengontrol kecepatan
23	occdist_scale	Pembobotan untuk seberapa banyak pengontrol harus berusaha menghindari rintangan
24	forward_point_distance	Jarak dari titik pusat robot untuk menempatkan titik penilaian tambahan, dalam meter
25	stop_time_buffer	Jumlah waktu robot harus berhenti sebelum tabrakan agar lintasan dianggap valid dalam hitungan detik
26	scaling_speed	Nilai absolut dari kecepatan untuk mulai menskalakan jejak robot, dalam m/s
27	max_scaling_factor	Faktor maksimum untuk menskalakan jejak robot
28	publish_cost_*grid*	Apakah akan mempublikasikan kisi biaya yang akan digunakan perencana saat merencanakan atau tidak
29	oscillation_reset_dist	Seberapa jauh robot harus berjalan dalam meter sebelum tanda osilasi direset
30	prune_plan	Menentukan apakah akan memakan rencana saat robot bergerak di sepanjang jalan.