

BAB 5 PERANCANGAN DAN IMPLEMENTASI

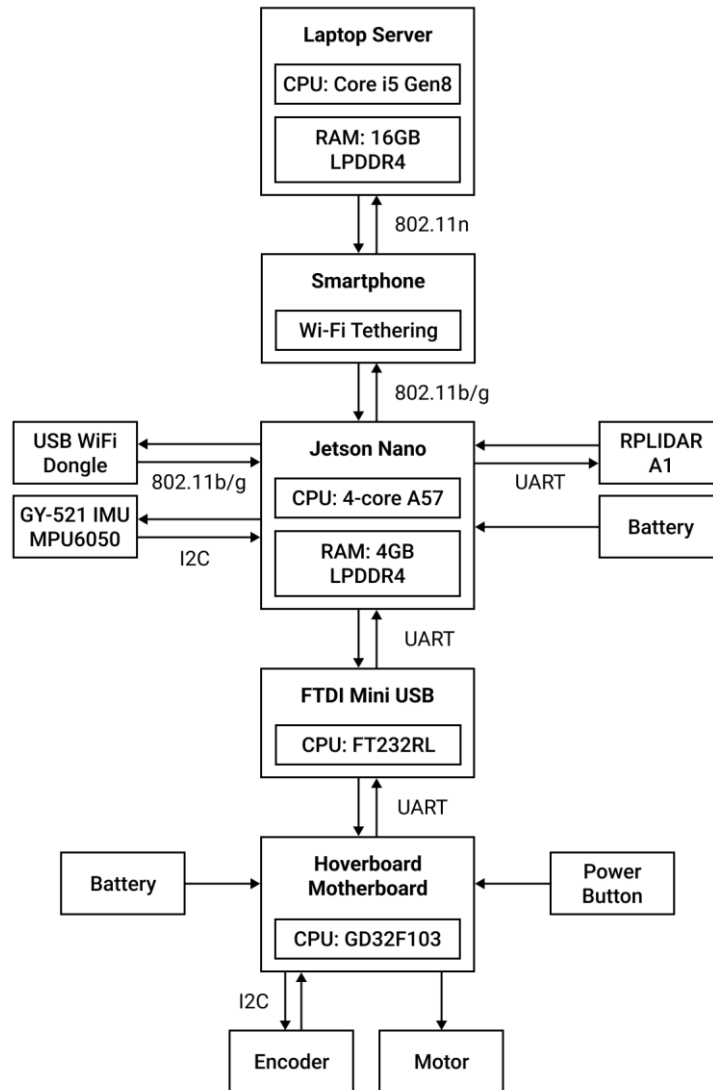
Dalam bab perancangan dan implementasi, penulis menjabarkan tahapan perancangan dari sistem secara skematis hingga pengimplementasiannya secara riil. Perancangan mencakup perancangan secara fisik yang berupa *assembling*, *wiring*, dan perancangan perangkat lunak dengan arsitektur sistem robot otonom. Sementara itu implementasi terfokus pada aspek bagaimana sistem tersebut berhasil dibentuk di dunia riil dan bagaimana pemrograman sistem ini bekerja. Hasil dari perancangan sistem kemudian direalisasikan dan dijelaskan pada implementasi sistem. Oleh karena itu disusunlah proses bagaimana merancang sistem dan mengimplementasikannya sebagai berikut.

5.1 Perancangan Sistem

Perancangan sistem bertujuan untuk memahami cara kerja serta alur kerja sistem, seperti apa yang nantinya dibuat sehingga dalam penjelasannya dalam perancangan sistem nantinya lebih terstruktur dan lebih sistematis. Perancangan sistem dibagi menjadi tiga bagian yaitu perancangan skematis arsitektur sistem, perancangan perangkat keras, dan perancangan perangkat lunak.

5.1.1 Perancangan Perangkat Keras Sistem

Perancangan perangkat keras dalam penelitian ini dibuat untuk membuat suatu sistem yang tepat dan utuh. Pada subbab ini, penulis menganalisis kebutuhan sistem yang dijelaskan proses kerja dari perangkat keras menggunakan blok diagram. Blok diagram digunakan sebagai alat untuk membantu pembaca dalam mengenali titik masalah atau fokus perhatian secara cepat pada perangkat keras keseluruhan sistem. Pembahasan dari sistem ini akan dibagi menjadi 2 fokus, antara lain sistem kendali utama dan sistem robot *hoverboard* yang masing-masing tergabung menjadi satu, hal ini dapat dilihat pada Gambar 5.1.

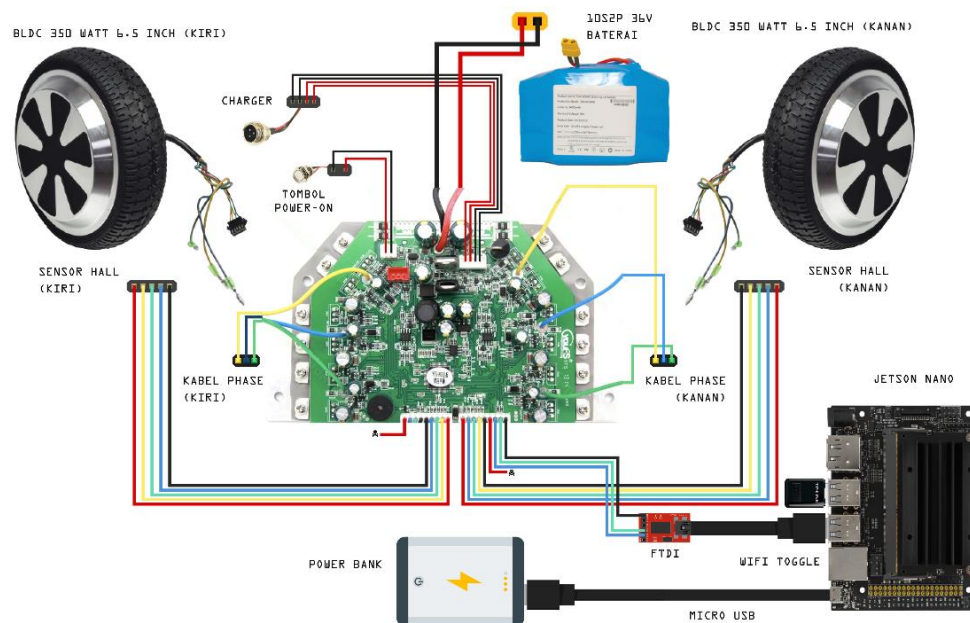


Gambar 5.1 Rancangan Perangkat Keras Sistem

Pada Gambar 5.1 dapat diperhatikan bahwa sistem menggunakan perangkat utama yang saling terintegrasi satu sama lain menggunakan beberapa protokol komunikasi sebagai penghubung. Perangkat utama yang dimaksud adalah Jetson Nano, personal komputer dan *hoverboard motherboard*, sedangkan protokol komunikasi penghubung yang dimaksud adalah *serial protocol* dengan FTDI mini USB dan VNC dengan jaringan Wi-Fi. Namun karena Jetson Nano tidak memiliki modul Wi-Fi pada komponen utamanya, maka kita perlu menambahkan modul berupa Wi-Fi *dongle* yang ditancapkan pada salah satu *port* USB Jetson Nano. Berikut ini merupakan penjelasan mengenai sistem kendali utama robot. Personal komputer atau komputer server berkomunikasi dengan Jetson Nano menggunakan protokol komunikasi *Virtual Network Computing* (VNC) pada jaringan yang saling terhubung dengan Wi-Fi yang sama. Sedangkan untuk Jetson Nano dengan *hoverboard motherboard* menggunakan komunikasi data *serial* UART melalui perangkat elektronik bernama FTDI mini USB *converter*. Jetson Nano merupakan komponen utama penyusun sistem penggerak.

Sistem penggerak berperan sebagai aktuator yang sekaligus mengirimkan data berupa data *odometry*. Sistem ini terdiri dari *motherboard hoverboard*, motor BLDC, baterai, *power on/off* dan Jetson Nano yang terhubung dengan FTDI melalui komunikasi UART. *Motherboard* berfungsi sebagai motor *driver* utama yang mengatur arah putar dan kecepatan putar motor BLDC. Dengan menggunakan 3 kabel *phase* di tiap sisi motor, *motherboard hoverboard* dapat dengan mudah mengatur kecepatan dengan memanipulasi nilai voltase yang dialirkan melalui 3 kabel *phase* ini. Berbeda dengan motor DC biasa yang menggunakan VCC dan *ground* saja, motor BLDC memerlukan 3 kabel *phase* untuk mengatur tegangan yang masuk di tiap *phase* magnetnya. Selain itu dibutuhkan juga 5 *pin hall* sensor yang digunakan untuk mengetahui posisi putaran robot sehingga didapatkan data *Odometry* pada motor BLDC.

5.1.1.1 Perancangan Perangkat Keras Sistem Utama



Gambar 5.2 Skema Sistem Penggerak Robot

Pada Gambar 5.2 terdapat diagram skematis dari sistem gerak robot di mana *motherboard hoverboard* yang berfungsi sebagai *driver* motor BLDC, terhubung langsung dengan kabel *phase*, *hall*, *on/off*, *charger* dan *supply*. Kabel *phase* dan kabel *hall* sudah dijelaskan sebelumnya, yakni untuk mengendalikan dan mendapatkan data *odometry* dari motor BLDC. Kabel *on/off power button* terhubung dengan soket yang sudah tersedia pada *motherboard* begitu pula dengan kabel *charger port*. Kemudian untuk dapat berkomunikasi dengan Jetson Nano, kita perlu menghubungkan *motherboard* dengan kabel sensor yang ada di sisi bawah *motherboard hoverboard*.

Kedua sisi sensor *motherboard hoverboard* memiliki total 8 buah kabel yang masing-masing terdiri dari VCC, RX, TX, dan *ground*. Kedua kabel ini seharusnya terhubung dengan sensor giroskop pada *hoverboard*, namun kita bisa membuang bagian sensor tersebut dan mengambil bagian *motherboard* saja. Namun secara

pabrik, *hoverboard* tidak dibuat untuk tujuan ini maka diperlukan proses *flashing firmware* pada *motherboard hoverboard*. Dari kedua *port* sensor ini, kita hanya menggunakan kabel sensor sebelah kanan. Kabel sebelah kanan digunakan karena tegangannya dapat mentoleransi komunikasi data dengan tegangan maksimal 5V. Konfigurasi ini dapat dilihat pada *datasheet* jenis *motherboard hoverboard* yang digunakan.

Hal ini dilakukan untuk menghindari kerusakan pada Jetson Nano yang mana jika kita menggunakan kabel sensor sebelah kiri maka *motherboard* bisa saja mengirimkan data dengan voltase sebesar 15V. Voltase sebesar itu tidak bisa diterima oleh Jetson Nano dan hal ini memicu terjadi *short* aliran listrik sehingga Jetson Nano bisa saja terjadi hubungan arus pendek atau lebih parahnya adalah terbakar. Kemudian perlu diperhatikan juga, dari ke-4 kabel sensor kanan (VCC MCU, RX MCU, TX MCU, GND MCU) hanya menggunakan 3 dari 4, jadi bagian *pin* yang digunakan hanyalah RX, TX, dan GND. Pin VCC tidak digunakan karena mengakibatkan *short circuit* dan membuat rusak *motherboard*. Dengan menghubungkan *pin* VCC berarti sama saja kita menabrakkan dua tegangan dengan voltase yang jauh berbeda. *Output* dari *pin* VCC pada sensor kanan *motherboard* memiliki nilai tegangan yang tinggi yaitu 15V, maka dari itu *pin* ini digunting dan di isolasi agar tidak membahayakan jikalau lupa bahwa *pin* VCC ini tidak boleh digunakan.

Selanjutnya untuk terhubung dengan Jetson Nano melalui USB *port*, dibutuhkan sebuah adapter yang mampu menjembatani komunikasi antar perangkat. Dalam penelitian ini digunakan FTDI mini USB untuk menjembatani kabel RX/TX sensor kanan dengan RX/TX Jetson Nano. Dari perangkat FTDI dilanjutkan dengan kabel mini USB menuju *port* USB Jetson Nano. Untuk penjelasan lebih lanjut mengenai apa saja *pin-pin* yang digunakan dari *motherboard* dapat dilihat pada Tabel 5.1.

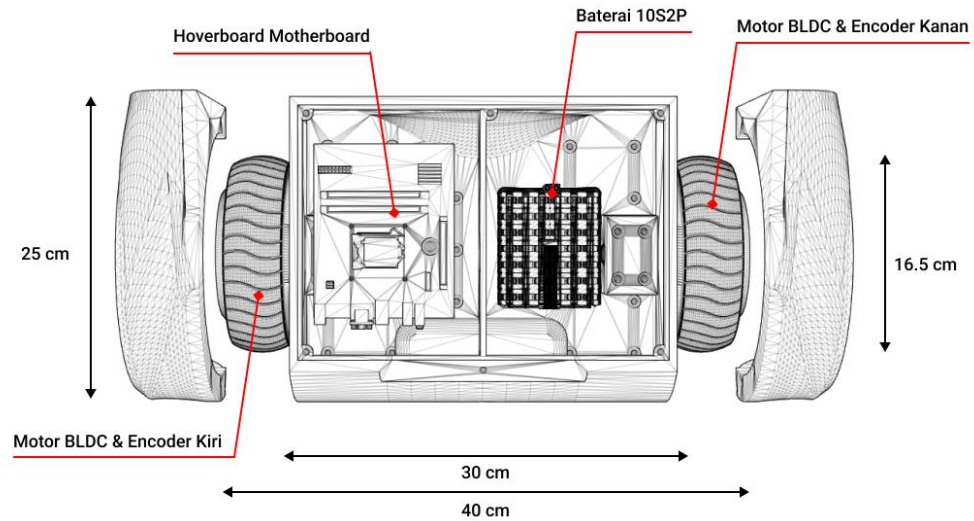
Tabel 5.1 Penjelasan Konfigurasi *Pinout Motherboard Hoverboard*

Sumber: (github.com/EFeru/hoverboard-firmware-hack-FOC, 2018)

No.	Pin <i>motherboard</i>	Pin tujuan	Deskripsi
1	XT60 VCC	36V	Pin daya <i>motherboard</i>
2	XT60 GND	GND	Pin <i>ground motherboard</i>
3	<i>Button Pin</i>	<i>Button Pin</i>	Pin tombol <i>power</i> kondisi <i>on</i>
4	<i>Latch Pin</i>	<i>Latch Pin</i>	Pin tombol <i>power</i> kondisi <i>off</i>
5	<i>Charger Connector</i>	<i>Charger Connector</i>	Konektor <i>charger</i> baterai
6	<i>Phase A LEFT</i>	<i>Phase A</i>	Pin daya untuk kutub A pada motor kiri

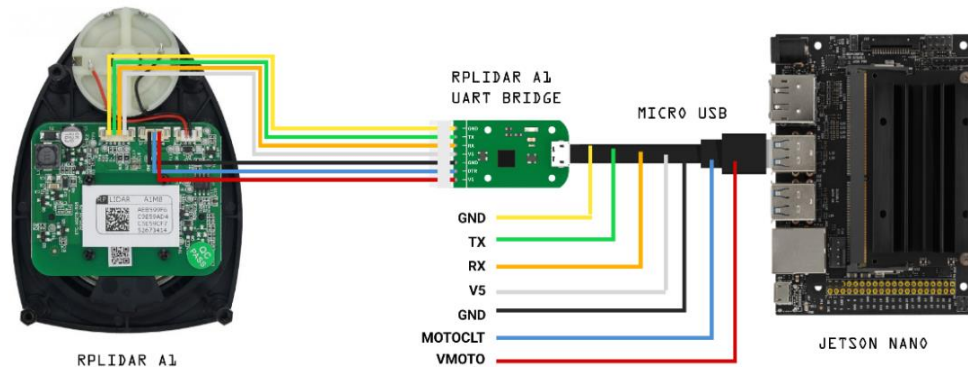
7	<i>Phase B</i> LEFT	<i>Phase B</i>	Pin daya untuk kutub B pada motor kiri
8	<i>Phase C</i> LEFT	<i>Phase C</i>	Pin daya untuk kutub C pada motor kiri
9	HALL A LEFT	HALL A	Pin data <i>encoder</i> pada motor kiri
10	HALL B LEFT	HALL B	Pin data <i>encoder</i> pada motor kiri
11	HALL C LEFT	HALL C	Pin data <i>encoder</i> pada motor kiri
12	GND LEFT	GND	Pin <i>ground encoder</i> motor kiri
13	VCC LEFT	VCC	Pin daya <i>encoder</i> motor kiri
14	<i>Phase A</i> RIGHT	<i>Phase A</i>	Pin daya untuk kutub A pada motor kanan
15	<i>Phase B</i> RIGHT	<i>Phase B</i>	Pin daya untuk kutub B pada motor kanan
16	<i>Phase C</i> RIGHT	<i>Phase C</i>	Pin daya untuk kutub C pada motor kanan
17	HALL A RIGHT	HALL A	Pin data <i>encoder</i> pada motor kanan
18	HALL B RIGHT	HALL B	Pin data <i>encoder</i> pada motor kanan
19	HALL C RIGHT	HALL C	Pin data <i>encoder</i> pada motor kanan
20	GND RIGHT	GND	Pin <i>ground encoder</i> motor kanan
21	VCC RIGHT	VCC	Pin daya <i>encoder</i> motor kanan
22	TX/USART3	RX	Pin UART untuk data yang terhubung dengan FTDI ke Jetson Nano
23	PB11/RX/USART3	TX	Pin UART untuk data yang terhubung dengan FTDI ke Jetson Nano
24	GND	GND	Pin <i>ground</i> USART3
25	3.3V MCU	3.3V	Pin daya ST-LINK untuk <i>flashing</i>
26	SWCLK MCU	SWCLK	Pin <i>clock</i> ST-LINK untuk <i>flashing</i>
27	GND MCU	GND	Pin <i>ground</i> ST-LINK untuk <i>flashing</i>
28	SWDIO MCU	SWDIO	Pin data input dan output untuk <i>flashing</i>

Penempatan *motherboard hoverboard*, motor BLDC, baterai 10S2P, *power button*, dan kabel-kabel yang terhubung langsung dengan motor DC disusun sesuai yang dapat dilihat pada Gambar 5.3.



Gambar 5.3 Penempatan Sistem Penggerak Robot

5.1.1.2 Perancangan Perangkat Keras Sistem Persepsi Sensor



Gambar 5.4 Pinout Sensor RPLIDAR A1

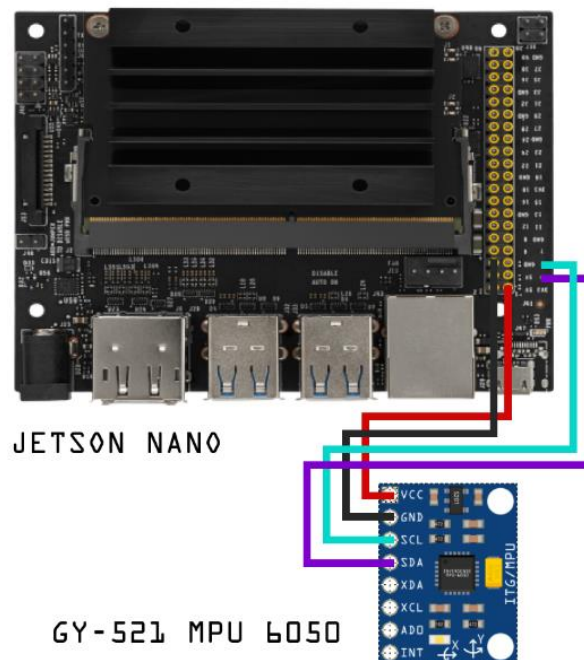
Seperti yang dapat dilihat pada Gambar 5.4 bahwa sensor RPLIDAR bekerja dengan ideal pada tegangan sebesar 5V yang bersumber dari Jetson Nano yang terhubung melalui kabel USB 2.0 *to micro* USB, sehingga dengan kabel ini kita dapat mendapatkan memberikan masukan tegangan sekaligus mengirim dan menerima data. Data yang diterima oleh Jetson Nano dari sensor RPLIDAR merupakan mentah berupa *point cloud* yang diproses oleh *driver*. Untuk menghubungkan RPLIDAR dengan kabel mikro USB diperlukan sebuah adapter dengan nama RPLIDAR A1 *UART Bridge*. Adapter ini berfungsi untuk menjembatani komunikasi sekaligus mengubah *pin* dari kabel kecil pada *input* dan *output* RPLIDAR menjadi *port* mikro USB. Sensor ini memiliki 7 buah *pin* yang memiliki fungsi seperti *supply* daya laser, data laser, *supply* daya motor, dan sinyal untuk motor. Untuk melihat penjelasan lebih lanjut mengenai *pin* dari RPLIDAR dapat dilihat pada Tabel 5.2 dan perangkat RPLIDAR dapat dilihat pada Gambar 5.4.

Tabel 5.2 Penjelasan *pinout* sensor RPLIDAR

Sumber: (www.slamtec.com, 2016)

No.	Pin modul	Pin tujuan	Deskripsi
1	GND	GND	Pin <i>ground</i> laser
2	TX	RX	Pin data dengan komunikasi <i>serial</i> UART
3	RX	TX	Pin data dengan komunikasi <i>serial</i> UART
4	V5	VCC	Pin daya pada laser
5	GND	GND	Pin <i>ground</i> motor
6	MOTOCLT	DTR	Pin sinyal motor
7	VMOTO	VCC	Pin daya pada motor

Sedangkan pada sensor GY-521 menggunakan tegangan sebesar 3.3V yang terhubung dengan *pin* Jetson Nano. Sensor GY-521 ini memiliki *port* dengan jumlah 8 *pin*, namun *pin* yang digunakan hanya berjumlah 4 buah yakni VCC & *Ground*, yang memiliki fungsi masukan daya dan *pin* SCL dan SDA, yang berfungsi sebagai pengirim data dengan protokol komunikasi I2C. Untuk melihat penjelasan lebih lanjut mengenai *pin* dari GY-521 dapat dilihat pada Tabel 5.3 dan untuk perangkat GY-521 dapat dilihat pada Gambar 5.5.



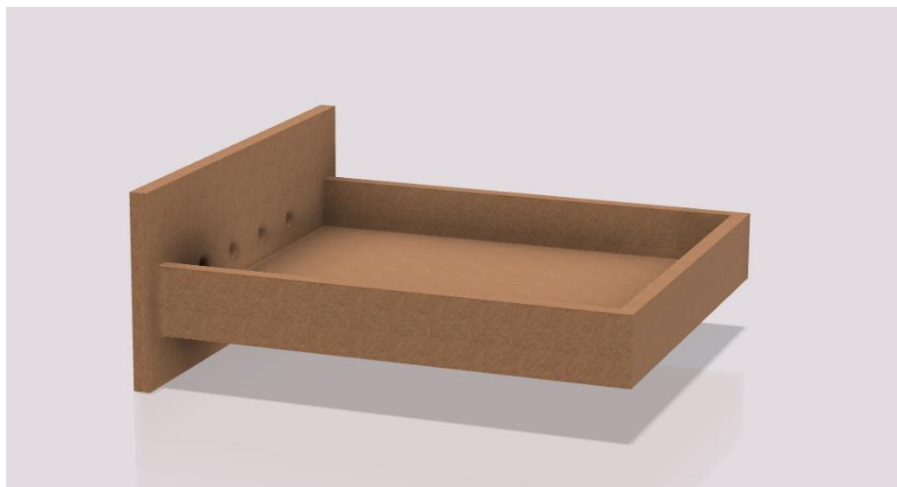
Gambar 5.5 *Pinout* sensor GY-521 IMU MPU6050

Tabel 5.3 Penjelasan *pinout* sensor GY-521 IMU MPU6050

Sumber: (xtcomp.co.za, 2022)

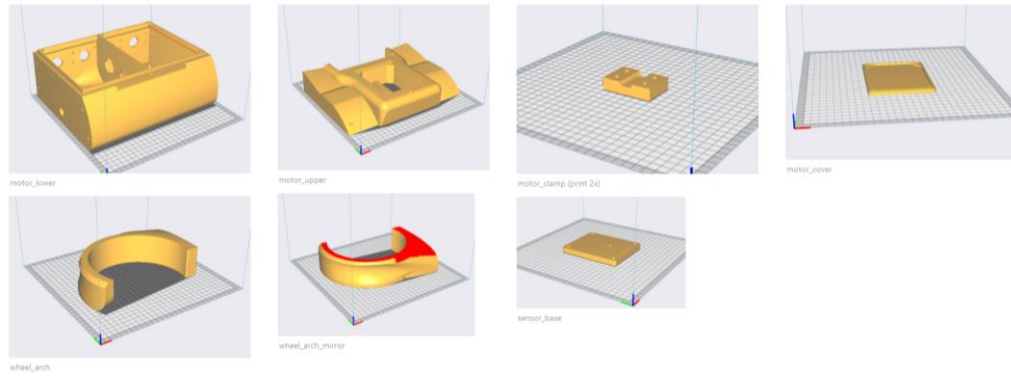
No.	Pin modul	Pin tujuan	Deskripsi
1	VCC	3.3V	Pin daya
2	GND	GND	Pin <i>ground</i>
3	SCL	GPIO 5 (SCL)	Pin data dengan komunikasi <i>serial</i> I2C
4	SDA	GPIO 3 (SDA)	Pin data dengan komunikasi <i>serial</i> I2C

5.1.1.3 Perancangan Perangkat Keras *Chassis*



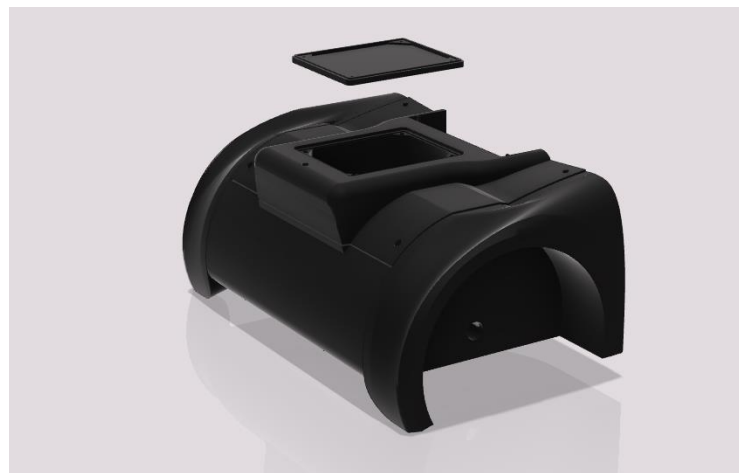
Gambar 5.6 Desain 3D CAD *Chassis* Badan Depan Robot

Purwarupa dari *autonomous mobile robot* dirancang dengan menggabungkan 3 komponen utama, yaitu *chassis* robot, robot beroda, dan sensor robot. *Chassis* robot dibuat dengan ukuran 40x40 cm berbahan kayu dengan ketebalan 2 cm. Pembuatan *chassis* ini cukup mudah, dari ke-4 papan kayu yang sudah disiapkan dipilih 1 papan sebagai alas. Kemudian papan ke-2 digunakan sebagai pembatas sisi dengan memotong papan tersebut menjadi 4 buah bagian dengan lebar masing-masing 10 cm. Lalu papan ke-3 dipotong dengan lebar 20 cm yang dijadikan sebagai penghubung sisi bagian belakang. Terakhir, papan ke-4 dibiarkan utuh yang nantinya digunakan sebagai penutup *chassis*. Untuk detail bentuk dari *chassis* ini dapat dilihat pada Gambar 5.6. Selanjutnya untuk bagian *chassis* robot beroda yang merupakan komponen utama robot sebagai penggerak dan dudukan motor. Terdiri dari gabungan hasil 3D *print* dan komponen elektronik *hoverboard*. Keseluruhan desain 3D ini dapat dilihat pada Gambar 5.7.



Gambar 5.7 Desain Semua Bagian *Printing* 3D

Desain dari komponen ini menggunakan desain dari proyek *Hover Mower*, yang merupakan robot pemotong rumput yang dapat di *download* pada *link* berikut: <https://github.com/HoverMower/hovermower-chassis>. Desain ini dapat digunakan bebas karena menggunakan lisensi GPL 3, jadi diperbolehkan menggunakan dan mengubah desain tersebut. Lisensi ini memperbolehkan untuk keperluan pembelajaran, riset, dan keperluan komersial. Pada bagian kedua sepatbor dijepit menggunakan baut M4, sedangkan bagian tutup cukup dijepit menggunakan baut M3. Semua komponen yang diberikan baut dipasangkan terlebih dahulu *brass knurled* sebagaiudukan untuk baut. Hasil dari gabungan semua desain ini dapat dilihat pada Gambar 5.8.



Gambar 5.8 *Assembling* Semua Bagian *Chassis* Belakang Robot

Komponen utama ketiga adalah sensor robot yang terletak pada bagian depan robot. Sensor terdiri dari sensor utama RPLIDAR, dan dua sensor untuk lokalisasi yaitu sensor IMU dan *build-in* Odometer. Gambar 5.9 merupakan gambar Rancangan 3D Keseluruhan Sistem yang terdiri dari bagian *mainboard* robot beroda sebagai penggerak, *chassis* robot sebagai kerangka dan sensor robot sebagai penglihatan dari robot. Semua desain ini dibuat menggunakan aplikasi Autodesk Fusion 360 yang merupakan aplikasi untuk desain 3D yang dikembangkan oleh Autodesk.



Gambar 5.9 Rancangan 3D Keseluruhan Sistem Tampak Atas

Sistem ini merupakan implementasi pengembangan dari penelitian sebelumnya yang dibahas pada bab 2 yang hanya berfokus pada *mapping* saja atau navigasi saja. Fokus utama dari penelitian ini adalah menciptakan purwarupa *autonomous mobile robot* yang mampu bergerak dari titik awal menuju titik tujuan dengan menggabungkan *mapping* dan navigasi. Proses *input* titik koordinat tujuan dapat dilakukan baik secara otomatis maupun manual. Proses secara otomatis menggunakan metode penjadwalan dengan melakukan perintah *terminal* dengan memanfaatkan fungsi waktu. Untuk proses manual menggunakan tampilan antar muka RViz dan meletakkan 2D *Nav Goal* pada aplikasi RViz. Pada penelitian ini, penulis menerapkan kedua metode ini pada pengujian sistem. Navigasi menggunakan global dan lokal *path planning*, sehingga sistem dapat bergerak pada lintasan dengan halangan statis dan dinamis.

5.1.2 Perancangan Perangkat Lunak

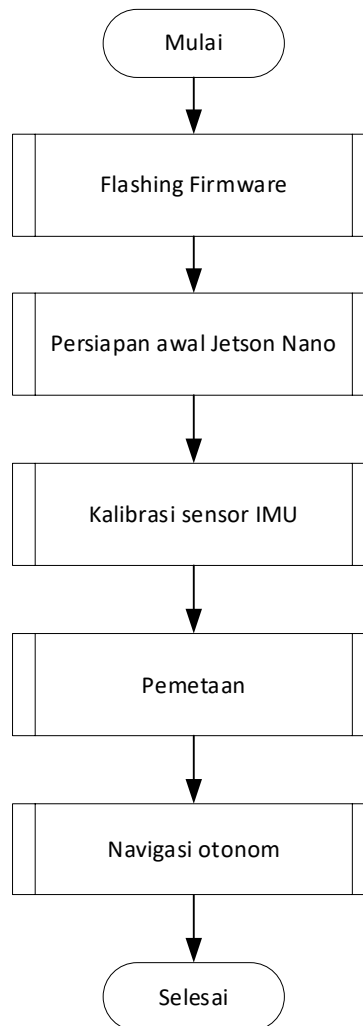
Pada subbab ini, penulis menjelaskan mengenai rancangan perangkat lunak yang digunakan pada penelitian ini. Perangkat lunak merupakan komponen penting dalam sistem guna menentukan bagaimana cara sistem bekerja. Perangkat lunak terdiri dari program perancangan utama yang menjadi keseluruhan alur dari semua proses. Perancangan program utama terbagi menjadi 4 proses besar yakni proses persiapan awal, kalibrasi sensor IMU, pemetaan dan navigasi Otonom. Ringkasan mengenai 5 proses besar ini dapat dijelaskan sebagai berikut.

1. Proses *flashing firmware* merupakan tahap mempersiapkan *motherboard hoverboard* untuk dilakukan *flashing* dengan *firmware custom* agar dapat digunakan pada sistem ini.
2. Proses persiapan Jetson Nano merupakan tahap mempersiapkan perangkat Jetson Nano agar dapat digunakan. Proses ini terdiri dari komunikasi Jetson Nano dengan laptop server menggunakan VNC dan proses *flashing firmware motherboard hoverboard*.
3. Proses kalibrasi sensor IMU merupakan tahap mempersiapkan sensor IMU agar dapat berfungsi dengan semestinya dengan dilakukan kalibrasi.

4. Proses pemetaan merupakan tahap pembuatan peta dengan menggunakan algoritme Hector SLAM, sebelum nantinya digunakan untuk proses navigasi.
5. Proses navigasi otonom merupakan tahap akhir untuk membuat sistem dapat bermanuver dengan secara mandiri sehingga menciptakan navigasi otonom.

5.1.2.1 Perancangan Program Utama

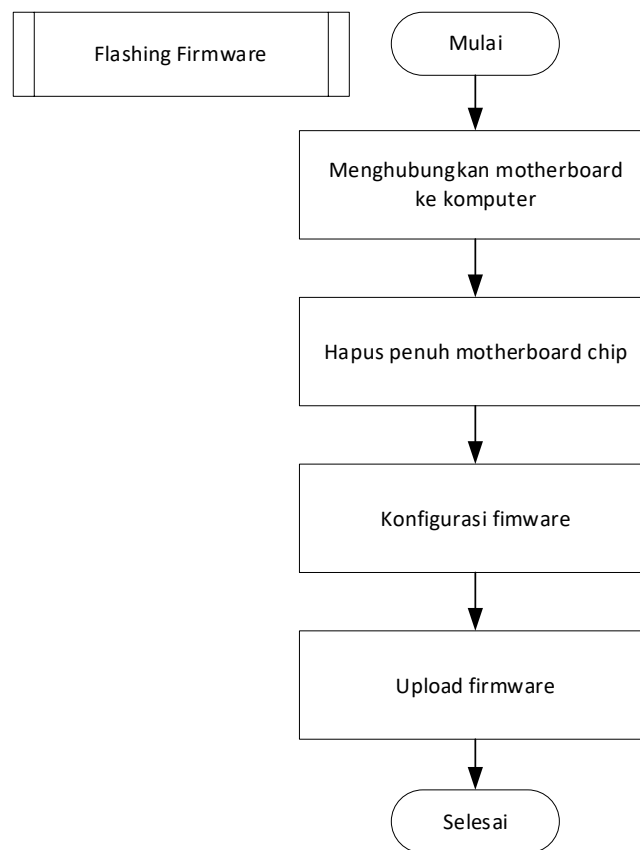
Perancangan program utama mencakup program utama pada penelitian ini, mulai dari melakukan persiapan awal, kalibrasi sensor IMU, pemetaan, dan navigasi robot. Kalibrasi hanya dilakukan pada sensor IMU karena sensor lainnya seperti RPLIDAR dan *encoder* di dalam motor BLDC sudah memiliki nilai *raw* yang presisi. Semua alur pada perancangan program utama menjadi diagram alir utama terhadap sub program lainnya. Diagram alir yang menjelaskan proses dari program utama dapat dilihat pada Gambar 5.10.



Gambar 5.10 Diagram Alir Program Utama

Pada Gambar 5.10 merupakan diagram alir perancangan program utama berisi poin-poin penting dari sebuah program secara keseluruhan. Semua urutan program ini sudah mencakup hal-hal atau kriteria yang dibutuhkan untuk membuat robot dengan sistem *Autonomous Mobile Robot* yaitu pemetaan dan navigasi. Hal pertama yang dilakukan dalam program ini adalah melakukan kalibrasi pada sensor IMU. Proses kalibrasi yang hanya dilakukan pada sensor IMU dikarenakan sensor IMU membutuhkan nilai kalibrasi pada setiap kondisi peletakannya. Jadi jika sensor IMU dipindahkan ke tempat lain, maka perlu dilakukan kalibrasi ulang. Hal ini tidak diperlukan kedua sensor lainnya, yaitu RPLIDAR dan *encoder* pada motor BLDC dikarenakan data *raw* yang konsisten dan perangkat dilengkapi dengan *microcontroller* masing-masing. Detail lebih lanjut mengenai alur dari kalibrasi ini dapat dilihat pada Gambar 5.15.

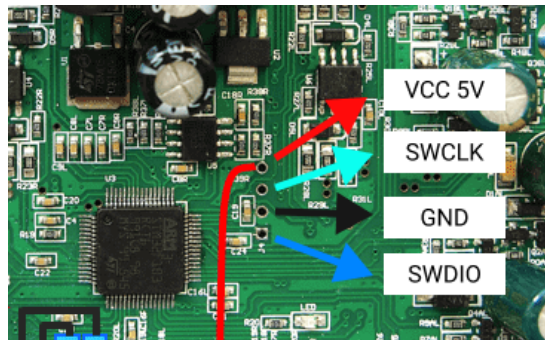
5.1.2.2 Perancangan Proses *Flashing Firmware*



Gambar 5.11 Diagram Alir Proses *Flashing Firmware*

Pada Gambar 5.11 ditunjukkan bahwa perancangan persiapan awal *motherboard* terdiri dari beberapa proses-proses persiapan sebelum merancang perangkat lunak sistem. Jadi tahap persiapan awal bukanlah proses atau alur algoritme sistem, melainkan tahap-tahap yang harus dijalankan sebelum membuat perangkat lunak sistem. Hal ini penting dikarenakan sistem perangkat lunak yang tidak biasa. Perangkat lunak pada robot ini terdiri dari banyak sekali dependensi dan paket-paket. Selain itu robot pada penelitian ini menggunakan

robot dari komponen *hoverboard* yang dimodifikasi sedemikian rupa sehingga proses persiapan awal ini sangat penting dilakukan untuk menciptakan robot modifikasi ini. Hal yang pertama dilakukan adalah menghubungkan komputer personal dengan *motherboard*. Ini merupakan tahap “*connecting*” untuk nantinya dilakukan proses *flashing*. Proses ini dimulai dengan memberikan kabel *jumper* pada perangkat ST-LINK yang kemudian dihubungkan dengan 4 buah *pin* yang ada di *motherboard*.

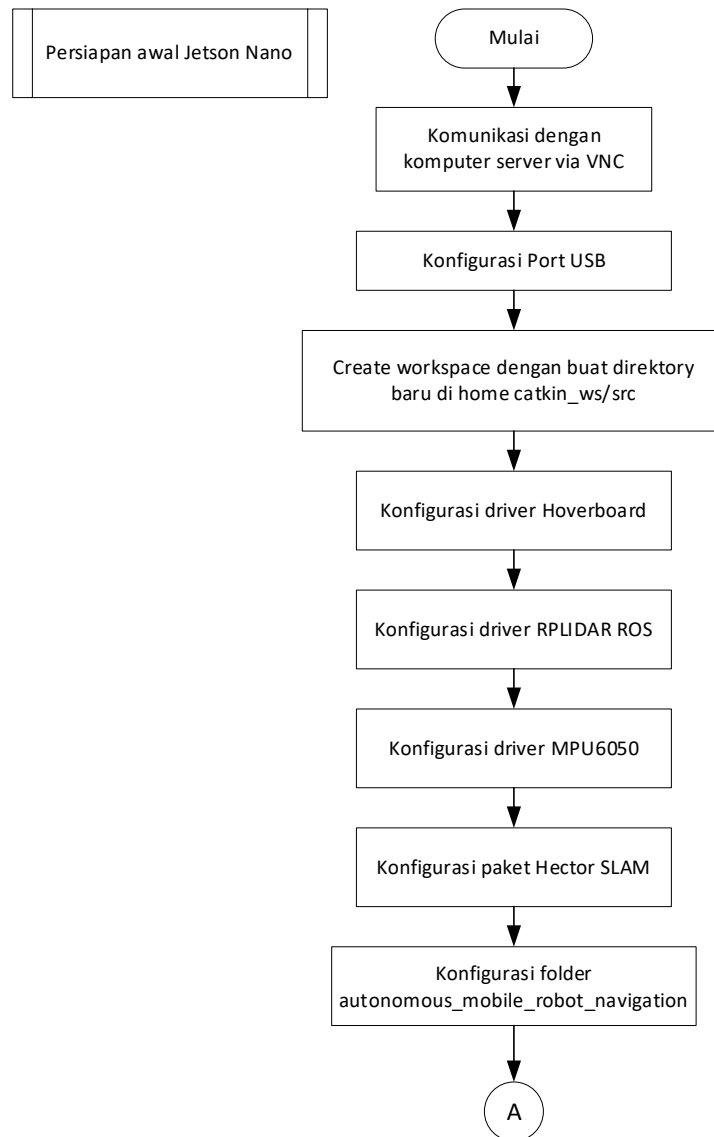


Gambar 5.12 Pinout dari *Motherboard* ke ST-LINK

Pada Gambar 5.12 dapat dilihat bahwa secara berurutan posisi *pin flash firmware* terdiri dari VCC, SWCLK, GND, SWDIO. Port ini dihubungkan dengan kabel *jumper male to female* menuju ST-LINK untuk selanjutnya ditancapkan ke port USB ke personal komputer. Selanjutnya untuk *motherboard hoverboard* proses ini dimulai dengan menghapus seluruh data yang ada di dalam *chip motherboard*. Hal ini bertujuan untuk menghapus kode program pabrik yang ditulis oleh pembuat *hoverboard*. Langkah selanjutnya adalah melakukan konfigurasi *firmware* yang merupakan proses utama dalam tahap persiapan awal *motherboard*. Konfigurasi *firmware* dibutuhkan untuk menyesuaikan jenis *board* yang digunakan berdasarkan kebutuhannya pengembang. Proses konfigurasi ini dimulai dengan melakukan *download firmware hoverboard hack FOC*. *Firmware hoverboard* ini sudah selesai ditulis, maka dari itu kita hanya perlu melakukan konfigurasi pada beberapa baris kode untuk menyesuaikan dengan kebutuhan penelitian. *File* yang menjadi fokus modifikasi adalah *file config* maka dari itu pada proses selanjutnya adalah dengan melakukan edit beberapa baris pada *file config.h* yang dijelaskan lebih lanjut pada subbab implementasi.

Tahap selanjutnya adalah melakukan *upload firmware motherboard*, jadi setelah *firmware* sudah siap dan *file config* sudah di edit, langkah selanjutnya adalah dengan melakukan *building code* menjadi sebuah *file* dengan ekstensi biner dengan melakukan proses *compile* dengan klik ikon centang atau “PlatformIO: *build*”. Maka langkah selanjutnya adalah dengan melakukan proses *upload file* *firmware.bin* dengan klik ikon panah atau “PlatformIO: *upload*” maka terjadi proses penyalinan data biner pada *file* *firmware.bin* ke *motherboard* dan pesan berhasil muncul. Maka dengan begini proses persiapan awal *motherboard* telah selesai. *Motherboard* sudah berada dalam kondisi siap digunakan dan sudah bisa dikendalikan menggunakan kabel sensor *motherboard* bagian kanan. Setelah proses ini maka berlanjut pada tahap persiapan awal perangkat Jetson Nano.

5.1.2.3 Perancangan Persiapan Awal Jetson Nano



Gambar 5.13 Diagram Alir Persiapan Awal Jetson Nano

Pada Gambar 5.13 dapat dilihat bahwa diagram alir dimulai dengan sub-proses komunikasi antara Jetson Nano dengan komputer server melalui *Virtual Network Computing* (VNC). Proses ini sangat berguna karena untuk melakukan *mapping* dibutuhkan sebuah kendali *remote* yang mampu menampilkan aplikasi GUI, metode yang paling mudah adalah dengan menggunakan VNC viewer. Tahap selanjutnya setelah koneksi *remote* melalui VNC adalah melakukan koneksi *remote* melalui terminal. Selanjutnya adalah melakukan konfigurasi *port* USB, hal ini dilakukan untuk mengubah konfigurasi *default* pada Linux dan membuat *port* USB yang terhubung ke RPLIDAR dan FTDI menjadi lebih fleksibel dan mudah untuk dikenali.

Setelah melakukan konfigurasi *port* USB selanjutnya melakukan pembuatan *workspace catkin* dengan membuat folder *catkin_ws* di dalam folder *home* Linux

dan membuat folder *src* di dalam folder *catkin_ws*. *Workspace* ini yang menjadi *directory parent* yang mewadahi *src* atau *source* pemrograman ROS pada *development software* robot. Jika sudah, maka selanjutnya adalah melakukan instalasi beberapa *driver* yang dibutuhkan ke dalam folder *src* seperti *driver hoverboard*, *driver rplidar_ros*, paket *hector_slam* dan folder *autonomous_mobile_robot_navigation* yang masing-masing terdapat pada Tabel 5.4, Tabel 5.5, dan Tabel 5.6.

Tabel 5.4 Parameter Konfigurasi hoverboard_driver *Driver Hoverboard*

No.	Parameter	Default	Konfigurasi Penelitian
1	left_wheel	"left_wheel"	"left_wheel"
2	right_wheel	"right_wheel"	"right_wheel"
3	pose_covariance_diagonal	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.001, 0.001, 1000000.0, 1000000.0, 1000000.0, 1000.0]
4	twist_covariance_diagonal	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.001, 0.001, 1000000.0, 1000000.0, 1000000.0, 1000.0]
5	publish_rate	50 Hz	50 Hz
6	wheel_separation_multiplier	1.0	0.34
7	cmd_vel_timeout	0.5 detik	0.5 detik
8	base_frame_id	"base_link"	"base_footprint"
9	linear/x/has_velocity_limits	false	true
10	linear/x/max_velocity	0 m/s	1.0 m/s
11	linear/x/mix_velocity	0 m/s	-0.5 m/s
12	linear/x/has_acceleration_limits	false	true
13	linear/x/max_acceleration	0 m/s ²	0.8 m/s ²
14	linear/x/min_acceleration	0 m/s ²	0 m/s ²
15	linear/x/has_jerk_limits	false	false
16	linear/x/max_jerk	0 m/s ³	0 m/s ³

17	angular/z/has_velocity_limits	false	true
18	angular/z/max_velocity	0 rad/s	3.14 rad/s
19	angular/z/min_velocity	0 rad/s	-3.14 rad/s
20	angular/z/has_acceleration_limits	false	true
21	angular/z/max_acceleration	0 m/s ²	3.14 m/s ²
22	angular/z/min_acceleration	0 m/s ²	3.14 m/s ²
23	angular/z/has_jerk_limits	false	true
24	angular/z/max_jerk	0 m/s ³	3.14 m/s ³
25	enable_odom_tf	true	false
26	wheel_separation	0.0	0.32
27	wheel_radius	0.0 meter	0.0825 meter
28	odom_frame_id	"/odom"	"/odom"
29	publish_cmd	false	false
30	allow_multiple_cmd_vel_publisher	false	true
31	velocity_rolling_window_size	10	10

Tabel 5.5 Parameter Konfigurasi paket rplidar_ros Sensor RPLIDAR

No.	Parameter	Default	Konfigurasi Penelitian
1	serial_port	"/dev/ttyUSB0"	"/dev/ttyRPLIDAR"
2	serial_baudrate	115200	115200
3	frame_id	laser_frame	laser
4	inverted	false	false
5	angle_compensate	false	true
6	scan_mode	""	""

Tabel 5.6 Parameter Konfigurasi mpu6050_driver Sensor IMU

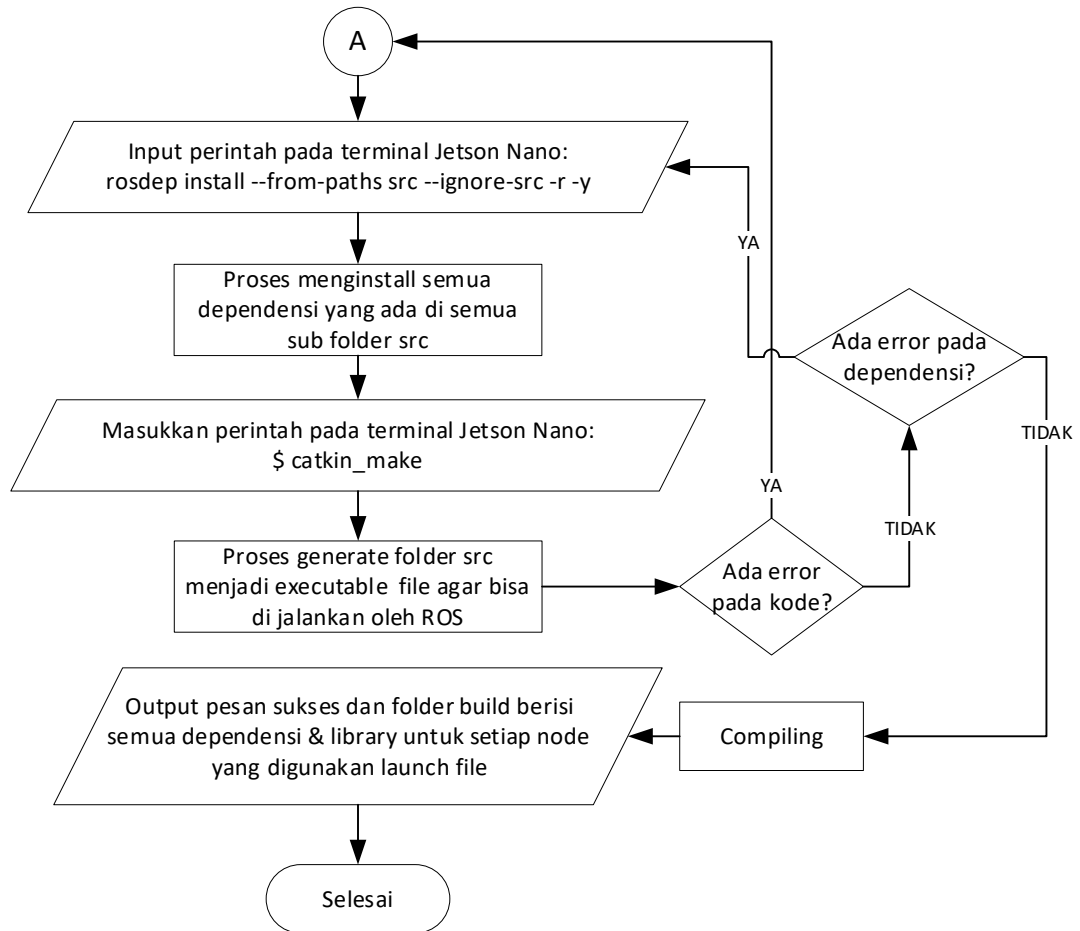
No.	Parameter	Default	Konfigurasi Penelitian
1	bus_uri	/dev/i2c-1	/dev/i2c-1
2	mpu_address	0x68	0x68
3	pub_rate	30 hertz	25 hertz
4	frame_id	"imu"	"imu"
5	axes_offsets	[0, 0, 0, 0, 0, 0]	[-1340, -9, 1463, 123, -42, 42]
6	ki	0.1	0.2
7	kp	0.1	0.1
8	delta	0.5	0.5

Untuk konfigurasi folder pemrograman utama, yaitu *autonomous_mobile_robot_navigation* dengan membuat folder dengan perintah di terminal `$ catkin_create_pkg autonomous_mobile_robot_navigation std_msgs roscpp rospy`. Detail mengenai konfigurasi ini dijelaskan pada subbab implementasi. Pada *file* package.xml terdapat beberapa paket dan *library* yang dibutuhkan pada penelitian ini, daftar paket dan *library* yang dibutuhkan terdapat pada Tabel 5.7.

Tabel 5.7 Daftar Paket Yang Digunakan pada AMR Navigation Package.xml

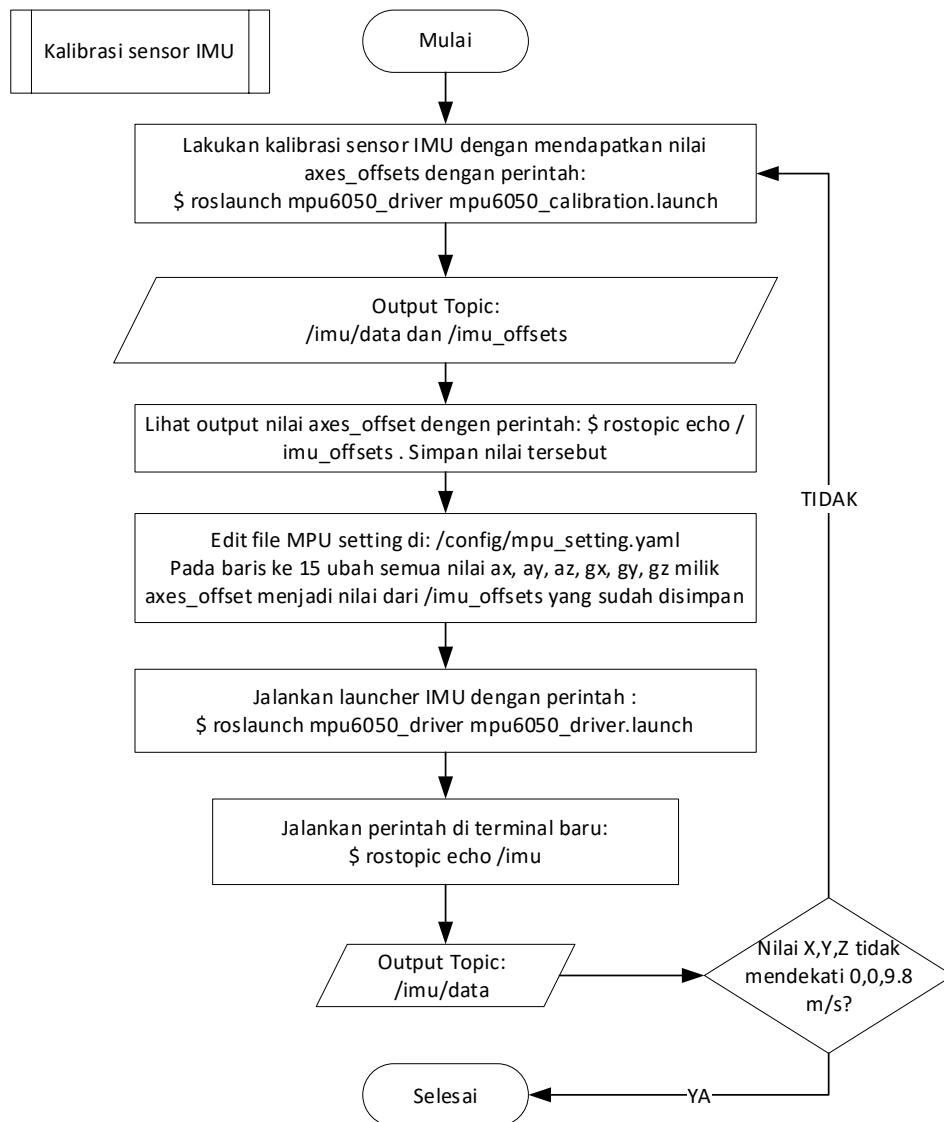
No.	Nama Paket	Tag
1	amcl	<depend> #namaPaket </depend>
2	actionlib	
3	actionlib_msgs	
4	geometry_msgs	
5	control_msgs	
6	message_generation	
7	controller_manager	
8	gmapping	
9	move_base	
10	roscpp	
11	rospy	

12	std_msgs	
13	std_srvs	
14	tf	
15	tf2	
16	control_toolbox	
17	map_server	
18	rosparam_shortcuts	
19	hardware_interface	
20	laser_filters	
21	imu_filter_madgwick	
22	rviz_imu_plugin	
23	robot_localization	
24	dwa_local_planner	
25	global_planner	
26	twist_mux	
27	nmea_navsat_driver	
28	moveit_ros_planning_interface	



Gambar 5.14 Diagram Alir Persiapan Awal Jetson Nano

Pada Gambar 5.14 yang merupakan lanjutan dari proses sebelumnya, setelah semua *driver* terpasang di dalam folder `~/catkin_ws/src` maka lakukan *install* dependensi semua *library* yang dibutuhkan ke dalam internal ROS dengan cara mengetikkan perintah `$ rosdep install --from-paths src --ignore-src -r -y`. Ketika perintah dijalankan maka terjadi proses *download* dan *install* dependensi yang dideklarasikan pada *file* `package.xml`. Setelah itu masukkan perintah `$ catkin_make` pada *terminal* sehingga terjadi proses *generate* oleh *cmake* sehingga *file generate* tersebut dapat digunakan pada platform ROS. Namun jika terjadi *error* pada baris kode, perbaiki kode dan lakukan alur dari *page A*, akan tetapi jika tidak ada *error* pada kode maka dilakukan pemeriksaan pada dependensi, apakah *node* yang di *generate* sudah memiliki *library* masing-masing di dalam sistem Linux. Jika “ya” maka kembali ke proses *install* dependensi ROS, namun jika tidak maka program *compile* berlanjut dan menampilkan *output compile* berhasil.



Gambar 5.15 Diagram Alir Sub Proses Kalibrasi Sensor IMU

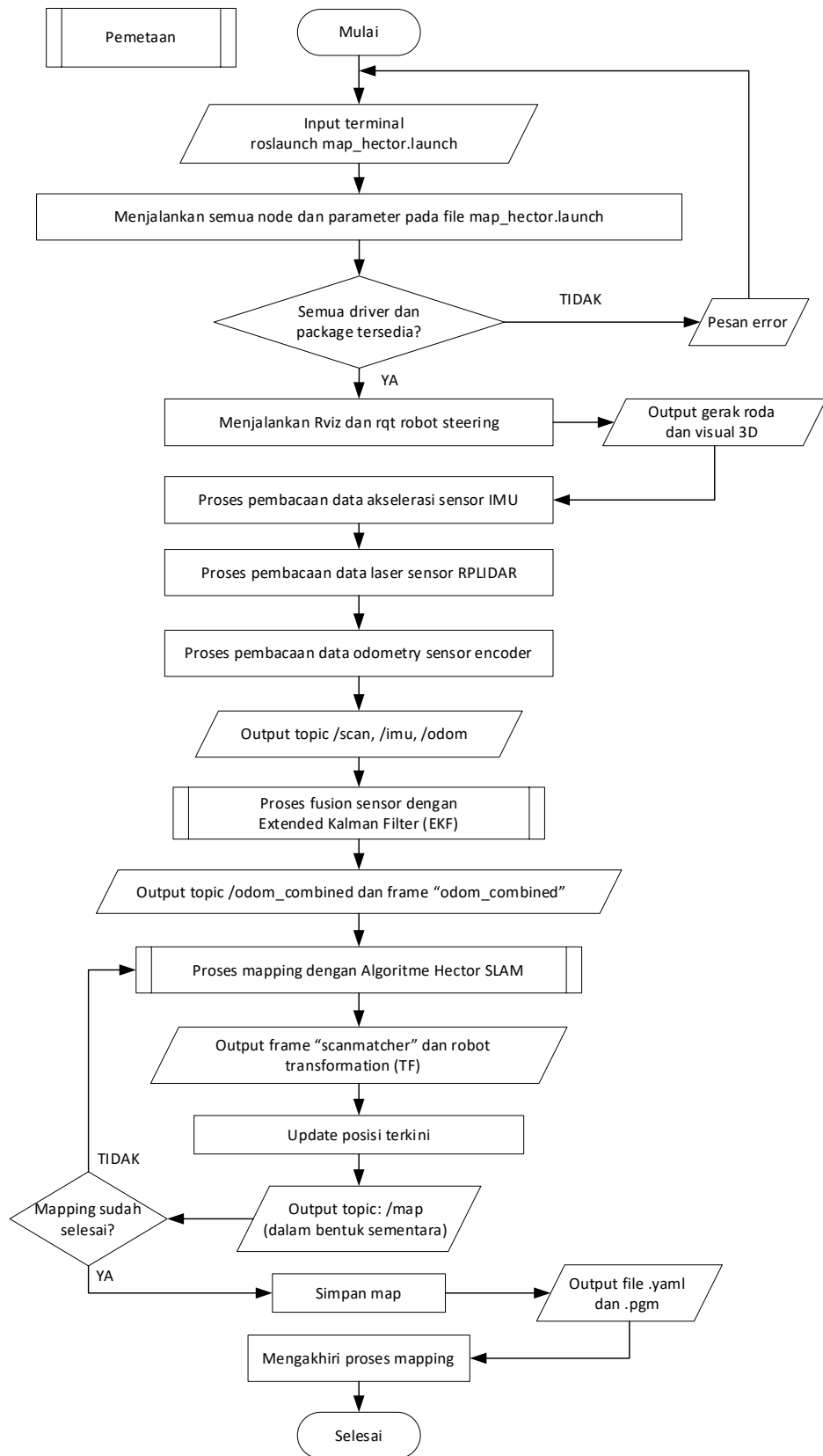
Pada diagram alir yang ditunjukkan pada Gambar 5.15 menjelaskan bahwa proses awal melakukan kalibrasi adalah dengan menjalankan perintah pada `$ roslaunch mpu6050_driver mpu6050_calibration.launch` pada terminal. Setelah menjalankan perintah tersebut maka *terminal* menjalankan proses kalibrasi dan menghasilkan *output* berupa topik baru dengan nama `/imu/data` dan `/imu_offsets`. Lihat nilai dari *output* tersebut dengan mencetaknya pada *terminal* dengan perintah `$ rostopic echo /imu_offsets` kemudian simpan nilai tersebut pada *text editor*. Langkah selanjutnya adalah mengubah *file setting* dari driver MPU pada *directory* `mpu6050_driver/config/mpu_setting.yaml`. Setelah proses edit selesai, langkah selanjutnya adalah menjalankan *file launcher driver* IMU dengan mengetikkan perintah `$ roslaunch mpu6050_driver mpu6050_driver.launch` pada terminal. Untuk dapat melihat *output* yang dihasilkan dari sensor, jalankan perintah `$ rostopic echo /imu` pada terminal. *Output* yang ditampilkan adalah nilai x,y,z pada akselerometer tidak mendekati 0

m/s, 0 m/s, dan 9.8 m/s maka proses kalibrasi perlu di ulang. Akan tetapi jika sudah mendekati nilai tersebut maka proses kalibrasi IMU sudah selesai.

5.1.2.4 Perancangan Program Pemetaan

Proses selanjutnya adalah *mapping* atau pemetaan, proses ini merupakan sub proses dari program utama, jadi penjelasan lebih lanjut mengenai pemetaan ini dibahas pada sub nomor berikutnya. Proses *mapping* ini kurang lebih berguna untuk mendapatkan data “map” pada suatu area di dalam ruangan. *Map* ini nantinya berguna untuk proses navigasi, hal ini dikarenakan untuk melakukan navigasi diperlukan proses *path planning* atau perencanaan jalur. Sehingga untuk mendapatkan data koordinat tujuan dan melakukan perencanaan jalur dibutuhkan data peta 2D, agar koordinat dapat di *initializes*. Setelah map berhasil dibuat, maka langkah selanjutnya adalah menyimpan map tersebut ke dalam folder yang sudah disiapkan khusus untuk *file map*.

Perancangan program pemetaan mencakup proses pembacaan sensor, lokalisasi, dan *hector mapping*. Program pemetaan merupakan salah satu program utama yang harus dijalankan terlebih dahulu sebelum melakukan program navigasi. Diagram alir dari perancangan program pemetaan ditunjukkan pada Gambar 5.16.

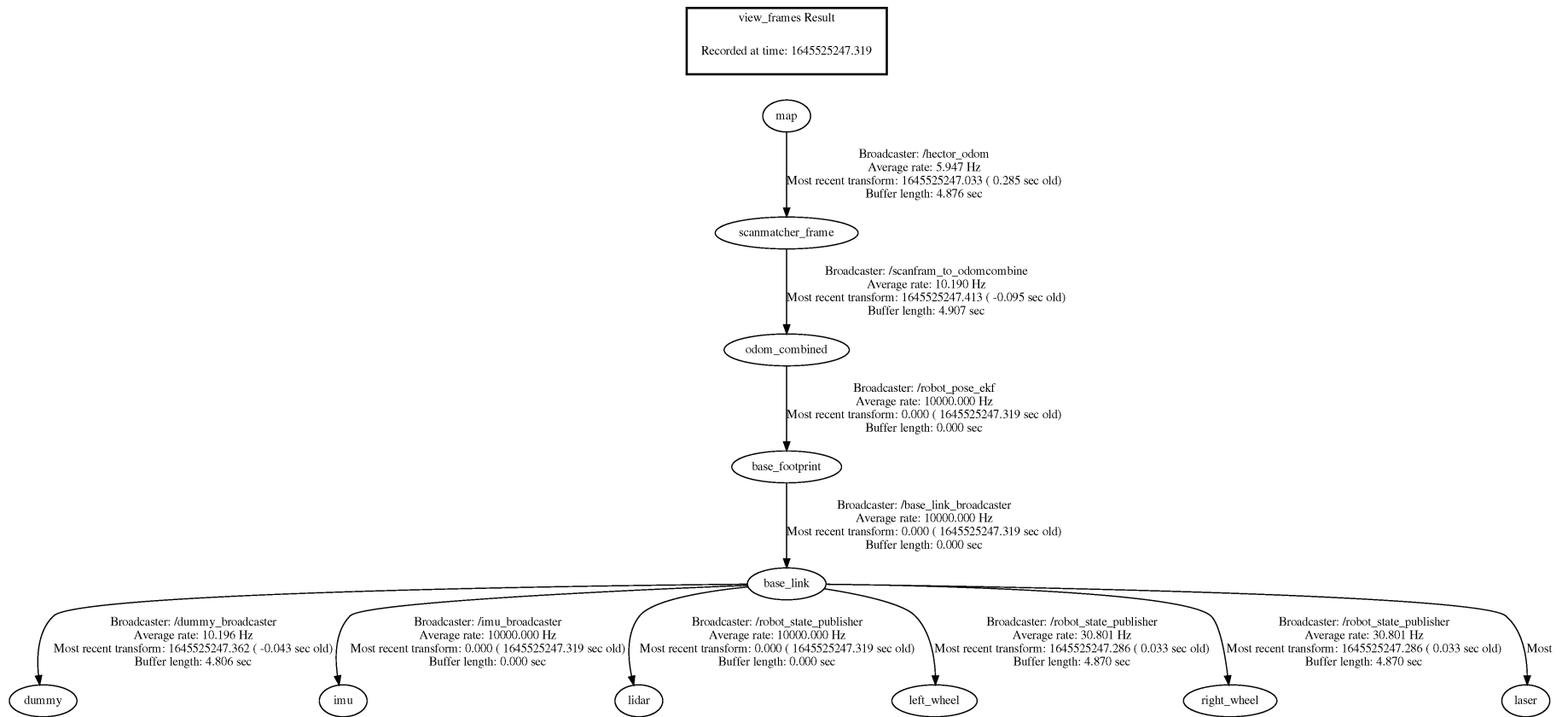


Gambar 5.16 Diagram Alir Program Pemetaan

Pada Gambar 5.16 merupakan diagram alir program pemetaan dengan proses yang saling berkaitan satu paket dengan paket lainnya. Diagram ini menjelaskan mengenai *node-node* yang dibutuhkan dalam proses *mapping*, seperti menjalankan semua *driver* sensor, *fusion* sensor, lokalisasi, *transform frame*, dan parameter-parameter yang digunakan. Semua hal ini dijadikan satu proses besar yang dinamakan *mapping* dan perencanaan dari proses-proses ini digabungkan ke dalam *file* dengan ekstensi *.launch* yang diberi nama *mapping.launch*.

Semua proses ini bermula dari perintah *terminal* yang di *input* oleh komputer server yang terhubung dengan VNC *client*. Seperti yang sudah dijelaskan pada bab 2 mengenai ROS dan perintah-perintah yang ada di dalamnya, terdapat *prefix* perintah “*roslaunch*” yang sering digunakan pada penelitian ini. *Prefix* ini memiliki arti, jalankan sebuah *ros system core* yang melakukan *launching* dengan *file* yang memiliki ekstensi “*.launch*” kemudian jalankan semua perintah-perintah di dalamnya. Jadi isi dari *file launch* ini adalah kumpulan perintah untuk menjalankan *node-node* beserta parameternya. *File* ini juga dapat memanggil *file launch* lainnya sehingga terjadi sinkronisasi karena tidak perlu memanggil *file launch* secara berulang-ulang. Setelah kita menjalankan *file launch mapping*, maka dependensi akan dijalankan. *Terminal* tidak bisa di *terminate* atau dihentikan, karena berakibat pada *node-node* yang sedang dijalankan pada *terminal* tersebut. Saat *terminal* menjalankan *file launch*, maka hal yang pertama kali dilakukan adalah melakukan pengecekan *driver* dan paket yang digunakan. Pengecekan ini hanya berlaku pada *node* yang disebutkan di dalam *file launch* tersebut, apabila terdapat *dependency* yang kurang atau ada beberapa paket pendukung *node* tersebut yang belum di *install* maka *terminal* menampilkan pesan *error* dan meminta untuk memasang paket tersebut. Selain menampilkan *error* karena paket yang belum terpasang, terkadang *terminal* juga menampilkan pesan *warning* jika terdapat kekurangan pada kode program. Selanjutnya jika semua *driver* dan paket sudah tersedia di dalam *dependency* ROS maka langkah selanjutnya adalah mengirimkan *input* dari ketiga sensor, yaitu pemindaian laser oleh RPLIDAR, akselerasi pergerakan robot oleh IMU, dan putaran *odometry* roda oleh *encoder* motor.

Perlu diperhatikan juga bahwa selama proses ini berlangsung, secara paralel *terminal* juga menjalankan aplikasi RViz dan juga menampilkan aplikasi rqt robot *steering*. Kedua aplikasi ini merupakan aplikasi pendukung yang memiliki tugasnya masing-masing. Hasil pemosisian robot akan terus diperbaharui secara simultan, sehingga kita dapat melihat posisi robot pada saat itu. Sedangkan aplikasi rqt robot *steering* digunakan menjadi *remote control* untuk mengendalikan robot selama proses *mapping* terjadi. Aplikasi ini melakukan *subscribe* pada topik *hoverboard_velocity_controller/cmd_vel* yang mana topik ini akan *publish* topik *geometry/pose* untuk menggerakkan motor, sehingga robot dapat bergerak dan sekaligus memberikan *input* data sensor *odometry* untuk digunakan pada proses selanjutnya. Ketika pertama kali dijalankan, setiap komponen akan membuat *frame* transformasi pada *node* masing-masing. Setiap *frame* tersebut harus saling terhubung dan terkoordinasi, perancangan yang diinginkan adalah seperti yang terlihat pada Gambar 5.17.



Gambar 5.17 Transformasi *Frame* Pada Pemetaan

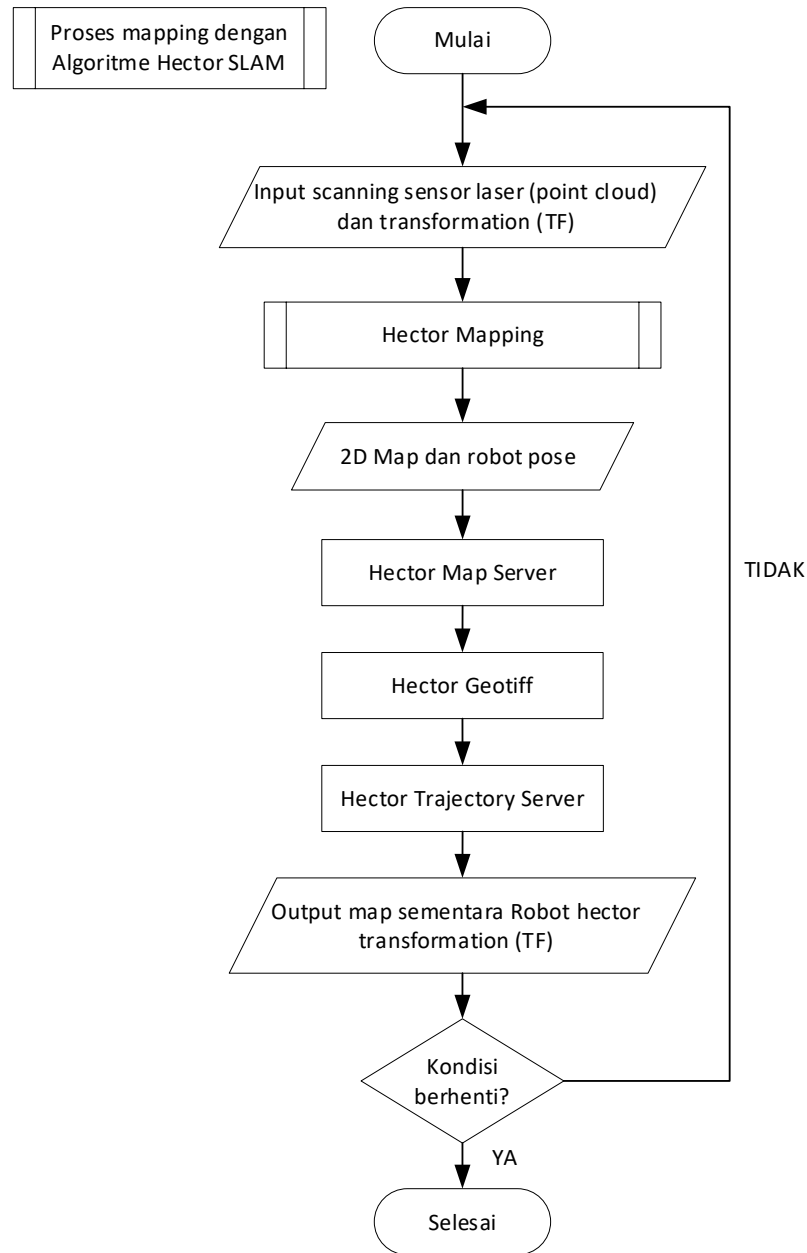
Pada Gambar 5.17 dapat terlihat bahwa setiap *node* dengan *broadcaster* saling menghubungkan sehingga membuat diagram alir yang sejajar. Untuk melakukan hal ini diperlukan konfigurasi *static* transformasi (TF) pada *file launcher* di program pemetaan. Terdapat beberapa keterangan seperti *broadcaster* yang merupakan *node* pengirim atau *publisher output* sebuah TF, *average rate* yang merupakan rata-rata pengiriman dalam Hertz *Flow* dimulai dari *node* map, *node* ini didapat dari *publisher node* *hector_mapping*. Selanjutnya TF tersebut dilanjutkan ke *odom_combined* yang di *broadcast* oleh algoritme EKF.

Melanjutkan proses sebelumnya yaitu mengirimkan *input* sensor ke *microcomputer*, proses ini dilakukan oleh *driver* pada masing-masing sensor. *Driver* ini memiliki *library* untuk melakukan pembacaan data sensor, pengolahan data, dan mengirimkan pesan catatan jika terjadi *error*. Pada *input* sensor pemindaian laser, *driver* RPLIDAR_ros berperan untuk membaca data dari tembakan laser dan mengubahnya ke dalam data *point cloud*. *Point cloud* merupakan setumpuk data *array* yang terdiri dari ribuan data per detiknya, dalam implementasinya *point cloud* ini berupa data jarak yang berhasil di *scan*. Pada umumnya sensor yang menggunakan teknik laser hanya mampu mengirimkan jarak pada satu titik fokus, namun sensor RPLIDAR memiliki kemampuan untuk melakukan pemindaian secara 360 derajat. Untuk memproses data putaran laser tersebut dibutuhkan *driver* khusus yang sudah disediakan oleh RoboPeak yang bisa di *download* langsung pada *website* atau *repository* GitHub. Tabel 5.5 adalah konfigurasi parameter yang terdapat pada *launcher rplidar_ros*.

Selagi proses pemindaian laser dilakukan, sistem juga melakukan pembacaan data akselerasi dari sensor IMU. Sensor ini memberikan data berupa percepatan pada sumbu x, y, dan z. X merupakan sumbu horizontal, jadi nilai akselerasi sumbu x berubah jika robot bergerak secara horizontal seperti berbelok kiri atau kanan. Sedangkan sumbu Y merupakan sumbu vertikal, jadi nilai akselerasi dari sumbu y berubah jika robot bergerak secara linier seperti maju atau mundur. Sedangkan nilai Z secara *default* bernilai 9.8 m/s mengikuti percepatan gravitasi bumi. Kemudian masih diproses paralel yang sama sistem ROS menjalankan *node odometry* di mana *driver hoverboard* melakukan pembacaan sensor *encoder*. Data yang didapatkan adalah sebuah pose dan orientasi dari roda, jadi perubahan orientasi dari posisi sebelumnya dapat terlihat langsung. Nilai dari orientasi dan posisi ini dihitung berdasarkan nilai putaran tiap rodanya yang dikonversi ke dalam radian per sekon.

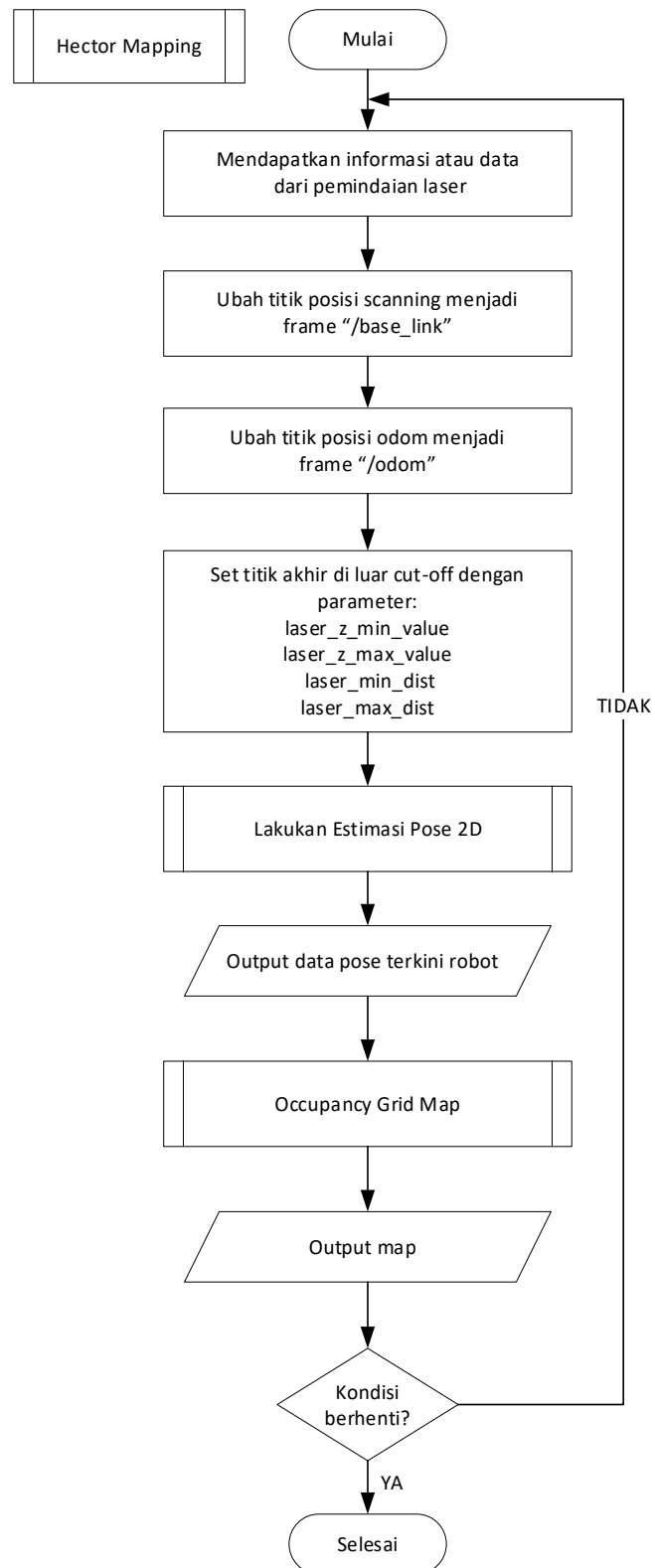
Proses selanjutnya mulai menggunakan algoritme yang digunakan pada penelitian ini. Setelah mendapatkan data dari sensor, sistem pada program *mapping* menggunakan dua algoritme yakni algoritme Hector SLAM dan algoritme EKF. Pada data sensor laser yang sudah menjadi data jarak ke dalam *point cloud*, langkah selanjutnya adalah memberikan data tersebut ke algoritme Hector SLAM.

5.1.2.5 Perancangan Proses Mapping Dengan Algoritme Hector SLAM



Gambar 5.18 Diagram Alir Proses Algoritme Hector SLAM

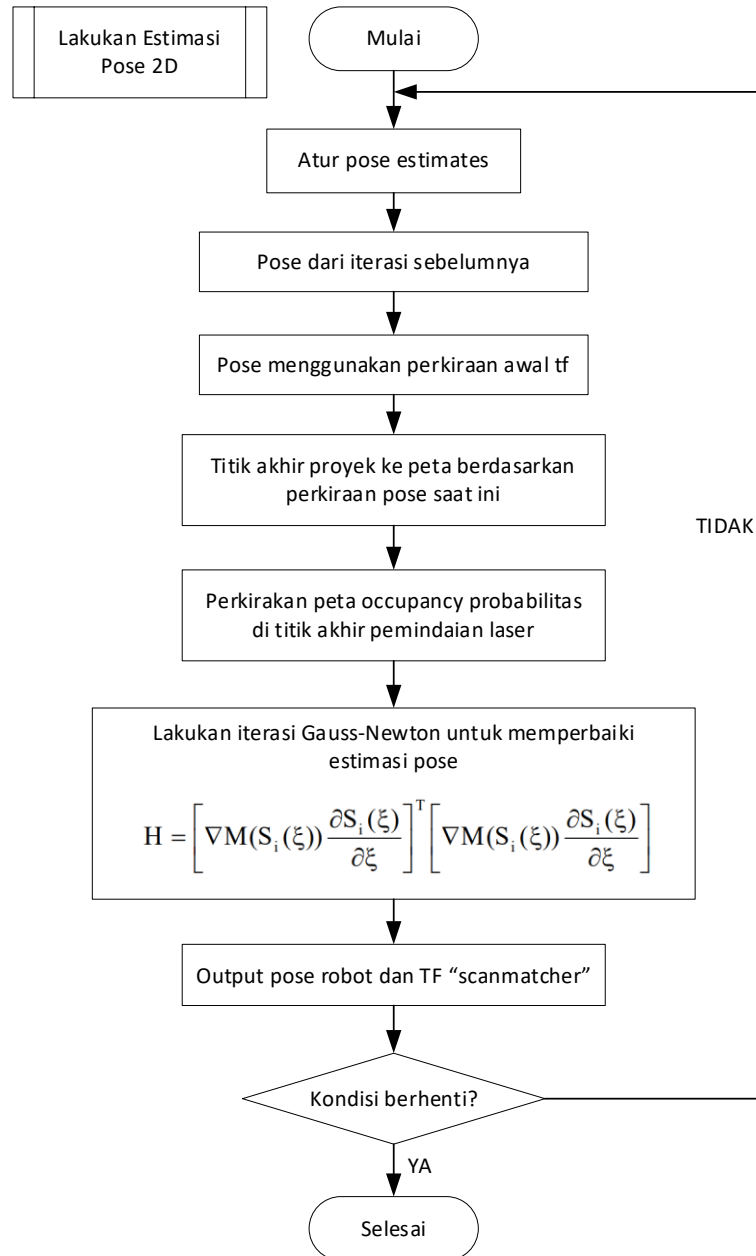
Pada Gambar 5.18 dapat dilihat bahwa proses dimulai dengan mengambil *input* dari *scanning* sensor laser dengan data berupa *point cloud* dan *input* dari *transformation* pose robot. Setelah itu *input scan* diolah ke dalam sub proses *Hector Mapping* dan menghasilkan 2D map serta pose, sedangkan *transformation* diolah oleh proses *Hector Trajectory Server*. Detail lebih lanjut mengenai sub proses *Hector Mapping* dapat dilihat pada Gambar 5.19.



Gambar 5.19 Diagram Alir Sub Proses Hector Mapping

Pada Gambar 5.19 dapat dilihat bahwa diagram alir terdiri dari beberapa proses konfigurasi dan terdapat proses inti berupa estimasi pose 2D. Diagram alir Hector Mapping dikutip dari jurnal peneliti yang dilakukan oleh (Souliman, 2019).

Dimulai dengan mendapatkan data informasi atau data dari pemindaian laser, kemudian proses beralir ke konfigurasi Hector *Mapping* berdasarkan konfigurasi pada *file mapping_default.launch* yakni berupa posisi *scanning* pada *frame /base_link* dan posisi odom menjadi *frame /Odom*. Selanjutnya lakukan proses pengaturan laser *value* pada parameter *laser_z_min_value*, *laser_z_max_value*, *laser_min_dist*, *laser_max_dist*. Setelah konfigurasi dijalankan oleh program maka selanjutnya adalah menjalankan sub-proses, yaitu diagram alir estimasi pose 2D yang dapat dilihat pada Gambar 5.20.



Gambar 5.20 Diagram Alir Sub Proses Estimasi Pose 2D

Pada Gambar 5.20 dapat dilihat bahwa sub proses dimulai dengan sebuah proses yang mengatur *pose estimate*, selanjutnya proses terbagi menjadi 2 yaitu

pose dari iterasi sebelumnya dan pose menggunakan perkiraan awal *transformation*. Setelah itu dilanjutkan dengan proses titik akhir proyek ke peta berdasarkan pose saat ini dan dilanjutkan dengan melakukan perkiraan peta *occupancy probability* di titik akhir pemindaian laser. Selanjutnya dilakukan proses iterasi menggunakan metode *Gauss-Newton* untuk memperbaiki nilai estimasi pose yang salah dengan rumus pada persamaan 2.14 dan 2.15. Persamaan ini digunakan untuk melakukan koreksi nilai estimasi pose menggunakan iterasi. Iterasi ini nantinya digunakan sebagai perhitungan pergerakan robot berdasarkan banyaknya jumlah iterasi yang menghasilkan titik proyeksi pose robot untuk selanjutnya proses dikembalikan ke titik akhir proyek ke peta berdasarkan perkiraan pose. Proses estimasi pose berakhir yang kemudian kembali ke proses di dalam diagram alir pada Gambar 5.19.

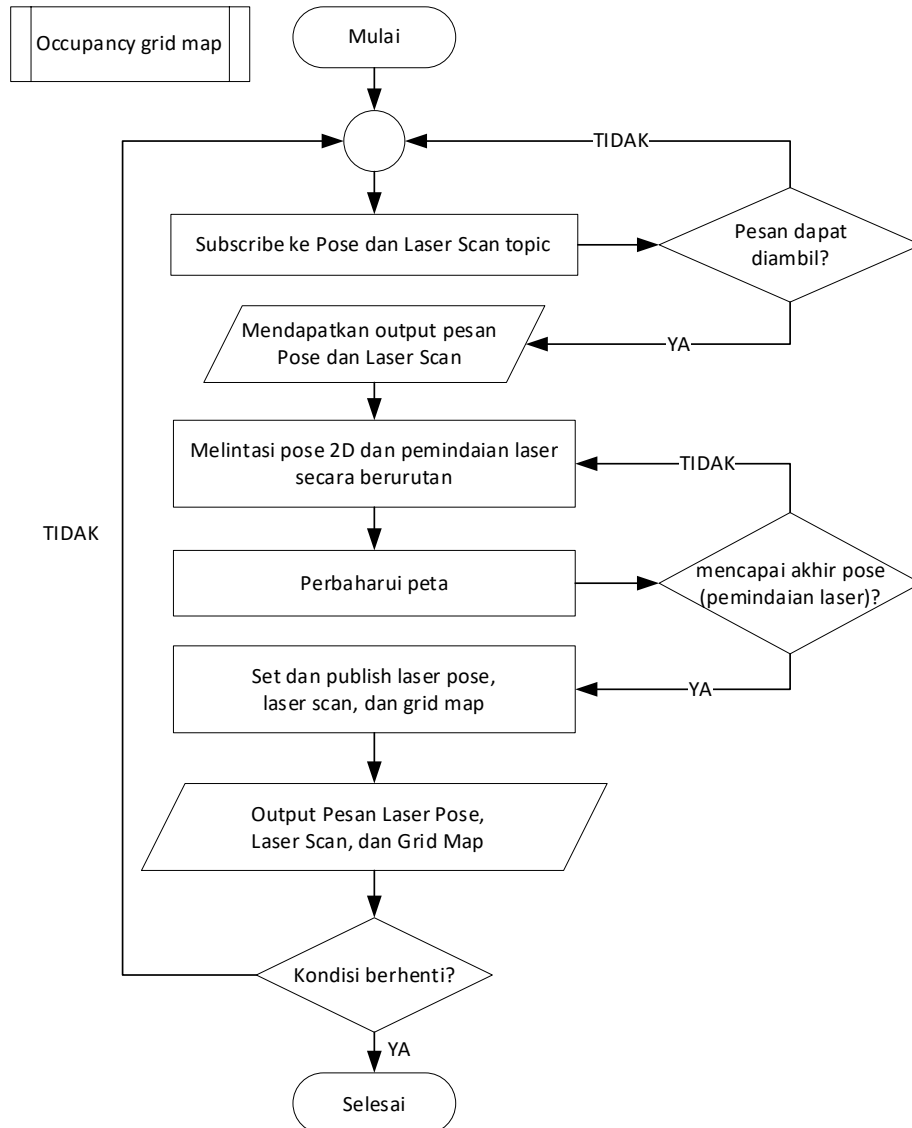
Tabel 5.8 Parameter Konfigurasi Paket Hector Mapping

No.	Parameter	Default	Konfigurasi Penelitianan
1	base_frame	"base_link"	"base_link"
2	map_frame	"map_link"	"map"
3	odom_frame	"odom"	"odom_combined"
4	map_resolution	0.025 meter	5.050 meter
5	map_size	1024	2048
6	map_start_x	0.5	0.5
7	map_start_y	0.5	0.5
8	map_update_distance_thresh	0.4 meter	0.4 meter
9	map_update_angle_thresh	0.9 rad	0.06 rad
10	map_pub_period	2.0 detik	2.0 detik
11	map_multi_res_levels	3	2
12	update_factor_free	0.4	0.4
13	update_factor_occupied	0.9	0.9
14	laser_min_dist	0.4 meter	0.4 meter
15	laser_max_dist	30.0 meter	30.0 meter
16	laser_z_min_value	-1.0 meter	-1.0 meter
17	laser_z_max_value	1.0 meter	1.0 meter

18	pub_map_odom_transform	true	false
19	output_timing	false	false
20	scan_subscriber_queue_size	5	5
21	pub_map_scanmatch_transform	true	false
22	tf_map_scanmatch_transform_frame_name	scanmatcher_frame	scanmathcer_frame
23	pub_odometry	false	true
24	advertise_map_service	false	true

Sub proses selanjutnya adalah *occupancy grid* map yang merupakan tambahan dari diagram alir yang dibuat oleh Aya Souliman. Proses ini memanfaatkan *output* yang dihasilkan proses estimasi pose 2D. Diagram alir algoritme *occupancy grid* diambil dari penelitian Xu, Lichao, dkk (Xu, dkk., 2019). Berdasarkan Gambar 5.20, diagram alir menunjukkan proses dimulai dari melakukan *subscribe* topik Pose dan Laser Scan, kemudian lakukan seleksi kondisi apakah pesan topik tersebut dapat diambil atau tidak. Jika tidak maka lakukan *subscribe* topik kembali dan lakukan proses yang sama, namun jika ya maka mendapatkan pesan *output* Pose dan Laser scan. Proses dilanjutkan dengan melintasi pose 2D dan pemindaian laser secara berurutan dan akhirnya bisa memperbaharui peta. Jika pose sudah mencapai titik akhir dari pemindaian laser, dalam arti laser ditembakkan kemudian bergerak mencapai posisi robot kembali maka dilakukan proses set dan *publish* transformasi laser pose, laser scan, dan *grid* map. Namun jika tidak maka proses dikembalikan ke perlintasan pose 2D dan pemindaian laser secara berurutan untuk nantinya bisa memperbaharui peta.

Setelah melakukan set dan *publish* topik, maka didapatkan *output* berupa pesan laser pose, laser scan, dan *grid* map. *Output* ini digunakan kembali ke proses awal yaitu *subscribe* pose dan laser scan topik sehingga algoritme terus berulang menciptakan proses *recursive*. Diagram alir proses ini dapat dilihat pada Gambar 5.21.



Gambar 5.21 Diagram Alir Sub Proses *Occupancy Grid Map*

Setelah proses *occupancy* selesai, maka kita sudah mendapatkan data *grid map* yang menjadi *output* dari proses *occupancy grid map*. Map ini digunakan sementara untuk nantinya disimpan dan digunakan pada proses navigasi. Kembali mengacu pada Gambar 5.18, ketika proses *mapping* diakhiri maka proses selesai, namun jika belum maka proses terus berlanjut hingga nantinya di *terminate* paksa melalui terminal. Hasil akhir perancangan dari diagram alir dari algoritme Hector SLAM berupa konfigurasi pada *file launcher*. Jadi untuk mencapai diagram alir tersebut, perlu dilakukan konfigurasi parameter terlebih dahulu pada paket *hector_slam* agar sesuai dengan yang diharapkan. Konfigurasi ini dapat dilihat pada Tabel 5.9 dan Tabel 5.10.

Tabel 5.9 Parameter Konfigurasi Paket Hector *Geotiff*

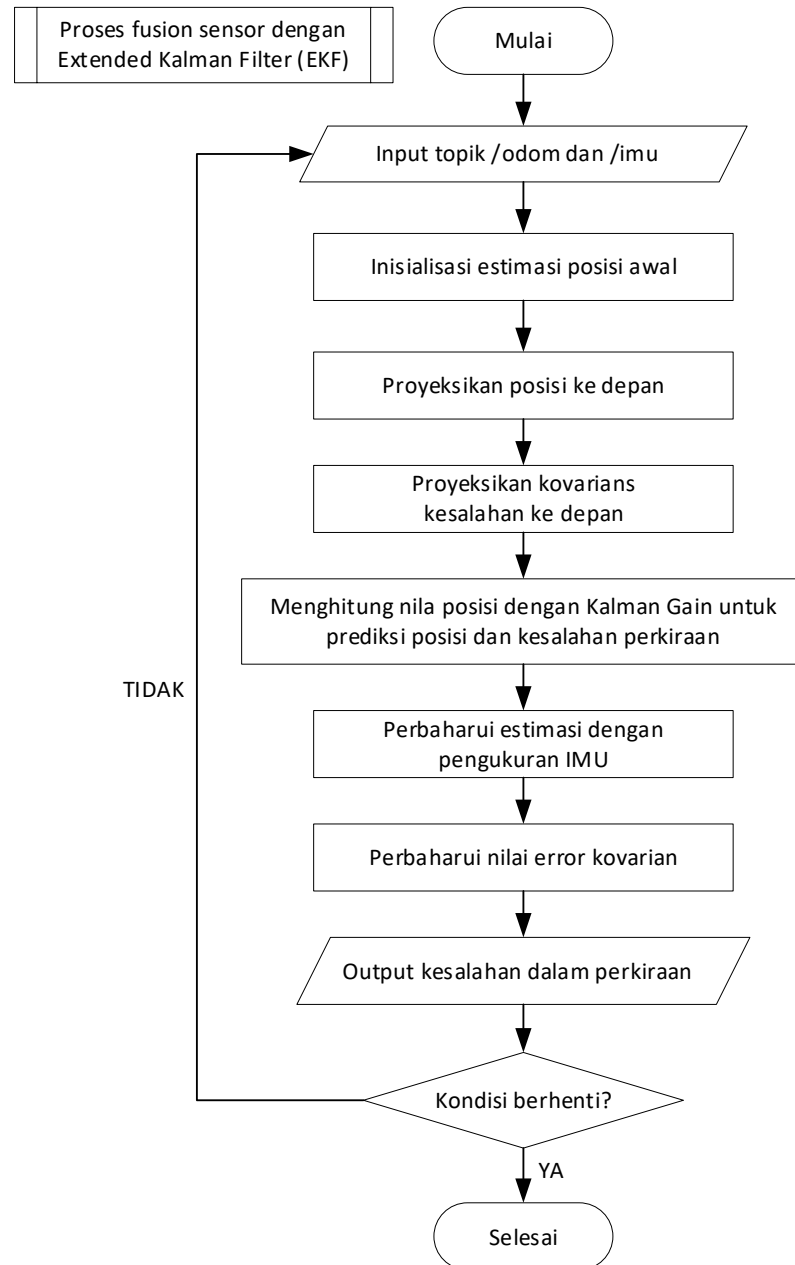
No.	Parameter	Default	Konfigurasi Penelitian
1	map_file_path	“.”	"\$(find hector_geotiff)/maps"
2	map_file_base_name	“GeoTiffMap”	"hector_slam_map"
3	geotiff_save_period	0	0
4	draw_background_checkerboard	false	true
5	draw_free_space_grid	false	true
6	plugins	“”	"hector_geotiff_plugins/ TrajectoryMapWriter"

Tabel 5.10 Parameter Konfigurasi Paket Hector *Trajectory*

No.	Parameter	Default	Konfigurasi Penelitian
1	target_frame_name	“map”	“/map”
2	source_frame_name	“base_link”	“/base_link”
3	trajectory_update_rate	4.0 Hz	4.0 Hz
4	trajectory_publish_rate	0.25 Hz	0.25 Hz

5.1.2.6 Perancangan Proses Algoritme Extended Kalman Filter (EKF)

Proses algoritme EKF digunakan untuk melakukan koreksi terhadap lokalisasi robot. Pada penelitian sebelumnya, algoritme Hector SLAM saja tidak mencukup untuk membaca posisi terkini robot dikarenakan area pengujian yang sangat luas, yaitu gudang sehingga pada saat laser tidak mendeteksi halangan sama sekali maka bisa menghasilkan pose *estimate* yang keliru. Maka dari itu digunakan algoritme EKF sekaligus untuk menggabungkan 2 buah *input* dari sensor IMU dan *odometry*. Diagram alir dari algoritme EKF dapat dilihat pada Gambar 5.22.



Gambar 5.22 Diagram Alir Proses Algoritme *Extended Kalman Filter* (EKF)

Sumber: (Wang, dkk., 2013)

Pada Gambar 5.22 dapat dilihat bahwa proses dari algoritme EKF melakukan proses secara bersamaan dengan asinkron. Dimulai dari melakukan perkiraan kesalahan asli, perkiraan asli, pengambilan *input* dari data IMU dan data *odometry*. Semua ini dilakukan secara bersamaan sehingga pada proses perkiraan kesalahan asli didapatkan nilai kesalahan dalam perkiraan dan kesalahan dalam data didapatkan dari perkiraan sebelumnya. Untuk proses perkiraan asli data diambil dari hasil pembaharuan perkiraan yang secara *default* pada awal mula proses dimulai perkiraan asli diambil dari nilai perkiraan awal berupa *covariant* sensor. Langkah selanjutnya adalah mendapatkan *input* dari data sensor IMU dan data

odometry melalui topik yang dihasilkan oleh masing-masing *driver*, lalu kemudian dilakukan proses pengukuran nilai.

Setelah itu proses berlanjut ke menghitung nilai dengan *Kalman Gain* untuk memprediksi posisi dan kesalahan perkiraan dengan *input* kesalahan dalam perkiraan dan kesalahan dalam data. Nilai prediksi ini dilanjutkan ke proses menghitung perkiraan saat ini yang juga mendapatkan *input* dari perkiraan sebelumnya. Setelah menghitung perkiraan saat ini, proses dilanjutkan dengan proses memperbaharui perkiraan yang hasil akhirnya berupa *output frame* transformasi EKF dan topik baru berupa */odom_combined* yang didapat berdasarkan posisi *odometry* dan IMU pada robot. Proses memperbaharui perkiraan dikembalikan ke proses perkiraan asli sehingga terjadi proses *recursive*.

Setelah melakukan proses algoritme Hector SLAM dan EKF maka dari masing-masing algoritme menghasilkan *output*. Algoritme *hector* SLAM memperbaharui topik */scan* sehingga bisa digunakan untuk proses *occupancy grid* pada aplikasi RViz, selain itu algoritme ini juga menghasilkan *frame* penggabungan transformasi map dengan *odometry* yang diberi nama “map_to_odom”. Selain itu juga menghasilkan *frame* *hector_mapping* dan *hector_trajectory* yang dimanfaatkan untuk membuat map. Sedangkan algoritme EKF menghasilkan topik baru dengan nama */odom_combined*, begitu pula dengan *frame* transformasinya dengan nama *odom_combined*. Semua hasil lokalisasi yang berupa posisi robot dan transformasinya digunakan untuk melakukan *update* posisi terkini yang pada akhirnya dikirimkan ke RViz dan *output* topik */map*. Dengan gabungan dua algoritme ini menghasilkan transformasi robot yang lebih akurat dan proses *mapping* berjalan dengan lancar. *Output* topik */map* tadi yang berupa peta dalam bentuk sementara disimpan menjadi *output file* baru dengan ekstensi *yaml* dan *pgm* yang nantinya digunakan pada proses navigasi. Namun hal ini berlaku jika proses map sudah selesai, jika belum maka proses dikembalikan ke algoritme Hector SLAM hingga semua area sudah di *mapping*.

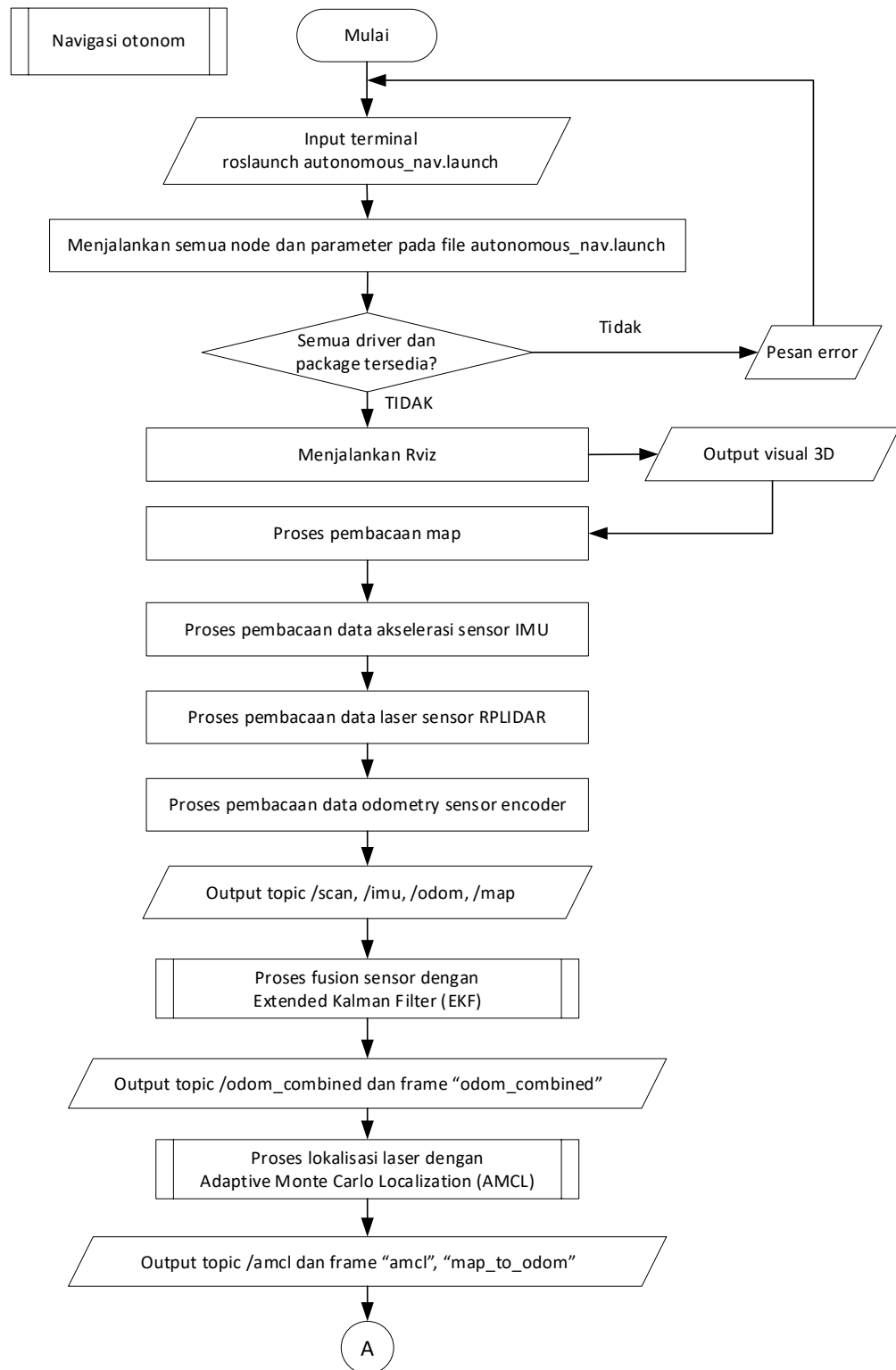
Ini adalah keseluruhan dari proses *mapping*, secara garis besar dapat diperhatikan kembali pada Gambar 5.16. *Output* dari proses ini yang berupa *file* map digunakan pada program selanjutnya yaitu navigasi. Pada program navigasi terdapat pula algoritme EKF dengan proses yang sama persis, maka dari itu pada sub proses Algoritme EKF di program navigasi tidak dijelaskan karena sudah dijelaskan pada proses *mapping*. Konfigurasi EKF *Localization* dapat dilihat pada dan Tabel 5.11.

Tabel 5.11 Parameter Konfigurasi Paket EKF *Localization*

No.	Parameter	Default	Konfigurasi Penelitian
1	output_frame	“odom”	“odom”
2	base_footprint_frame	“base_footprint”	“base_footprint”
3	freq	30 Hz	100 Hz

4	sensor_timeout	1.0 detik	1.0 detik
5	odom_used	true	true
6	imu_used	true	true
7	vo_used	true	false
8	gps_used	false	false
9	debug	false	false
10	self-diagnose	false	false

5.1.2.7 Perancangan Program Navigasi

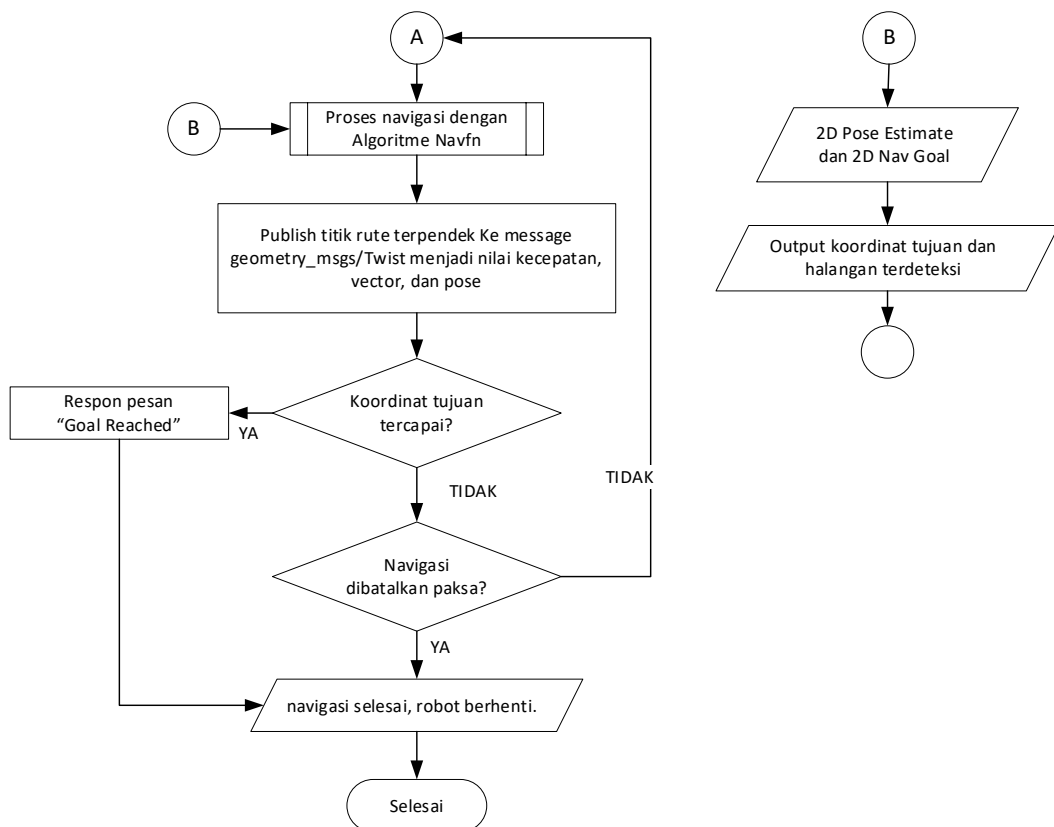


Gambar 5.23 Diagram Alir Program Navigasi Bagian A

Diagram alir dari perancangan keseluruhan program ditunjukkan pada Gambar 5.23. Pada gambar tersebut kita dapat melihat bahwa terdapat banyak proses

yang saling terkait satu dengan yang lain. Diagram alir ini menjelaskan mengenai *node-node* yang dibutuhkan dalam proses navigasi, seperti menjalankan semua *driver* sensor kemudian membaca sensor, penggabungan sensor teknik *fusion* sensor, lokalisasi, *transform frame*, dan parameter-parameter yang digunakan. Semua hal ini dijadikan satu proses besar yang diberi nama *autonomous_nav*. Semua proses yang terjadi di dalam program ini digabungkan ke dalam *file* dengan ekstensi *launch*.

Proses awal pada navigasi kurang lebih sama seperti proses *mapping*, yakni dimulai dari melakukan *input* pada *terminal* berupa perintah `$ roslaunch autonomous_nav.launch`. Maka selanjutnya proses berpindah ke *terminal* dengan menjalankan semua *node* dan parameter pada *file launcher* tersebut. Dilanjutkan dengan membuka program RViz dan melakukan pengecekan pada semua *driver* dan paket yang digunakan, jika didapati adanya kesalahan maka mengirimkan pesan *error* dan program di *terminate* untuk nantinya dievaluasi kembali mengenai pesan *error* tersebut. Namun jika semua *driver* dan paket sudah tersedia maka dilanjutkan dengan proses paralel dengan melakukan pembacaan pada masing-masing sensor yang nantinya digunakan untuk proses *subscribe*.

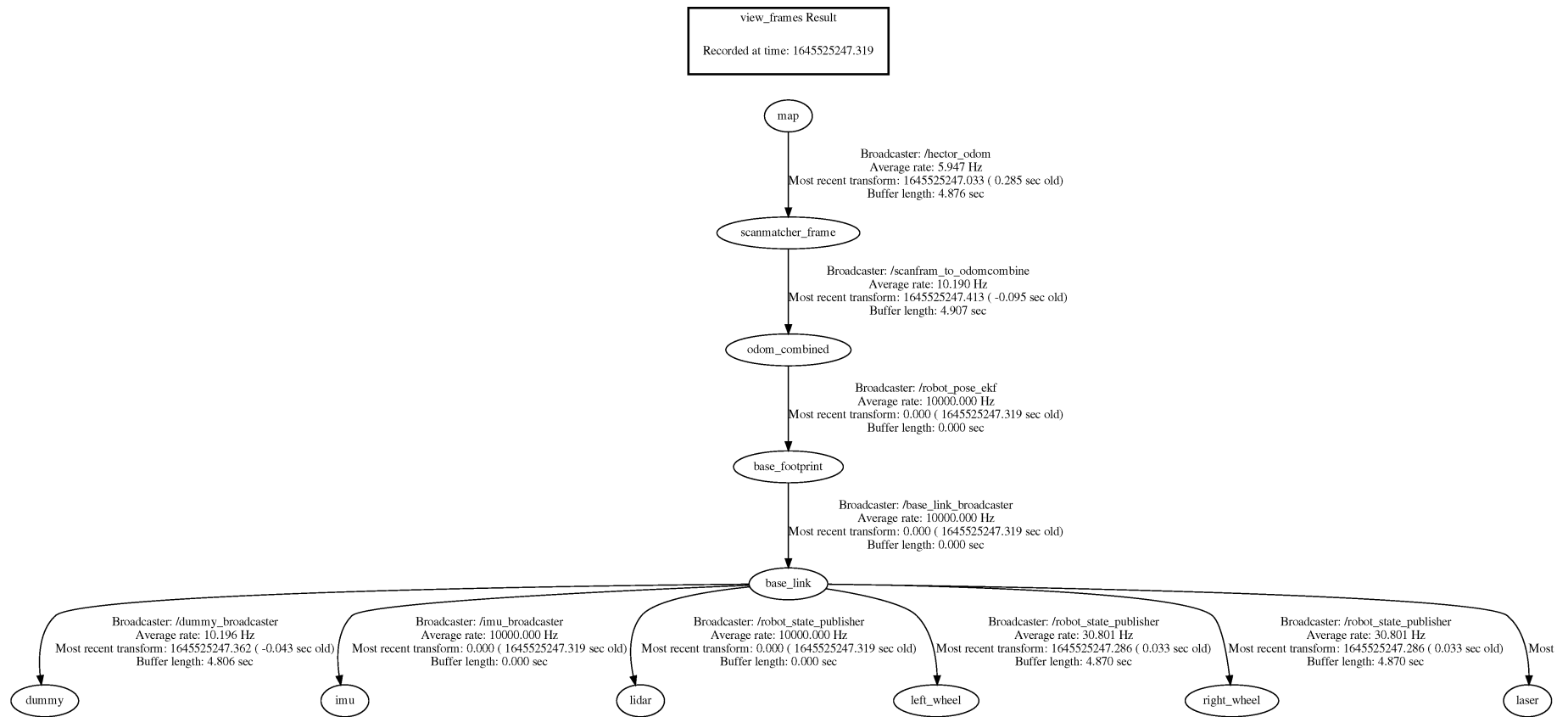


Gambar 5.24 Diagram Alir Program Navigasi

Pada aplikasi RViz terdapat dua *tool* utama yang digunakan untuk melakukan navigasi, yaitu *2D Pose Estimate* dan *2D Nav Goal*. *2D Pose estimate* digunakan untuk mengkonfirmasi ulang posisi robot berdasarkan posisi robot sebenarnya di dunia nyata. Sedangkan untuk *2D Nav Goal* berguna untuk memberikan *input*

berupa koordinat tujuan robot yang digunakan untuk *input* awal proses *path planning*. Untuk melakukan navigasi diperlukan 2 *input* ini, terutama *input* dari 2D *Nav Goal*. Kembali ke proses pembacaan data sensor, nilai pemindaian laser diolah oleh *driver* RPLIDAR sehingga didapatkan *output* topik baru berupa */scan* yang merupakan data *point cloud*. Selanjutnya pada *input* akselerasi robot dibaca oleh sensor IMU yang diproses menggunakan *driver* IMU sehingga menghasilkan *output* topik baru berupa */imu*.

Ketika pertama kali dijalankan, setiap komponen membuat *frame* transformasi pada *node* masing-masing. Setiap *frame* tersebut harus saling terhubung dan terkoordinasi, perancangan yang diinginkan adalah seperti yang terlihat pada Gambar 5.25.

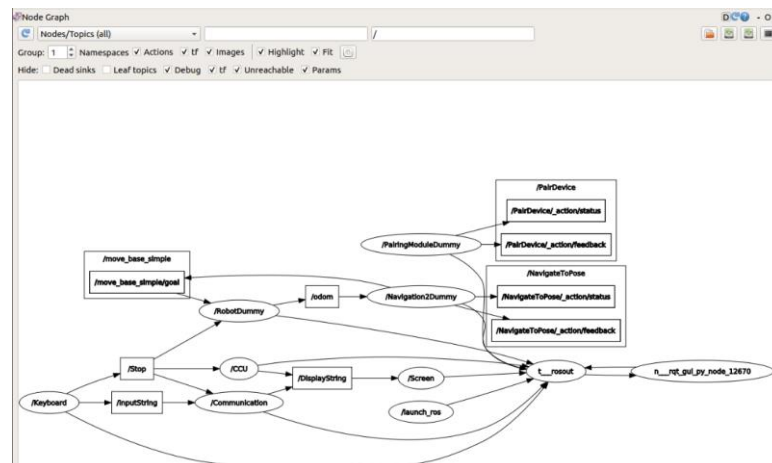


Gambar 5.25 Transformasi *Frame* Pada Navigasi

Pada Gambar 5.25 dapat terlihat bahwa setiap *node* dengan *broadcaster* saling menghubungkan sehingga membuat diagram alir yang sejajar. Untuk melakukan hal ini diperlukan konfigurasi *static* transformasi (TF) pada *file launcher* di program pemetaan. Terdapat beberapa keterangan seperti *broadcaster* yang merupakan *node* pengirim atau *publisher output* sebuah TF, *average rate* yang merupakan rata-rata pengiriman dalam Hertz *Flow* dimulai dari *node* map, *node* ini didapat dari *publisher node* *hector_mapping*. Selanjutnya TF tersebut dilanjutkan ke *odom_combined* yang di *broadcast* oleh algoritme EKF.

Selanjutnya pada *input* nilai putaran roda dibaca oleh *driver hoverboard* yang menghasilkan topik baru berupa */odom* yang merupakan data posisi roda dan orientasinya. Terdapat tambahan berupa *input* dari *file* map yang disimpan di folder tertentu dengan nama *maps*. *File* ini di *generate* oleh paket map server dan dikirimkan ke dalam sub proses algoritme AMCL sebagai *input* awalnya begitu pula dengan data yang didapat dari sensor RPLIDAR menggunakan topik */scan*. Sedangkan data dari sensor IMU dan *odometry* di kirimkan ke sub proses *Fusion* Sensor EKF menggunakan topik */imu/data* dan */odom*.

Kemudian dilakukan seleksi kondisi jika koordinat tujuan sudah ditentukan maka memberikan pesan global *path planning* ke *move base*, namun jika tidak maka proses kembali di ulang ke awal sampai nantinya koordinat tujuan sudah ditentukan. Ketika koordinat tujuan sudah ditentukan maka mengirimkan pesan global *path planning* ke *move base*, sehingga memerintahkan global *path planning* untuk menentukan jalur terpendek menggunakan Algoritme Navfn. Algoritme Navfn mendapatkan 3 *input* sekaligus dan mengolahnya ke dalam sub proses Algoritme Navfn. Sebagai tambahan, penggunaan algoritme global dapat diubah dengan mengganti *node* global *planner* di dalam kode paket *move base*. Penjelasan lebih detail dan diagram alir mengenai sub proses algoritme Navfn dapat dilihat pada Gambar 5.26.



Gambar 5.26 Grafik Semua Node Pada Proses Navigasi

Konfigurasi paket *move base* untuk menggerakkan robot dapat dilihat pada dan Tabel 5.12.

Tabel 5.12 Parameter Konfigurasi Paket *Move Base*

No.	Parameter	Default	Konfigurasi Penelitian
1	base_global_planner	"navfn/NavfnROS"	" navfn/NavfnROS "
2	base_local_planner	"base_local_planner/ TrajectoryPlannerROS "	"dwa_local_planner / DWAPlanerROS"
3	recovery_behaviors	[]	[]
4	controller_frequency	20 Hz	5 Hz
5	planner_patience	5.0 detik	5.0 detik
6	controller_patience	15.0 detik	15.0 detik
7	conservative_reset_dist	3.0 meter	3.0 meter
8	recovery_behavior_enabled	true	true
9	clearing_rotation_allowed	true	true
10	shutdown_costmaps	false	false
11	oscillation_timeout	0.0 detik	0.0 detik
12	oscillation_distance	0.5 meter	0.5 meter
13	planner_frequency	0 Hz	0 Hz
14	max_planning_retries	-1	0

Navigasi membutuhkan *costmap* untuk memisahkan jalur dengan halangan, berikut konfigurasi *common_costmap.yaml* pada dapat dilihat pada Tabel 5.13.

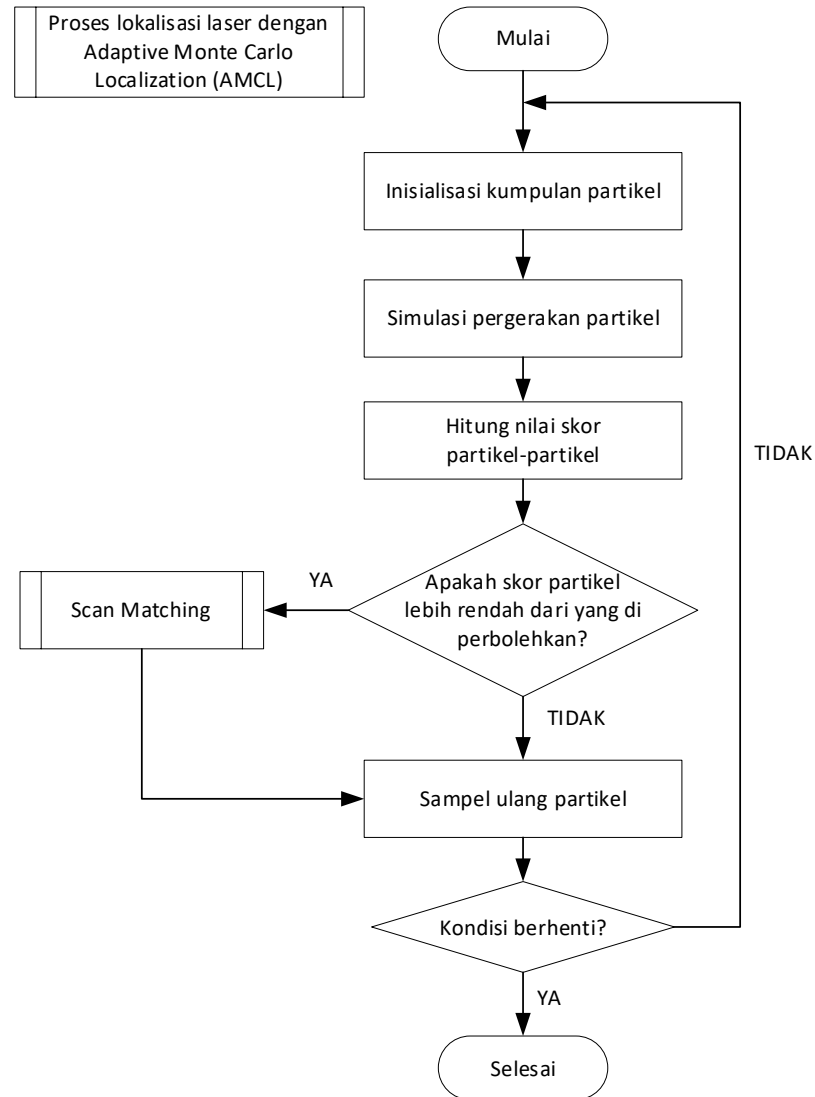
Tabel 5.13 Parameter Konfigurasi 2D *Common Costmap*

No.	Parameter	Default	Konfigurasi Penelitian
1	global_frame	"/map"	"/map"
2	robot_base_frame	"base_link"	"base_footprint"
3	transform_tolerance	0.2	0.2
4	update_frequency	5.0 Hz	5.0 Hz
5	publish_frequency	5.0 Hz	5.0 Hz

6	rolling_window	false	false
7	always_send_full _costmap	false	false
8	width	10 meter	10 meter
9	height	10 meter	10 meter
10	resolution	0.05 meter/cell	0.05 meter/cell
11	origin_x	0.0 meter	origin_y meter

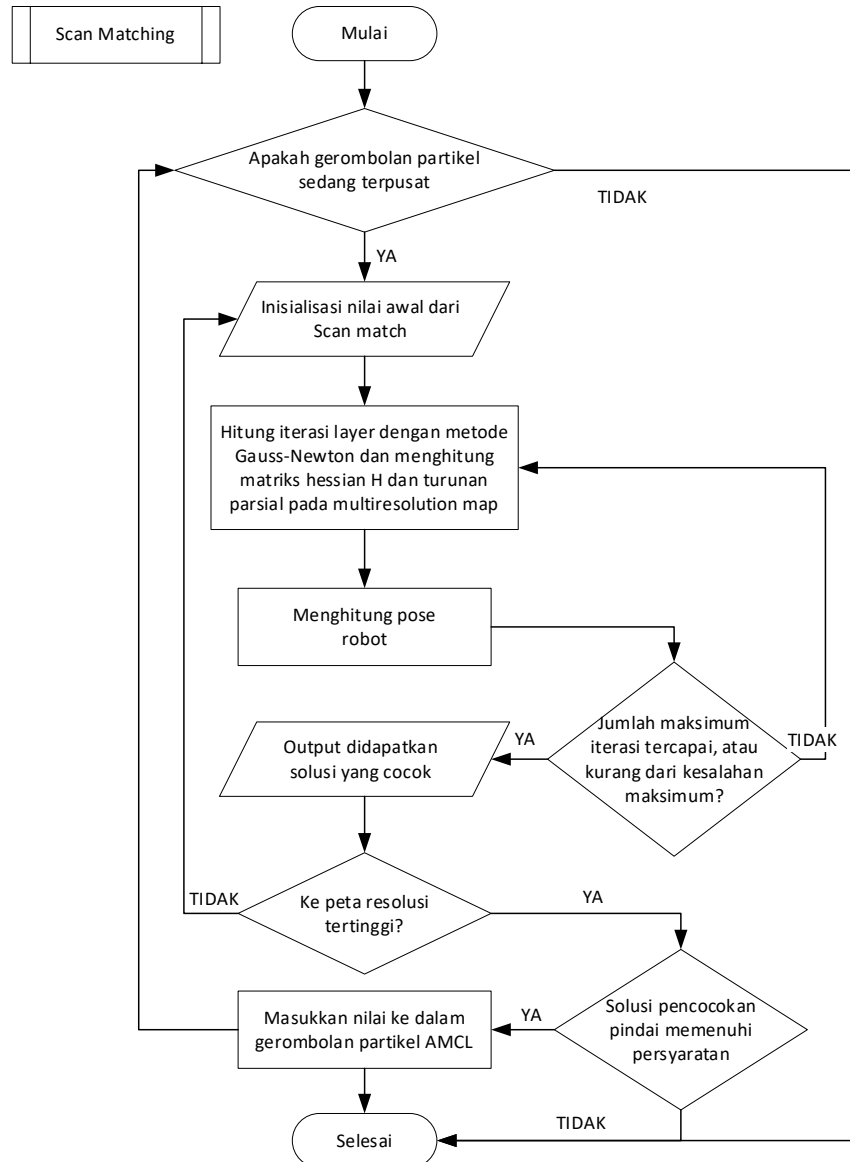
5.1.2.8 Perancangan Proses Algoritme *Adaptive Monte Carlo Localization* (AMCL)

Proses algoritme AMCL digunakan untuk melakukan koreksi terhadap lokalisasi robot. AMCL memanfaatkan *file* map yang sudah dibuat di program *mapping* sebelumnya. Detail mengenai algoritme ini dirangkum ke dalam diagram alir yang dapat dilihat pada Gambar 5.27.



Gambar 5.27 Diagram Alir Algoritme *Adaptive Monte Carlo Localization* (AMCL)

Pada Gambar 5.27 dapat dilihat bahwa proses diagram alir terdiri dari beberapa proses dan seleksi kondisi. Diagram alir ini dikutip dari penelitian yang dilakukan oleh Gang Peng dan kawan-kawan (Peng, dkk., 2018). Dalam penelitian tersebut disebutkan bahwa untuk memecahkan masalah selama algoritma AMCL untuk memperkirakan pose robot sebagai deviasi pusat tertimbang dari kawanan partikel di lingkungan tidak terstruktur yang kompleks, penelitian ini menambahkan pencocokan pemindaian yang cocok dengan pemindaian laser dengan peta. Proses yang terjadi dalam algoritme AMCL adalah kumpulan partikel (*point cloud*) yang didapat dari data laser. Kemudian dilakukan simulasi pergerakan partikel sebagai bentuk pose *estimate*. Selanjutnya dilakukan perhitungan nilai skor tiap partikel. Setelah dilakukan perhitungan skor tiap partikel, dilakukan proses *scan matching*, proses ini menggunakan rumus matriks *Hessian* dan gabungan dari metode *Gaussian-Newton iteration* untuk melakukan perkiraan menggantikan model regresi non-linier. Detail mengenai proses ini dapat dilihat pada Gambar 5.28.



Gambar 5.28 Diagram Alir Sub Proses AMCL: Scan Matching

Pada Gambar 5.28 dapat dilihat bahwa setelah memasuki proses *scan matching*, pertama dilakukan seleksi kondisi terlebih dahulu untuk mengetahui apakah gerombolan partikel (*particle cloud*) sedang dalam posisi terpusat atau terkonsentrasi (solusi dari estimasi pose konvergen). Jika kondisinya adalah tidak, maka proses langsung di *terminate* dan proses *scan matching* sudah selesai, sehingga berlanjut ke proses sampel ulang partikel. Hal ini dikarenakan algoritme *scan matching* digunakan untuk lebih mengoptimalkan pose. Sehingga hanya memproses data-data *point cloud* yang terkonsentrasi saja. Metode ini merupakan bentuk dari koreksi data laser yang *error* dan tidak bisa digunakan untuk menentukan posisi.

Setelah inisialisasi nilai dari *scan match*, maka langkah selanjutnya adalah dengan melakukan kalkulasi iterasi *Gaussian-Newton* dilakukan *layer per layer* dari resolusi rendah hingga ke resolusi tinggi pada peta *multiresolution*. Jika Jumlah

maksimum iterasi tercapai, atau kurang dari kesalahan maksimum maka didapatkan *output* berupa solusi yang cocok. Setelah sebelumnya melakukan iterasi dan mendapatkan pose yang lebih akurat dari sebelumnya. Dilakukan seleksi kondisi untuk menyeleksi apakah hasil dari perhitungan sudah merupakan hasil dari peta resolusi tertinggi, jika belum maka artinya *error* yang dihasilkan lebih besar dari *threshold* yang didapat, ini berarti terdapat solusi *scan matching* yang *error* dan itu harus dibatalkan. Setelah dibatalkan maka dikembalikan ke proses awal yaitu inisialisasi nilai awal dari *scan match*.

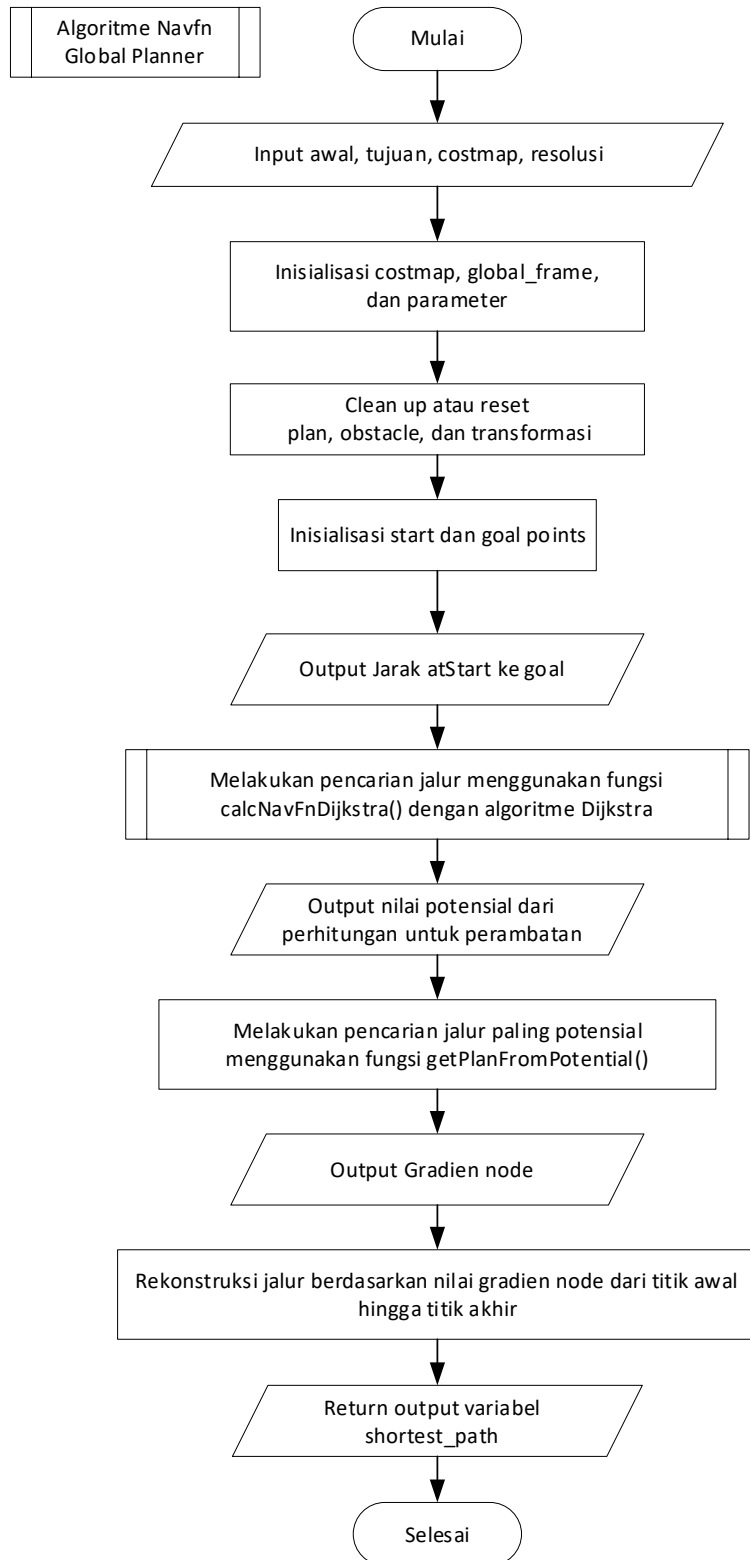
Namun jika peta sudah memiliki resolusi tertinggi maka, dilakukan solusi pencocokan pemindaian untuk memenuhi persyaratan. Jadi jika hasil sangat berbeda dari pose sebelumnya maka hasil perhitungan diabaikan dan proses *scan matching* di *terminate* ke selesai. Namun jika persyaratan sudah memenuhi kriteria dan tidak terlalu melebihi nilai sebelumnya, maka data dimasukkan ke dalam gerombolan *particle* AMCL dan hasil dari proses filtrasi data *point cloud* dikembalikan ke seleksi kondisi awal, jika terdapat gerombolan partikel yang terpusat kembali, maka lakukan kalkulasi ulang, begitu seterusnya hingga terbentuk proses yang *recursive*. Konfigurasi AMCL *Localization* dapat dilihat pada dan Tabel 5.14.

Tabel 5.14 Parameter Konfigurasi Paket AMCL *Localization*

No.	Parameter	Default	Konfigurasi Penelitian
1	min_particles	100	500
2	max_particles	5000	5000
3	kld_err	0.01	0.01
4	kld_z	0.99	0.99
5	update_min_d	0.2 meter	0.1 meter
6	update_min_a	$\pi/6.0$ radian	0.2 radian
7	resample_interval	2	2
8	transform_tolerance	0.1 detik	0.1 detik
9	recovery_alpha_slow	0.0	0.0
10	recovery_alpha_fast	0.0	0.0
11	initial_pose_x	0.0	0.0
12	initial_pose_y	0.0	0.0
13	initial_pose_a	0.0	0.0
14	initial_cov_xx	0.5*0.5 meter	0.25 meter

15	initial_cov_yy	0.5*0.5 meters	0.25 meters
16	initial_cov_aa	$(\pi/12)*(\pi/12)$ rad	0.0685 rad
17	gui_publish_rate	-1.0 Hz	-1.0 Hz
18	save_pose_rate	0.5 Hz	0.5 Hz
19	use_map_topic	false	false
20	first_map_only	false	false
21	selective_resampling	false	false
22	laser_min_range	-1.0	-1.0
23	laser_max_range	-1.0	-1.0
24	laser_max_beams	30	30
25	laser_z_hit	0.95	0.95
26	laser_z_short	0.1	0.1
27	laser_z_max	0.05	0.05
28	laser_z_rand	0.05	0.05
29	laser_sigma_hit	0.2 meter	0.2 meter
30	laser_lambda_short	0.1	0.1
31	laser_likelihood_max_dist	2.0 meter	2.0 meter
32	laser_model_type	"likelihood_field"	"likelihood_field"
33	odom_model_type	"diff"	"diff-corrected"
34	odom_alpha1	0.2	0.2
35	odom_alpha2	0.2	0.2
36	odom_alpha3	0.2	0.2
37	odom_alpha4	0.2	0.2
38	odom_alpha5	0.2	0.2
39	odom_frame_id	"odom"	"odom"
40	base_frame_id	"base_link"	"base_link"
41	global_frame_id	"map"	"map"
42	tf_broadcast	true	true

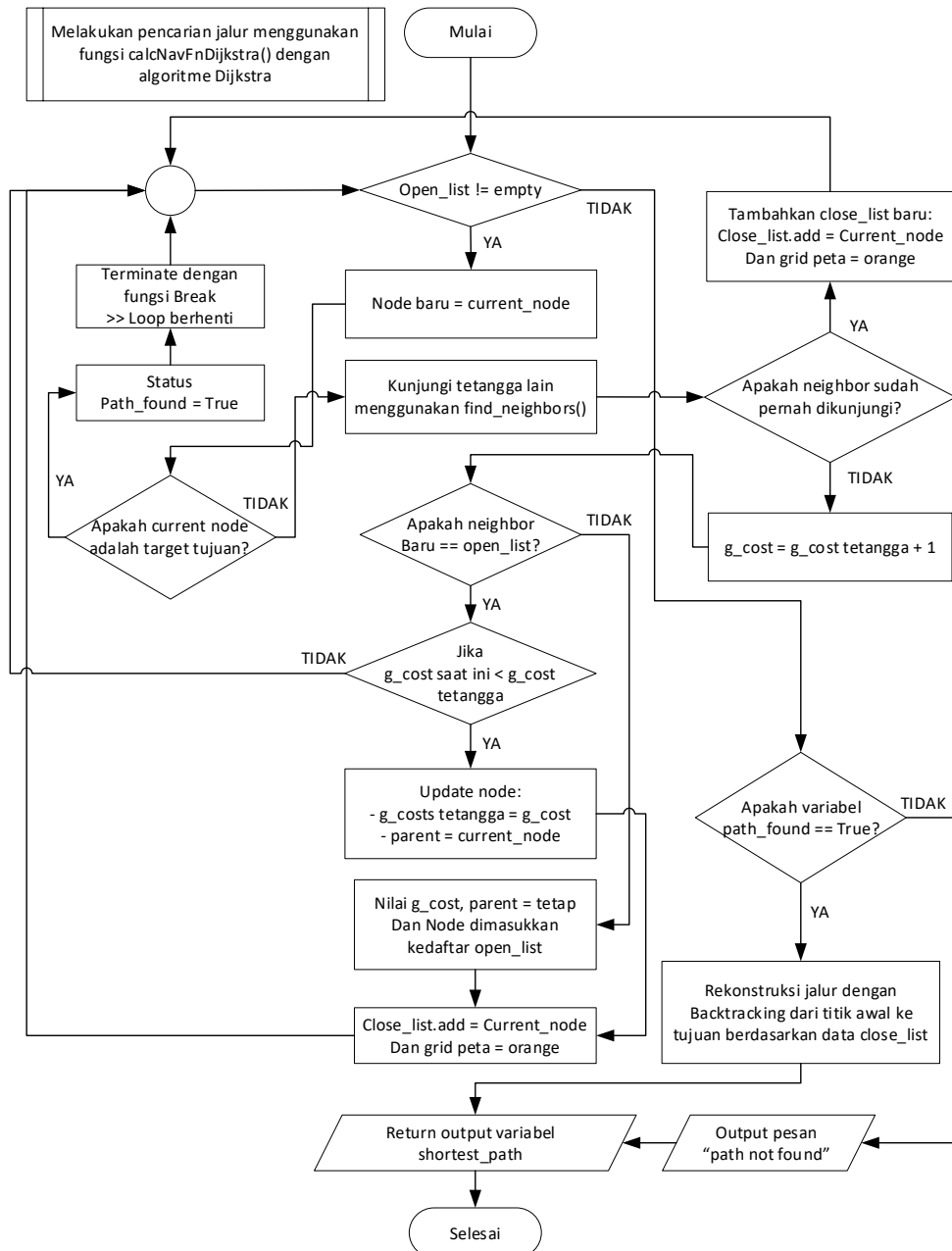
5.1.2.9 Perancangan Proses Algoritme Navfn



Gambar 5.29 Diagram Alir Algoritme Navfn

Pada Gambar 5.29 dapat dilihat bahwa berdasarkan diagram alir, algoritme Navfn dimulai dari memasukkan *input* awal, tujuan, *costmap*, dan resolusi ke

dalam parameter fungsi ini. Dan dapat diperhatikan juga bahwa algoritme ini memanggil *class NavfnROS* yang terintegrasi dengan *library ROS*. Selanjutnya menjalankan fungsi proses *clean up* plan, *obstacle*, dan TF untuk memastikan bahwa setiap perencanaan jalur yang dibuat sudah bersih dari data sebelumnya. Kemudian dilakukan inisialisasi *starting* dan *goal point* untuk dan memanggil fungsi *calcNavFnDijkstra* dari kelas *NavfnROS*, sehingga dapat menyelesaikan penghitungan jalur global. Berikut penjelasan mengenai sub proses dari *Dijkstra* yang dapat dilihat pada Gambar 5.30.



Gambar 5.30 Diagram alir penggunaan *Dijkstra* pada Navfn

Pada Gambar 5.30 hal pertama yang dilakukan adalah melakukan *looping* dengan kondisi, hanya akan berhenti jika *array* *open_list* bernilai kosong. Logika

ini digunakan untuk melakukan pencarian ke seluruh area *free space* yang ada, jadi selama masih terdapat nilai pada *array* *open_list* maka proses pencarian terus berlanjut hingga target ditemukan. Jika hasilnya *true* maka program mengubah status *node* baru menjadi *current_node*, setelah itu dilakukan seleksi kondisi apakah *current_node* merupakan titik koordinat target. Jika *true* maka status *path_found* diubah menjadi *True* dan mengirimkan *break* untuk menghentikan *looping*. Namun jika tidak maka dilakukan pencarian ke seluruh tetangga terdekat dari *current_node* sebanyak 8 *node* berdasarkan maksimal sisi yang bisa diraih dalam 1 *node*. Setelah melakukan kunjungan ke *node* tetangga, maka lakukan seleksi kondisi untuk memeriksa apakah tetangga sudah pernah dikunjungi. Jika hasilnya *true*, maka masuk ke proses penambahan jumlah *close_list* berdasarkan indeks *current_node* untuk mencegah mengunjungi *node* ini lagi dan ubah *grid* peta berdasarkan indeks *current_node* menjadi *orange*. Jika hasilnya *false*, maka lakukan proses untuk *update* nilai *g_cost* baru dengan menambahkan *g_cost* tetangga + 1. Selanjutnya lakukan seleksi kondisi untuk mengetahui apakah status *neighbor* atau tetangga adalah *open_list*, jika hasilnya *false* maka nilai *g_cost* dan *parent* pada *current_node* tetap atau tidak berubah dan *node* dimasukkan ke dalam daftar *open_list* yang kemudian dilanjutkan dengan proses penambahan *array* indeks *close_list* berdasarkan posisi *current_node* dan *grid* peta menjadi *orange*. Setelah itu proses dikembalikan ke *open_list* lagi untuk melakukan pengulangan selama masih terdapat data pada *open_list*.

Namun jika *neighbor* yang baru merupakan *open_list* (indeksnya *open_list*) maka memasuki seleksi kondisi lagi, yakni jika *g_cost* saat ini kurang dari (<) *g_cost* tetangga, maka semua tetangga diberikan nilai *g_cost* yang baru dengan melakukan *update* nilai *g_cost* tetangga = *g_cost* yang sudah ditambah 1. Dengan begini tetangga yang baru dikunjungi mendapatkan *cost+1* dan *parent* status dari *current_node*. Setelah itu proses dilanjutkan dengan memasukkan tetangga yang sudah di *update* tadi ke *close_list* dan ubah *grid* peta menjadi *orange* kemudian proses kembali lagi ke pengulangan *open_list*. Proses ini terus berlangsung hingga *current_node* mencapai titik koordinat tujuan, titik diketahui karena pada parameter Navfn terdapat variabel *goal* yang berisi letak koordinat dalam matriks 2 dimensi, yaitu berupa nilai koordinat x dan y. Jika hal ini terjadi maka memasuki proses penemuan jalur dengan melakukan perubahan status pada variabel *path_found* = *True* dan fungsi *break* yang mampu menghentikan *looping* atau pengulangan. Perubahan status pada variabel *path_found* ditujukan untuk fase rekonstruksi jalur dengan proses *backtracking* atau penelusuran jalur secara terbalik dari titik awal menuju titik tujuan berdasarkan data yang ada di *close_list*. Jika status *path_found* == *false* yang artinya jalur tidak ditemukan sama sekali, maka melakukan skip pada fase rekonstruksi dan mengembalikan nilai berupa *output* dengan pesan "*path not found*".

Adapun hasil yang didapatkan mengembalikan nilai *output* berupa variabel *shortest_path* proses algoritme Navfn telah selesai. Navigasi membutuhkan global *planner* untuk menentukan jalur, berikut konfigurasi *global_planner.yaml* untuk membuat *path* yang dapat dilihat pada dan Tabel 5.15

Tabel 5.15 Parameter Konfigurasi Navfn Global Planner

No.	Parameter	Default	Konfigurasi Penelitian
1	allow_unknown	true	true
2	planner_window_x	0.0	1.0
3	planner_window_y	0.0	1.0
4	default_tolerance	0.0	0.0
5	visualize_potential	false	true

5.2 Implementasi Sistem

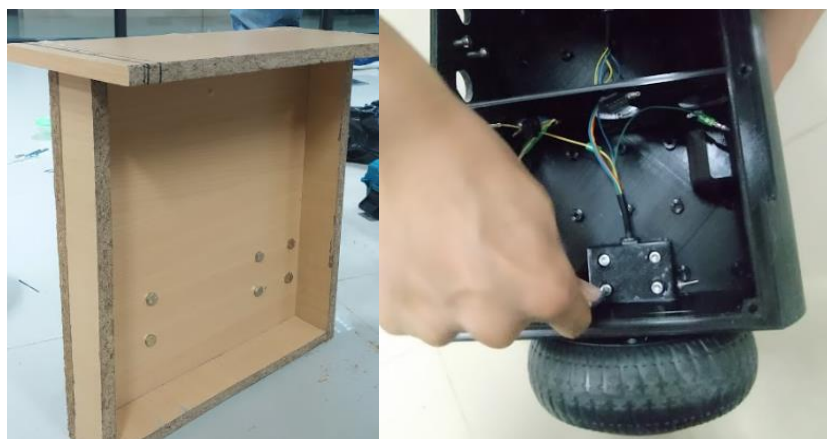
Implementasi sistem merupakan realisasi dari tahapan perancangan yang telah disusun sebelumnya untuk diimplementasikan secara riil. Dalam implementasi sistem ini terbagi menjadi 2 bagian yaitu implementasi perangkat keras dan implementasi perangkat lunak. Pada tiap proses implementasi dijelaskan terkait proses untuk pengaplikasian secara langsung menjadi sebuah satu sistem yang utuh.

5.2.1 Implementasi Perangkat Keras

Implementasi perangkat keras merupakan bagian dari merealisasikan hasil dari perancangan perangkat keras sebelumnya. Semua komponen perangkat keras yang telah dirancang dipastikan berfungsi atau tidaknya dalam mencapai tujuan yang diinginkan. Implementasi pada perangkat keras menggunakan *microcomputer* dan personal komputer sebagai media pendukung dalam pembuatan. Personal komputer juga dijadikan sebagai komputer server yang memantau dan menampilkan *output* dari sistem. Hal pertama yang ditunjukkan pengimplementasiannya adalah proses pembuatan sistem *mainboard*/penggerak robot.

5.2.1.1 Implementasi Perangkat Keras Chassis

Sesuai dengan perancangan perangkat keras *chassis* pada subbab sebelumnya, *chassis* robot terbagi menjadi dua buah bagian, yakni badan robot dengan kayu dan *chassis* sistem penggerak robot dengan cetak 3D. Hasil dari perancangan perangkat keras *chassis* dapat dilihat pada Gambar 5.31.



Gambar 5.31 Hasil Implementasi *Chassis* Robot

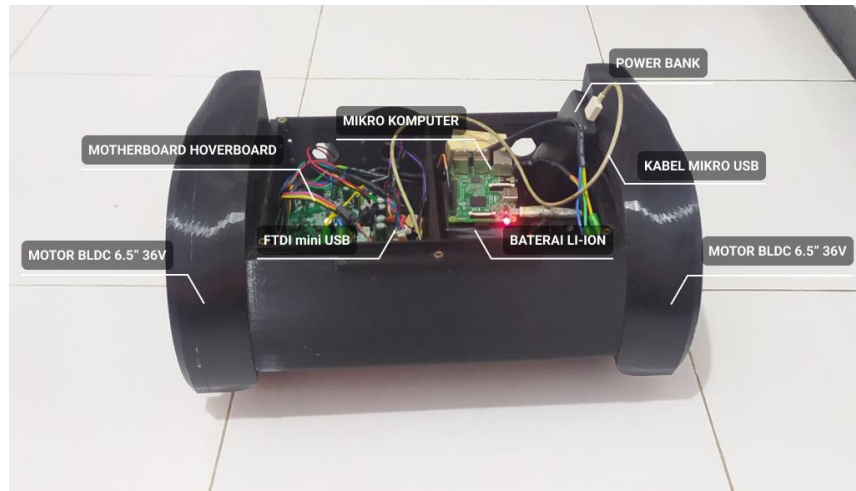
5.2.1.2 Implementasi Perangkat Keras Utama

Sesuai dengan perancangan, komponen *mainboard*/penggerak pada penelitian tidak menggunakan bagian yang terpisah, melainkan menggunakan perangkat elektronik yang dijual secara massal yaitu *hoverboard*. Komponen *motherboard*, baterai Li-ion, dan motor BLDC diambil dari perangkat *hoverboard* tersebut. Jenis *hoverboard* yang dipakai adalah *hoverboard* dengan lebar roda 6.5 *inch* dan memiliki *single motherboard*. Pembuatan sistem ini dimulai dengan melakukan percobaan menggunakan *motherboard* dengan komputer personal terlebih dahulu tanpa menggunakan *microcomputer*, Jetson Nano. Tujuan dari percobaan ini adalah untuk memastikan bahwa roda dapat berputar dengan kendali dari *output* komputer. Selain memastikan roda dapat berputar, dalam implementasinya proses ini digunakan untuk memasang *firmware* baru ke dalam *motherboard*. Pemasangan kabel untuk melakukan *flashing motherboard* ditunjukkan pada Tabel 5.16.

Tabel 5.16 Pemasangan Kabel Untuk *Flashing Motherboard*

Perangkat Keras	Pin	<i>Motherboard</i>	Komputer
ST-LINK V2 mini	3.3 V	3.3 V MCU	Port USB
	SWCLK	SWCLK MCU	
	GND	GND MCU	
	SWDIO	SWDIO MCU	

Seperti yang sudah dijelaskan dalam perancangan perangkat keras sistem, bahwa sistem *mainboard*/penggerak robot terdiri dari *motherboard* dan baterai yang terhubung dengan komputer menggunakan FTDI mini USB. Hasil akhir dari implementasi sistem penggerak robot dapat dilihat pada Gambar 5.32.



Gambar 5.32 Implementasi Sistem Penggerak Robot

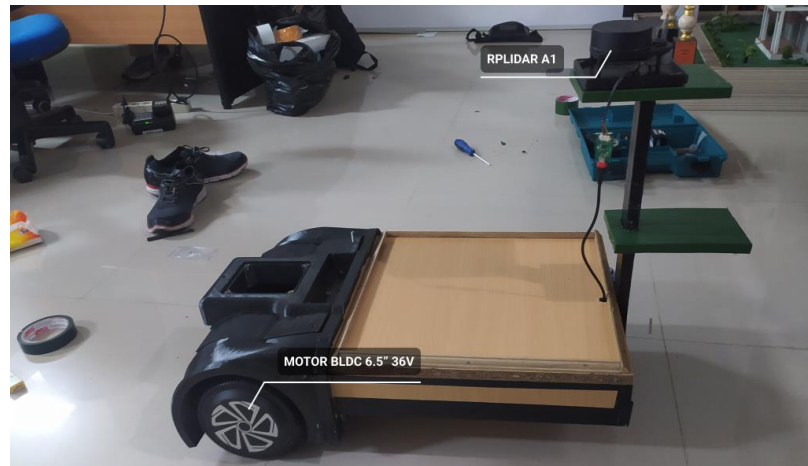
Gambar 5.32 merupakan hasil pengimplementasian dari sistem penggerak robot pada *prototype autonomous mobile robot*. Sistem penggerak ini terdiri dari beberapa komponen elektronik yang diletakkan di dalam *chassis mainboard* robot. Dengan menggunakan 2 buah roda aktif sebagai aktuator penggerak yang dihubungkan dengan *motherboard hoverboard*. Sebenarnya sistem ini sudah dapat dioperasikan dengan menggunakan *remote* kontrol karena sudah terhubung dengan komputer. Komponen yang digunakan dalam sistem ini merupakan komponen penyusun utama yang terdiri dari *microcomputer* Jetson Nano, *motherboard hoverboard*, baterai li-ion 10S2P, dan juga komponen pendukung lainnya seperti FTDI mini USB, kabel mikro USB, *power button*, *charger port*, *power bank*, dan banyak kabel lainnya. Semua komponen ini tidak ditaruh menjadi satu di dalam *chassis* hitam ini saja karena sistem penggerak robot memerlukan tutup pada *chassis* untuk menghindari percikan air. Jika menggunakan tutup maka *chassis* ini tidak bisa ditutup karena tidak muat terhalang *microcomputer* dan *power bank*, selain itu dikhawatirkan *microcomputer* mengalami *overheat*.

Jetson Nano digunakan sebagai pemrosesan utama dalam sistem ini, semua jenis pemrosesan seperti pengolahan data sensor dan pengendalian sistem *hoverboard* juga dikendalikan oleh perangkat *microcomputer* ini. Jetson Nano mendapatkan sumber daya dari *power bank* dengan masukan 5 volt. Jetson Nano juga terhubung dengan *motherboard* menggunakan kabel sensor sebelah kanan *motherboard*. Kabel sensor ini dapat dimanipulasi sehingga mampu menggerakkan motor BLDC. Motor ini terhubung langsung dengan *motherboard* pada *pin phase* dan *socket* kabel *hall*. Motor ditenagai langsung oleh *motherboard* yang diambil dari sumber daya baterai *lithium ion* dengan tipe baterai 10S2P yang memiliki voltase hingga 42 volt. Penjelasan lebih lanjut mengenai penyambungan kabel dan *port* dijelaskan pada Tabel 5.17.

Tabel 5.17 Pemasangan Kabel Perangkat Keras Sistem Penggerak

Perangkat Keras	Pin	Tujuan	Pin
Motor BLDC	Phase A	<i>Motherboard</i>	Phase A MCU
	Phase B		Phase B MCU
	Phase C		Phase C MCU
	HALL A		HALL A
	HALL B		HALL B
	HALL C		HALL C
	GND		GND
	VCC		VCC
<i>Motherboard</i>	PB10/TX/USART3	FTDI mini USB	RX
	PB11/RX/USART3		TX
	GND		GND
FTDI mini USB	RX	<i>Jetson Nano</i>	Port USB
	TX		
	GND		
Baterai Li-Ion	36 volt VCC	<i>Motherboard</i>	XT60 VCC
	GND		XT60 GND
Power Bank	Port USB	<i>Jetson Nano</i>	Port Mikro USB

Setelah selesai mengimplementasikan sistem penggerak robot, maka implementasi selanjutnya adalah menghubungkan sensor RPLIDAR dan IMU pada *Jetson Nano* sebagai sistem persepsi robot. Berdasarkan perancangan, sensor ini di pasang setelah kedua *chassis* terhubung dan posisi robot sudah dalam kondisi tetap. Sensor RPLIDAR di posisi bagian depan robot, hal ini ditujukan untuk memudahkan pembacaan ketika terdapat barang di bagian tengah robot. Jika sensor RPLIDAR ditaruh pada bagian belakang robot, maka ketika ada barang yang lebih tinggi posisinya dari pada sensor RPLIDAR, maka robot tidak bisa melihat lingkungan bagian depan yang mengakibatkan tidak mempunya melakukan perencanaan jalur. Sebaliknya jika di posisi depan maka setidaknya RPLIDAR masih mampu membaca lingkungan di depan dan posisi halangan membelakangi gerak maju robot, sehingga perencanaan jalur tidak terganggu. Dokumentasi dari implementasi ini dapat dilihat pada Gambar 5.33.



Gambar 5.33 Implementasi Sistem Persepsi Robot Dengan Sensor

Pada Gambar 5.33 dapat dilihat bahwa sensor RPLIDAR berdiri di atas dudukan sensor dan tiang penyangga dengan tinggi 50 cm dari lantai yang terhubung langsung dengan Jetson Nano menggunakan kabel mikro USB dan adapter USB yang tergantung di tiang penyangga. Sedangkan sensor IMU persis terletak di tengah robot yang terhubung langsung dengan Jetson Nano menggunakan kabel *jumper*. Sensor RPLIDAR menggunakan protokol komunikasi *serial* menggunakan UART, sedangkan sensor IMU menggunakan protokol komunikasi *serial* menggunakan I2C. Penjelasan lebih lanjut mengenai penyambungan kabel dan *port* dijelaskan pada Tabel 5.18.

Tabel 5.18 Pemasangan Kabel Sistem Sensor

Perangkat Keras	Pin	Tujuan	Pin
RPLIDAR	GND	UART TTL Adapter	GND
	TX		TX
	RX		RX
	V5		V5
	GND		GND
	MOTOC LT		DTR
	VMOTO		VCC
UART Adapter	Mikro USB	Jetson Nano	Port USB
GY-521 IMU	VCC	Jetson Nano	3.3V
	GND		GND
	SCL		GPIO (SCL)
	SDA		GPIO 3 (SDA)

5.2.2 Implementasi Perangkat Lunak

Implementasi perangkat lunak merupakan bagian dari merealisasikan algoritma pada sistem dari bab perancangan perangkat lunak sebelumnya. Implementasi ini meliputi realisasi dari persiapan awal, program utama dan sub proses dari algoritma yang dijelaskan pada subbab berikut.

5.2.2.1 Implementasi Persiapan Awal Motherboard

Implementasi persiapan awal *motherboard* ditujukan untuk melakukan proses *flashing firmware* yang sudah diubah ke dalam *motherboard* dengan menggunakan adapter berupa ST-LINK V2. Terkait kode program untuk konfigurasi *firmware* dapat dilihat pada Tabel 5.19.

Tabel 5.19 Program Konfigurasi *Firmware Motherboard Hoverboard*

No.	Kode Program	
	hoverboard-firmware-hack-FOC/Inc/config.h	
12	#if !defined(PLATFORMIO)	
13	// #define VARIANT_ADC	
14	#define VARIANT_UART	
15	// #define VARIANT_NUNCHUK	
16	// #define VARIANT_PPM	
17	// #define VARIANT_PWM	
18	// #define VARIANT_IBUS	
19	// #define VARIANT_HOVERCAR	
20	// #define VARIANT_HOVERBOARD	
21	// #define VARIANT_TRANSPOTTER	
22	// #define VARIANT_SKATEBOARD	
23	#endif	
...	...	
176	#define INACTIVITY_TIMEOUT	30
177	#define BEEPS_BACKWARD	0
178	#define ADC_MARGIN	100
179	#define ADC_PROTECT_TIMEOUT	100
180	#define ADC_PROTECT_THRESH	200
181	// #define AUTO_CALIBRATION_ENA	
...	...	
310	// ##### VARIANT_UART SETTINGS #####	
311	#ifdef VARIANT_UART	
312		
313	// #define CONTROL_SERIAL_USART2 0	
314	// #define FEEDBACK_SERIAL_USART2	
315		
316	// #define SIDEBBOARD_SERIAL_USART2 0	
317	// #define CONTROL_SERIAL_USART2 0	
318	// #define FEEDBACK_SERIAL_USART2	
319		
320	// #define SIDEBBOARD_SERIAL_USART3 0	
321	#define CONTROL_SERIAL_USART3 0	
322	#define FEEDBACK_SERIAL_USART3	
323		
324	// #define DUAL_INPUTS	
325	#define PRI_INPUT1	3, -1000, 0, 1000, 0
326	#define PRI_INPUT2	3, -1000, 0, 1000, 0
327	#ifdef DUAL_INPUTS	

328	#define FLASH_WRITE_KEY 0x1102
329	/// #define SIDEBBOARD_SERIAL_USART2 1
330	/// #define SIDEBBOARD_SERIAL_USART3 1
331	#define AUX_INPUT1 3, -1000, 0, 1000, 0
332	#define AUX_INPUT2 3, -1000, 0, 1000, 0
333	#else
334	#define FLASH_WRITE_KEY 0x1002
335	#endif
336	
337	// #define SUPPORT_BUTTONS_LEFT
338	// #define SUPPORT_BUTTONS_RIGHT
339	#endif
340	// ##### END OF VARIANT_UART SETTINGS #####

Tabel di atas merupakan kode program untuk mengonfigurasi *firmware motherboard hoverboard* yang berisikan potongan-potongan baris yang pentingnya saja. Jumlah baris kode pada *file config.h* adalah 772, akan tetapi hanya beberapa baris kode saja yang diubah tetapi mampu mempengaruhi keseluruhan kode. *File config.c* merupakan *file library* menggunakan bahasa pemrograman C. Di bawah ini penjelasan mengenai kode program konfigurasi *firmware motherboard hoverboard*.

1. Baris ke-12, merupakan *if preprocessor* dengan kondisi jika *macro* PLATFORMIO tidak terdefinisi maka kondisi *true*, jika tidak maka *false*. Baris ini bertujuan agar proses *flashing* dapat menggunakan *extension* PlatformIO, sehingga melakukan *upload* bisa dilakukan dengan mudah.
2. Baris ke-13 (non aktif/dikomentari), merupakan *macro define* varian untuk *input* ADC. Jika ingin melakukan kontrol menggunakan sensor potensiometer maka baris ini diaktifkan.
3. Baris ke-14, merupakan *macro define* varian untuk *input* menggunakan *serial control* UART. Jika ingin melakukan kontrol menggunakan komputer melalui RX/TX maka baris ini harus diaktifkan. Penelitian ini menggunakan kontrol dari UART, maka dari itu baris ini diaktifkan dengan batalkan komentar.
4. Baris ke-15 hingga 22 (non aktif/dikomentari), merupakan *macro define* varian untuk *input* dengan mode-mode lainnya, setiap dari setiap baris kode memiliki sinyalnya masing-masing namun karena penelitian ini menggunakan UART maka kita berfokus pada baris ke 14 saja.
5. Baris ke-23, merupakan akhir atau penutup dari *if preprocessor*. Jika *#if preprocessor* sudah dilakukan maka wajib mengakhirinya menggunakan *#endif*.
6. Baris ke-176, merupakan INACTIVITY_TIMEOUT, yaitu ketika motor tidak bergerak sama *define macro* sekali setelah *motherboard* menyala maka *motherboard* akan mati. Nilai 30 berarti baru akan *shutdown* ketika 30 menit tidak bergerak.

7. Baris ke-177, merupakan *define macro* BEEPS_BACKWARD, yaitu ketika roda mundur maka akan berbunyi. Nilai 0 berarti *false*, dan *beep* tidak akan berbunyi.
8. Baris ke-178 - 180, merupakan *define macro* ADC_MARGIN, yaitu min dan *max* menggunakan *input* ADC.
9. Baris ke-179, merupakan *define macro* ADC_PROTECTED_TIMEOUT, yaitu *timeout* yang digunakan untuk proteksi dari kesalahan *input* ADC.
10. Baris ke-180, merupakan *define macro* ADC_PROTECT_THRESH, yaitu *threshold* yang dibutuhkan untuk proteksi *input* ADC. ADC perlu ada proteksi karena nilainya yang cepat dan sensitif untuk berubah.
11. Baris ke-181 (non aktif/dikomentari), merupakan *define* AUTO_CALIBRATION_ENA, yaitu ketika *hoverboard* baru menyala maka akan melakukan auto kalibrasi namun sebelumnya harus menekan tombol *power* dengan waktu yang lama. *Macro* ini tidak dibutuhkan jadi non aktifkan.
12. Baris ke-310 hingga 340, merupakan konfigurasi untuk menggunakan USART3 (kabel sensor *motherboard* kanan) sebagai *input* RX/TX. Komentar semua *define* yang berhubungan dengan USART2 atau SIDEBOARD, cukup aktifkan baris CONTROL_SERIAL_USART3 0 dan FEEDBACK_SERIAL_USART3. Masing-masing fungsinya untuk melakukan kontrol dan menerima *feedback*, sehingga mampu menerima sinyal *cmd_vel* dan membaca topik *odometry*.

5.2.2.2 Implementasi Persiapan Awal Jetson Nano (Proses A)

Implementasi persiapan awal Jetson Nano terdiri dari beberapa pengaturan komunikasi dan konfigurasi. Persiapan awal Jetson Nano bertujuan untuk memudahkan pembaca dalam mengetahui apa saja yang dibutuhkan sebelum membuat kode untuk program utama. Penelitian ini menggunakan banyak sekali *library* dan paket/modul dari berbagai kode sumber untuk menunjang sistem otomatisasi manuver robot.

Tabel 5.20 Program Konfigurasi Driver Hoverboard

No.	Kode Program
	hoverboard_driver/include/config.h
1	#pragma once
2	
3	#define DEFAULT_PORT "/dev/ttyHOVER"
4	
5	#define ENCODER_MIN 0
6	#define ENCODER_MAX 9000
7	#define ENCODER_LOW_WRAP_FACTOR 0.3
8	#define ENCODER_HIGH_WRAP_FACTOR 0.7
9	
10	#define TICKS_PER_ROTATION 90

Tabel di atas merupakan kode program untuk mengonfigurasi *driver hoverboard*. Berikut ini penjelasan mengenai Kode Program Konfigurasi *Driver Hoverboard*.

1. Baris ke-1, merupakan *preprocessor #pragma* yang ditujukan untuk menyebabkan *file* sumber saat ini (*config.h*) disertakan hanya sekali dalam satu kompilasi. Keuntungan lainnya lebih sedikit kode, menghindari bentrokan nama, dan meningkatkan kecepatan kompilasi.
2. Baris ke-3, merupakan *preprocessor define* yang *initialization macro* *DEFAULT_PORT* bernilai `"/dev/ttyHOVER"` sehingga *port* *ttyHOVER* menjadi *default port driver hoverboard*.
3. Baris ke-5, merupakan *macro* *ENCODER_MIN* bernilai 0 yang artinya pembacaan *encoder* atau *odometry* dimulai dari 0
4. Baris ke-6, merupakan *macro* *ENCODER_MAX* bernilai 9000 yang artinya pembacaan *encoder* atau *odometry* berakhir atau maksimal mencapai 9000.
5. Baris ke-7, merupakan *macro* *ENCODER_LOW_WRAP_FACTOR* bernilai 0.3 yang artinya pembungkus *encoder* terkecil bernilai 0.3
6. Baris ke-8, merupakan *macro* *ENCODER_HIGH_WRAP_FACTOR* bernilai 0.7 yang artinya pembungkus *encoder* terbesar bernilai 0.7
7. Baris ke-10, merupakan *macro* *TICKS_PER_ROTATION* bernilai 90 yang artinya *encoder* mempunyai 90 *tick* per 1 revolusi/rotasi roda.

Setelah mengubah *file config*, langkah selanjutnya adalah mengubah *file hoverboard.launch* pada *directory* `"hoverboard_driver/hoverboard.launch"` seperti yang dapat dilihat pada Tabel 5.21. Kesimpulan dari konfigurasi ini adalah mengubah nilai `"/dev/ttyTH1"` menjadi `"/dev/ttyHOVER"` sesuai dengan nama alias yang sebelumnya sudah dibuat.

Tabel 5.21 Program Konfigurasi *Launcher Hoverboard*

No.	Kode Program
	<code>hoverboard_driver/hoverboard.launch</code>
1	<pre> <launch> <param name="port" type="str" value="/dev/ttyHOVER"/> <rosparam file="\$(find hoverboard_driver)/config/hardware.yaml" command="load"/> <rosparam file="\$(find hoverboard_driver)/config/controllers.yaml" command="load"/> <node name="hoverboard_driver" pkg="hoverboard_driver" type="hoverboard_driver" output="screen"/> <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false" output="screen" args="hoverboard_joint_publisher hoverboard_velocity_controller" /> </pre>
2	
3	
4	
5	
6	
7	
8	

9	</launch>
---	-----------

Tabel di atas merupakan kode program untuk mengonfigurasi *launcher hoverboard*. Berikut ini penjelasan mengenai Kode Program Konfigurasi *Launcher Hoverboard*.

1. Baris ke-1, merupakan open *tag* tipe *launch file*.
2. Baris ke-2, merupakan inisialisasi nilai parameter *port* = */dev/ttyHOVER*.
3. Baris ke-4 dan ke-5, merupakan inisialisasi parameter *ros* dengan mengambil nilai parameter yang ada pada *file* *hardware.yaml* dan *controllers.yaml*.
4. Baris ke-6, merupakan inisialisasi *node* *hoverboard_driver* yang diambil dari paket *hoverboard_driver* pada sub folder *include*.
5. Baris ke-7, merupakan inisialisasi *node* *controller_spawner* pada paket *controller_manager* dengan nilai argumen diambil dari *controllers.yaml* berupa *hoverboard_join_publisher* dan *hoverboard_velocity_controller*. Di dalam *hoverboard_velocity_controller* memanggil *node* *diff_drive_controller* untuk mengendalikan roda.
6. Baris ke-9, merupakan penutup tak tipe *launch file*.

Proses selanjutnya adalah *install driver* RPLIDAR ROS ke dalam *workspace*.

```
$ git clone https://github.com/robopeak/rplidar_ros
```

Ubah nama folder tersebut menjadi "*rplidar_ros*" dan ubah juga *file* RPLIDAR.launch pada *path* "*rplidar_ros/launch/rplidar.launch*" seperti yang dapat dilihat pada Tabel 5.22. Konfigurasi ini untuk mengubah nilai *"/dev/ttyUSB0"* menjadi *"/dev/ttyRPLIDAR "* seperti nama alias yang sudah dibuat sebelumnya

Tabel 5.22 Program Konfigurasi *Launcher* RPLIDAR

No.	Kode Program
	<i>rplidar_ros/launch/rplidar.launch</i>
1	<launch>
2	<node name="rplidarNode" pkg="rplidar_ros"
3	type="RPLiDARNode" output="screen">
4	<param name="serial_port" type="string"
5	value="/dev/ttyRPLIDAR"/>
6	<param name="serial_baudrate" type="int"
7	value="115200"/><!--A1/A2 -->
8	<param name="frame_id" type="string"
9	value="laser"/>
10	<param name="inverted" type="bool"
11	value="false"/>
12	<param name="angle_compensate" type="bool"
13	value="true"/>
14	</node>
15	</launch>

Tabel di atas merupakan kode program untuk mengonfigurasi *launcher driver* RPLIDAR. Berikut ini penjelasan mengenai Kode Program Konfigurasi *Launcher* RPLIDAR.

1. Baris ke-1, merupakan open *tag* tipe *launch file*.
2. Baris ke-2, merupakan inisialisasi *node* *rplidarNode* pada paket *rplidar_ros*
3. Baris ke-3, merupakan inisialisasi nilai parameter *serial port* = */dev/ttyRPLIDAR*.
4. Baris ke-4, merupakan inisialisasi parameter *serial_baudrate* dengan nilai *baud rate* sebesar 115200.
5. Baris ke-5, merupakan inisialisasi parameter nama *frame* pada *RPLIDAR*, yaitu "laser"
6. Baris ke-6, merupakan parameter *inverted* yang artinya jika RPLIDAR dalam kondisi terbalik/digunakan terbalik, maka nilai *inverted* harus *true*. Namun karena posisi RPLIDAR normal maka dari itu value *false*
7. Baris ke-7, merupakan parameter *angle_compensate* untuk *compensation* sudut karena prinsip *message* pemindaian *ros* dengan sudut yang ditingkatkan akan tetap.
8. Baris-8 dan ke-9, merupakan *tag* penutup dari *node* dan *launch*

Proses selanjutnya adalah *install driver* sensor IMU dan melakukan kalibrasi.

```
$ git clone https://github.com/Brazilian-Institute-of-Robotics/mpu6050_driver
$ git clone https://github.com/Brazilian-Institute-of-Robotics/i2c_device_ros
```

Kemudian ubah *file* *mpu6050_node.cpp* pada *path* "*mpu6050_driver/src/mpu6050_node.cpp*" dan ubah baris ke 39 dari "*imu/data_raw*" menjadi "*imu/data*" seperti yang dapat dilihat pada Tabel 5.23.

Tabel 5.23 Program Konfigurasi Nama Topik *Output* IMU

No.	Kode Program
	<i>mpu6050_driver/src/mpu6050_node.cpp</i>
38	<code>void MPU6050Node::init() {</code>
39	<code> mpu_data_pub_ =</code>
	<code> nh_.advertise<sensor_msgs::Imu>("imu/data", 1);</code>
40	
41	<code> this->loadParameters();</code>
42	
43	<code> mpu6050_.setAddress(static_cast<uint8_t>(mpu6050_addr_));</code>
44	<code> mpu6050_.initialize(i2c_bus_uri_);</code>
45	<code> mpu6050_.setDLPFMode(static_cast<uint8_t>(4));</code>
46	<code> mpu6050_.setIntDataReadyEnabled(true);</code>
47	<code> this->setMPUOffsets();</code>
48	
49	<code> ROS_INFO("MPU6050 Node has started");</code>
50	<code>}</code>

Tabel di atas merupakan kode program untuk mengonfigurasi Nama Topik *Output* IMU. Berikut ini penjelasan mengenai Kode Program Konfigurasi Nama Topik *Output* IMU.

1. Baris ke-38, merupakan inialisasi void `init()` dalam kelas `MPU6050Node`
2. Baris ke-39, merupakan *call function* `advertise` dengan parameter "imu/data dan nilai 1.
3. Baris ke-41, merupakan *call function* `loadParameters`
4. Baris ke-43 hingga ke-50, merupakan inialisasi objek `mpu6050_` seperti menentukan alamat, inialisasi alamat `i2c`, menentukan `DLPMode`, inialisasi fungsi `setIntDataReadyEnable` bernilai *true* dan mencetak ke *terminal* "MPU6050 Node has started"

Selanjutnya kita akan berpindah pada pemasangan modul atau paket yang digunakan pada program pemetaan, yaitu Hector SLAM.

```
$ git clone https://github.com/tu-darmstadt-ros-pkg/hector_slam
```

Setelah itu, lakukan perubahan pada beberapa *file* yang ada di folder `hector_slam` terutama *file* `mapping_default.launch` pada *directory* "hector_slam/hector_mapping/launch/mapping_default.launch, dengan kode yang ada pada Tabel 5.24.

Tabel 5.24 Program Launcher Mapping Default

No.	Kode Program
	hector_slam/hector_mapping/launch/mapping_default.launch
1	<pre> <?xml version="1.0"?> <launch> <arg name="tf_map_scanmatch_transform_frame_name" default="scanmatcher_frame"/> <arg name="base_frame" default="base_link"/> <arg name="odom_frame" default="odom_combined"/> <arg name="pub_map_odom_transform" default="false"/> <arg name="scan_subscriber_queue_size" default="5"/> <arg name="scan_topic" default="scan"/> <arg name="map_size" default="2048"/> <!-- New --> <arg name="pub_odometry" default="true"/> <node pkg="hector_mapping" type="hector_mapping" name="hector_odom" output="screen"> <!-- Frame names --> <param name="map_frame" value="map" /> <param name="base_frame" value="\$(arg base_frame)" /> <param name="odom_frame" value="\$(arg odom_frame)" /> <!-- Tf use --> <param name="use_tf_scan_transformation" value="true"/> </pre>
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	

23	<param name="use_tf_pose_start_estimate"
	value="false"/>
24	<param name="pub_map_odom_transform" value="\$ (arg
	pub_map_odom_transform)"/>
25	<!-- New -->
26	<param name="pub_odometry" value="\$ (arg
	pub_odometry)"/>
27	
28	<!-- Map size / start point -->
29	<param name="map_resolution" value="0.050"/>
30	<param name="map_size" value="\$ (arg map_size)"/>
31	<param name="map_start_x" value="0.5"/>
32	<param name="map_start_y" value="0.5" />
33	<param name="map_multi_res_levels" value="2" />
34	
35	<!-- Map update parameters -->
36	<param name="update_factor_free" value="0.4"/>
37	<param name="update_factor_occupied" value="0.9" />
38	<param name="map_update_distance_thresh" value="0.4"/>
39	<param name="map_update_angle_thresh" value="0.06" />
40	<param name="laser_z_min_value" value = "-1.0" />
41	<param name="laser_z_max_value" value = "1.0" />
42	
43	<!-- Advertising config -->
44	<param name="advertise_map_service" value="true"/>
45	<param name="scan_subscriber_queue_size" value="\$ (arg
	scan_subscriber_queue_size)"/>
46	<param name="scan_topic" value="\$ (arg scan_topic)"/>
	<param name="tf_map_scanmatch_transform_frame_name"
47	value="\$ (arg tf_map_scanmatch_transform_frame_name)" />
	</node>
48	</launch>

Tabel di atas merupakan kode program untuk mengonfigurasi *file mapping* pada *hector mapping*. Berikut ini penjelasan mengenai Kode Program *Launcher Mapping Default*.

1. Baris ke-1 hingga ke-3, merupakan open *tag* untuk format *launch* dengan *define xml version* dan open *tag* <launch>
2. Baris ke-4 hingga ke-10, merupakan inisialisasi argumen yang akan digunakan nantinya pada parameter. Argumen pada *launch file* persis seperti sebuah variabel pada bahasa pemrograman, yaitu penyimpanan sementara yang nilainya akan digunakan pada baris berikutnya. Keuntungan menggunakan *tag* arg atau argumen pada *launch file* ada kode yang rapi dan memudahkan jika menginginkan perubahan *value* karena tidak perlu mencari setiap parameter pada *node* atau *rosparam*.
3. Baris ke-14 hingga ke-47, merupakan inisialisasi *node* *hector_mapping* dengan nilai parameter menggunakan variabel argumen yang sudah ditentukan sebelumnya.
4. Baris ke-48, merupakan penutup *tag launch*

Di dalam *geotiff* juga memanggil *node* trajectory_server, namun saat ini kita akan berfokus pada mengubah *file* tutorial.launch menjadi *file* tutorial.launch dengan kode program yang terdapat pada Tabel 5.25.

Tabel 5.25 Program Percobaan Launcher pada Hector SLAM Launch

No.	Kode Program
	hector_slam/hector_slam_launch/launch/map_tutorial.launch
1	<?xml version="1.0"?>
2	
3	<launch>
4	
5	<arg name="geotiff_map_file_path" default="\$(find hector_geotiff)/maps"/>
6	
7	<param name="/use_sim_time" value="true"/>
8	
9	<node pkg="rviz" type="rviz" name="rviz" args="-d \$(find hector_slam_launch)/rviz_cfg/mapping_demo.rviz"/>
10	
11	<include file="\$(find hector_mapping)/launch/mapping_default.launch"/>
12	
13	<include file="\$(find hector_geotiff_launch)/launch/geotiff_mapper.launch">
14	<arg name="trajectory_source_frame_name" value="scanmatcher_frame"/>
15	<arg name="map_file_path" value="\$(arg geotiff_map_file_path)"/>
16	</include>
17	
18	</launch>

Tabel di atas merupakan kode program untuk mengonfigurasi *file* tutorial.launch pada hector_slam_launch. Berikut ini penjelasan mengenai Kode Program Launcher Map Hector pada AMR Launch.

1. Baris ke-1 hingga ke-3, merupakan open *tag* untuk format *launch* dengan *define xml version* dan open *tag* <launch>.
2. Baris ke-5, merupakan argumen dengan nama geotiff_map_file_path untuk parameter letak map sementara yang dihasilkan oleh *hector mapping*.
3. Baris ke-7, merupakan parameter jika menggunakan simulasi atau dunia 3D. Berikan *value false* jika robot berada di dunia nyata dan *true* jika bermain dengan simulasi.
4. Baris ke-9, merupakan *node* yang digunakan untuk memanggil aplikasi RViz. *Node* ini diberikan tambahan argumen berupa *file mapping_demo.rviz* yang ada di folder rviz_cfg. *File* ini berisi parameter untuk mengaktifkan atau mematikan fitur/topik/frame pada aplikasi RViz.
5. Baris ke-13 hingga ke-16, merupakan pemanggilan *file* hector_mapper.launch, namun bedanya pemanggilan *file* ini ditambahkan

dengan 2 argumen tambahan, yaitu `trajectory_source_frame_name` dengan nilai `frame = frame` hasil `hector_mapping` dan parameter yang kedua `map_file_path` yang berisikan `path folder` di mana map sementara disimpan.

6. Baris ke-17-18, merupakan penutup `tag launch`.

Selesai melakukan konfigurasi pada semua *driver* sensor berupa *library* atau paket algoritme yang dibutuhkan pada penelitian ini, maka langkah selanjutnya adalah melakukan konfigurasi pada folder utama penyusun sistem robot yang diberi nama folder "*autonomous_mobile_robot_navigation*". Folder ini tidak didapatkan dari sumber GitHub atau *library* ROS melainkan folder sumber yang di kode secara manual. Jadi pada kali ini penulis benar-benar membuat kode dari awal hingga akhir. Folder *autonomous_mobile_robot_navigation* atau yang nanti pada penulisan disingkat dengan *AMR Navigation*.

Proyek pada *folder autonomous mobile robot navigation* dibuat dengan menggunakan perintah *create catkin package* pada *terminal* pada folder *src*.

```
$ catkin_create_pkg autonomous_mobile_robot_navigation std_msgs rospy
roscpp
```

Maka dibuatkan sebuah folder dengan nama tersebut dengan *file* dan *folder template* di dalam, seperti *include*, *package.xml*, dan *CMakeLists.txt*. Langkah selanjutnya adalah mengubah isi dari *file package.xml*, tujuannya adalah untuk mengunduh semua paket yang dibutuhkan pada penelitian ini yang dapat dilihat pada Tabel 5.26.

Tabel 5.26 Program Paket ROS pada package.xml

No.	Kode Program
	<i>autonomous_mobile_robot_navigation/package.xml</i>
1	<?xml version="1.0"?>
2	<package format="2">
3	<name>autonomous_mobile_robot_navigation</name>
4	<version>0.0.2</version>
5	<description>The AMR Navigation package</description>
6	<maintainer email="bambanggunawan887@gmail.com">Bambang Gunawan</maintainer>
7	
8	<license>BSD</license>
9	<buildtool_depend>catkin</buildtool_depend>
10	
11	<depend>amcl</depend>
12	<depend>actionlib</depend>
13	<depend>actionlib_msgs</depend>
14	<depend>geometry_msgs</depend>
15	<depend>control_msgs</depend>
16	<depend>message_generation</depend>
17	<depend>controller_manager</depend>
18	<depend>gmapping</depend>
19	<depend>move_base</depend>
20	<depend>roscpp</depend>

21	<depend>rospy</depend>
22	<depend>std_msgs</depend>
23	<depend>std_srvs</depend>
24	<depend>tf</depend>
25	<depend>tf2</depend>
26	<depend>control_toolbox</depend>
27	<depend>map_server</depend>
28	<depend>rosparam_shortcuts</depend>
29	<depend>hardware_interface</depend>
30	<depend>laser_filters</depend>
31	<depend>imu_filter_madgwick</depend>
32	<depend>rviz_imu_plugin</depend>
33	<depend>robot_localization</depend>
34	<depend>teb_local_planner</depend>
35	<depend>dwa_local_planner</depend>
36	<depend>global_planner</depend>
37	<depend>twist_mux</depend>
38	<depend>nmea_navsat_driver</depend>
39	<depend>moveit_ros_planning_interface</depend>
40	</package>

Tabel di atas merupakan kode program untuk inisialisasi semua paket yang dibutuhkan. Berikut ini penjelasan mengenai Kode Program.

1. Baris ke-1 hingga ke-9, merupakan open *tag xml* dan beberapa deskripsi pada *file* serta deklarasi *build tool* menggunakan *catkin*.
2. Baris ke-11 hingga ke-39, merupakan nama-nama paket yang dibutuhkan pada penelitian ini, dengan menjadikan satu semua paket maka tidak perlu memasang paket satu persatu.

5.2.2.3 Implementasi Persiapan Awal Jetson Nano (Proses B)

Program implementasi selanjutnya adalah proses kalibrasi sensor IMU. Proses kalibrasi merupakan proses pengecekan dan pengaturan akurasi dari alat ukur dengan cara membandingkannya dengan standar/tolak ukur. Kalibrasi sensor dilakukan tersedia di dalam *driver* mpu6050_driver sehingga kita hanya perlu menjalankan *file launch* seperti yang terlihat pada Tabel 5.27.

Tabel 5.27 Program Konfigurasi Parameter MPU6050

No.	Kode Program
	mpu6050_driver/config/mpu_settings.yaml
1	# I2C Bus URI used to communicate with I2C devices (default: "/dev/i2c-1")
2	bus_uri: "/dev/i2c-1"
3	
4	# I2C address of MPU6050 (default: 0x68)
5	mpu_address: 0x68
6	
7	# Frequency in Hertz wich IMU data is published (default: 30)
8	pub_rate: 25
9	
10	# Frame if of IMU message (default: "imu")
11	frame_id: "imu"

12	
13	# Offsets to fix wrong values caused by misalignment
14	# Sequence is (ax, ay, az, gx, gy, gz) (default: [0, 0, 0, 0, 0, 0])
15	axes_offsets: [-1340, -9, 1463, 123, -42, 42]
16	
17	# PID constants used in calibration procedure
18	ki: 0.2 # (default: 0.1)
19	kp: 0.1 # (default: 0.1)
20	
21	# The calibration process is finished when the error is
22	aproximate zero with
23	# the precision set by delta (default: 0.5)
23	delta: 0.5

Tabel di atas merupakan kode program untuk mengonfigurasi pengaturan utama IMU. Berikut ini penjelasan mengenai Kode Program Konfigurasi Parameter MPU6050.

1. Baris ke-2, merupakan inialisasi variabel `bus_uri` = `"/dev/i2c-1"`
2. Baris ke-5, merupakan inialisasi alamat sensor GY 521 pada jalur i2c. Alamat menggunakan nilai *default*, yakni 0x68
3. Baris ke-8, merupakan inialisasi kecepatan *publish* dengan satuan hertz, nilai yang diberikan = 25 hertz
4. Baris ke-11, merupakan inialisasi nama *frame* dari pesan IMU, menggunakan nilai *default* yaitu "imu"
5. Baris ke-15, merupakan nilai *offsets* untuk membenarkan kesalahan nilai karena posisi yang tidak sejajar. Nilai *offsets* ini yang keluar saat melakukan kalibrasi, ubah nilai *offsets* berdasarkan data kalibrasi yang diberikan. Dalam hal ini, nilai kalibrasi sensor pada penelitian ini adalah [-1340, -9, 1463, 123, -42, 42].
6. Baris ke-18 dan ke-19, merupakan nilai PID tetap saat proses kalibrasi
7. Baris ke-23, merupakan nilai delta 0.5 yang berarti proses kalibrasi baru akan berhenti ketika kesalahan mendekati 0 dengan presisi diatur oleh delta.

Setelah melakukan kalibrasi, nilai dari *output* sensor di bandingkan antara sebelum dan sesudah melakukan kalibrasi, hasilnya dapat dilihat pada Tabel 5.28.

Tabel 5.28 Perbandingan *Output* Sensor IMU Sebelum Dan Sesudah

No.	Sebelum Kalibrasi			Sesudah Kalibrasi		
	ax	ay	az	ax	ay	az
1	-3.881	0.358	8.412	0.152	0.152	9.821
2	-3.392	0.324	8.444	0.037	0.151	9.820

3	-3.378	0.341	8.432	0.028	0.029	9.831
4	-3.891	0.363	8.401	0.025	0.014	9.804
5	-3.401	0.335	8.395	0.038	0.152	9.959
6	-3.389	0.344	8.439	0.038	0.017	9.808
7	-3.378	0.352	8.401	0.028	0.018	9.794
8	-3.368	0.336	8.421	0.030	0.012	9.808
9	-3.372	0.355	8.399	0.041	0.022	9.833
10	-3.491	0.334	8.412	0.038	0.018	9.822

Dapat dilihat bahwa proses kalibrasi sangat berpengaruh pada *output* hasil pembacaan sensor IMU, perbedaan sekitar 5-10% dapat berdampak pada kepresisi posisi yang terbaca. Kondisi ini menandakan bahwa sensor IMU dalam keadaan stabil sehingga menghasilkan *output* akselerasi yang akurat.

5.2.2.4 Implementasi Program Pemetaan

Implementasi program pemetaan merupakan salah satu dari proses program utama. Implementasi dari program ini memanfaatkan paket Hector SLAM untuk pemetaan dan paket EKF untuk lokalisasinya. Program pemetaan terdiri dari *file launcher* yang memanggil *file launcher* lainnya. Implementasi proses *launcher mapping* memanggil beberapa *file* lain dan *node* dari *driver*. *File* yang dimaksud adalah *file launcher* dari *driver* yang sebelumnya sudah di konfigurasi. Memanggil *file launcher* lain di dalam *file* adalah kelebihan yang ada pada ekstensi *launcher*, dengan begitu kita dapat menghemat baris dan membuat kode lebih terstruktur dengan rapi. Semua kode yang berhubungan dengan pemetaan dapat dilihat pada Tabel 5.29.

Tabel 5.29 Program Pemetaan Dengan Hector SLAM

No.	Kode Program
	<i>autonomous_mobile_robot_navigation/launch/map_hector.launch</i>
1	<pre> <?xml version="1.0"?> <launch> <!-- Declare launch rviz first --> <arg name="use_rviz" default="true"/> <param name="/use_sim_time" value="false"/> <!-- Static transformation for TF --> <node pkg="tf" type="static_transform_publisher" name="base_laser_broadcaster" args="0 0.06 0.02 0 0 0 base_link laser 100" /> <node pkg="tf" type="static_transform_publisher" name="imu_broadcaster" args="0 0.06 0.02 0 0 0 base_link imu 100" /> </pre>
2	
3	
4	
5	
6	
7	
8	
9	

```

10 <node pkg="tf" type="static_transform_publisher"
    name="dummy_broadcaster" args="0 0 0 0 0 0 base_link dummy
    100" />
11 <node pkg="tf" type="static_transform_publisher"
    name="caster_broadcaster" args="0 0 0 0 0 0 base_link
    caster_wheel 100" />
12 <node pkg="tf" type="static_transform_publisher"
    name="base_link_broadcaster" args="0 0 0.09 0 0 0
    base_footprint base_link 100" />
13
14 <!-- odom to base_footprint transform will be provided by
    the robot_pose_ekf node -->
15 <node pkg="tf" type="static_transform_publisher"
    name="hectorscan_to_odom" args="0 0 0 0 0 0
    scanmatcher_frame odom_combined 100" />
16 <node pkg="tf" type="static_transform_publisher"
    name="map_to_hector" args="0 0 0 0 0 0 map
    scanmatcher_frame 100" />
17
18 <!-- Hoverboard usb port -->
19 <param name="port" type="str" value="/dev/ttyHOVER"/>
20
21 <!-- Hoverboard Driver Setup Parameter -->
22 <include file="$(find
    hoverboard_driver)/launch/hoverboard.launch" />
23 <!-- Run RPLIDAR and IMU driver launch -->
24 <include file="$(find rplidar_ros)/launch/rplidar.launch"
    />
25 <include file="$(find
    mpu6050_driver)/launch/mpu6050_driver.launch" />
26
27 <!-- Robot model -->
28 <arg name="model" default="$(find
    autonomous_mobile_robot_navigation)/urdf/mobile_robot_v2.xa
    cro"/>
29 <param name="robot_description" command="$(find
    xacro)/xacro $(arg model)" />
30 <rosparam file="$(find
    autonomous_mobile_robot_navigation)/config/joint_limits.yam
    l" command="load"/>
31 <node name="robot_state_publisher"
    pkg="robot_state_publisher" type="robot_state_publisher"/>
32
33 <!-- Rviz config param declare -->
34 <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
    autonomous_mobile_robot_navigation)/rviz/mapping_config.rvi
    z"/>
35
36 <!-- Extended Kalman Filter from robot_pose_ekf Node-->
37 <!-- Subscribe: /odom, /imu_data, /vo -->
38 <!-- Publish: /robot_pose_ekf/odom_combined -->
39 <remap from="odom"
    to="/hoverboard_velocity_controller/odom" />
40 <remap from="imu" to="/imu/data" />
41 <node pkg="robot_pose_ekf" type="robot_pose_ekf"
    name="robot_pose_ekf">
42 <param name="output_frame" value="odom_combined"/>
43 <param name="base_footprint_frame"
    value="base_footprint"/>

```

44	<code><param name="freq" value="100.0"/></code>
45	<code><param name="sensor_timeout" value="1.0"/></code>
46	<code><param name="odom_used" value="true"/></code>
47	<code><param name="imu_used" value="true"/></code>
48	<code><param name="vo_used" value="false"/></code>
49	<code><param name="gps_used" value="false"/></code>
50	<code><param name="debug" value="false"/></code>
51	<code><param name="self_diagnose" value="false"/></code>
52	<code></node></code>
53	
54	<code><!-- Hector SLAM --></code>
55	<code><arg name="geotiff_map_file_path" default="\$(find</code> <code>hector_geotiff)/maps"/></code>
56	<code><include file="\$(find</code> <code>hector_mapping)/launch/mapping_default.launch"/></code>
57	
58	<code><include file="\$(find</code> <code>hector_geotiff_launch)/launch/geotiff_mapper.launch"></code>
59	<code><arg name="trajectory_source_frame_name"</code> <code>value="scanmatcher_frame"/></code>
60	<code><arg name="map_file_path" value="\$(arg</code> <code>geotiff_map_file_path)"/></code>
61	<code></include></code>
62	<code><!-- Rqt Robot Steering --></code>
63	<code><node name="rqt_robot_steering" pkg="rqt_robot_steering"</code> <code>type="rqt_robot_steering"></code>
64	<code><remap from="/cmd_vel"</code> <code>to="/hoverboard_velocity_controller/cmd_vel"/></code>
65	<code></node></code>
66	
67	<code></launch></code>

Tabel di atas merupakan kode program untuk melakukan pemetaan. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1 hingga ke-2, merupakan deklarasi penggunaan versi 1.0 *xml* dan deklarasi *tag* pembuka *launch file*
2. Baris ke-4, merupakan argumen untuk deklarasi penggunaan *RViz = true*
3. Baris ke-5, merupakan parameter untuk menggunakan waktu pada simulasi, berikan nilai *false* jika tidak menggunakan simulasi
4. Baris ke-8 hingga ke 16, merupakan inisialisasi *node static transformation* untuk mendeklarasikan secara manual TF dari setiap *node* yang ada.
 - a. Baris ke-8, posisi TF sensor laser RPLIDAR terhubung (child) terhadap *base_link* (parent), agar posisi laser mengikuti pergerakan badan robot,
 - b. Baris ke-9, posisi TF sensor IMU terhubung (child) terhadap *base_link* (parent), agar posisi IMU mengikuti pergerakan badan robot,

- c. Baris ke-10, posisi TF sensor dummy terhubung (child) terhadap base_link (parent), agar posisi dummy mengikuti pergerakan badan robot,
 - d. Baris ke-11, posisi TF sensor caster wheel terhubung (child) terhadap base_link (parent), agar posisi caster wheel mengikuti pergerakan badan robot,
 - e. Baris ke-12, posisi TF base_link terhubung (child) terhadap base_footprint (parent) yang merupakan bagian paling dasar dekat tanah pada robot, agar base_link mengetahui posisi permukaan,
 - f. Baris ke-15, posisi TF odom_combined terhubung (child) terhadap scanmatcher_frame (parent).
 - g. Baris ke-16, posisi TF scanmatcher_frame terhubung (child) terhadap map (parent), sehingga robot akan terkoneksi dengan map
5. Baris ke-19, merupakan redefinisi parameter penamaan *port* USB *hoverboard*.
 6. Baris ke-22 hingga ke-25, merupakan pemanggilan *file launcher* dari *driver* *hoverboard_driver*, *rplidar_ros*, *mpu6050_driver* yang dapat dilihat pada tabel 5.24, 5.25 dan 5.30.
 7. Baris ke-28 hingga ke-30, merupakan deklarasi parameter dan argumen untuk menggunakan model robot yang ada pada *file urdf*. Kode keseluruhan paket *urdf* ini dapat dilihat pada lampiran.
 8. Baris ke-31, merupakan pendeklarasian dan inisialisasi paket *robot state publisher* yang merupakan paket untuk memungkinkan robot mengirimkan TF untuk setiap komponen di dalamnya.
 9. Baris ke-34, merupakan deklarasi dan inisialisasi paket RViz dengan parameter RViz berupa *nav_config.rviz* yang ada pada folder *RViz*. Dengan mendeklarasikan parameter kita dapat menentukan topik-topik apa yang mau di munculkan dan topik apa yang tidak.
 10. Baris ke-39 hingga ke-52, merupakan deklarasi dan inisialisasi paket *Extended Kalman Filter* untuk lokalisasi. Dengan melakukan *remap* pada TF odom yang diambil dari *"/hoverboard_velocity_controller/odom"* dan remap pada TF imu yang diambil dari *"/imu/data"* untuk menginisialisasi ulang topik dengan TF nya. Kemudian mendeklarasikan parameter-parameter yang digunakan pada penelitian ini, seperti *output_frame = odom*, *odom_used = true* dan sebagainya seperti yang sudah ada pada perancangan sebelumnya.
 11. Baris ke-55 hingga ke 61, merupakan pemanggilan *node* Hector SLAM untuk melakukan pemetaan sekaligus lokalisasi berdasarkan map sementara yang didapatkan secara real time

12. Baris ke-63 hingga ke-65, merupakan pemanggilan *node* rqt robot steering yang mensubscribe topik *hoverboard* velocity controller/cmd_vel

13. Baris ke-67, merupakan penutup dari tag launch.

a. Implementasi Parameter *Hoverboard Driver*

Implementasi konfigurasi *hoverboard driver* berisikan *file* konfigurasi untuk tiap-tiap parameter yang dipakai pada perencanaan sebelumnya, jika tidak terdefinisi maka nilai yang dipakai adalah nilai *default*. Semua kode yang berhubungan dengan parameter *hoverboard driver* dapat dilihat pada Tabel 5.30, Tabel 5.31 dan Tabel 5.32.

Tabel 5.30 Program Parameter *Hoverboard Driver*

No.	Kode Program
	hoverboard_driver/config/controllers_hoverboard.yaml
1	hoverboard_joint_publisher:
2	type: "joint_state_controller/JointStateController"
3	publish_rate: 50
4	left_wheel : "left_wheel"
5	right_wheel : "right_wheel"
6	
7	hoverboard_velocity_controller:
8	type : "diff_drive_controller/DiffDriveController"
9	left_wheel : "left_wheel"
10	right_wheel : "right_wheel"
11	pose_covariance_diagonal : [0.001, 0.001, 1000000.0,
12	1000000.0, 1000000.0, 1000.0]
13	twist_covariance_diagonal: [0.001, 0.001, 1000000.0,
14	1000000.0, 1000000.0, 1000.0]
15	publish_rate: 50.0 # default: 50
16	# Wheel separation and radius multipliers
17	wheel_separation_multiplier: 1.0 # default: 1.0
18	wheel_radius_multiplier : 1.0 # default: 1.0
19	# Velocity commands timeout [s], default 0.5
20	cmd_vel_timeout: 0.5
21	# Base frame_id
22	base_frame_id: base_footprint #default: base_link
23	
24	# Velocity and acceleration limits
25	# Whenever a min_* is unspecified, default to -max_*
26	linear:
27	x:
28	has_velocity_limits : true # default false
29	max_velocity : 1.0 # m/s
30	min_velocity : -0.5 # m/s
31	has_acceleration_limits: true # default false
32	max_acceleration : 0.8 # m/s^2
33	has_jerk_limits : false # default false
34	max_jerk : 0.0 # m/s^3
35	angular:
36	z:
37	has_velocity_limits : true # default false
38	max_velocity : 3.14 # rad/s

39	min_velocity	: -3.14	# rad/s
40	has_acceleration_limits	: true	# default false
41	max_acceleration	: 3.14	# rad/s^2
42	has_jerk_limits	: true	# default false
43	max_jerk	: 3.14	# rad/s^3
44			
45	enable_odom_tf	: false	# default true
46	wheel_separation	: 0.34	
47	wheel_radius	: 0.0825	
48	odom_frame_id	: "/odom"	# default: "/odom"
49	publish_cmd	: false	# default false
50	allow_multiple_cmd_vel_publishers	: true	
51	velocity_rolling_window_size	: 10	# default 10

Tabel di atas merupakan kode program untuk melakukan pemetaan. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1 hingga ke-5, merupakan parameter objek dengan nama `hoverboard_joint_publisher`. "`joint_state_controller/JointStateController`" digunakan sebagai tipe kontrolnya agar sistem mengenali bahwa *hoverboard* menggunakan join pada setiap TF nya, sehingga TF dari setiap komponen robot dapat bergerak karena saling terhubung. Publish rate dari paket ini adalah 50 hz dan deklarasi `left_wheel` dan `right_wheel` dengan nilai masing-masingnya karena kendali robot adalah differential drive, sehingga setiap roda independen antara satu dengan yang lain.
2. Baris ke-7 hingga ke-22, merupakan parameter deklarasi untuk paket `move_base` sehingga pose robot dan ukuran roda dapat diketahui.
3. Baris ke-26 hingga ke-43, merupakan parameter untuk mendeklarasikan *velocity* dan *acceleration* pada gelar linier dan angular.
4. Baris ke-45 hingga ke-51, merupakan parameter deklarasi untuk transformasi odom.

Tabel 5.31 Program Parameter *Hoverboard Driver*

No.	Kode Program
	<code>hoverboard_driver/config/hardware.yaml</code>
1	<code>robaka:</code>
2	<code> direction: 1</code>
3	<code> hardware_interface:</code>
4	<code> loop_hz: 50 # hz</code>
5	<code> joints:</code>
6	<code> - left_wheel</code>
7	<code> - right_wheel</code>

Tabel di atas merupakan kode program untuk melakukan pemetaan. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1 hingga ke-7, merupakan parameter objek dengan nama `amr_oclone`. Arah atau *direction* yang dideklarasikan adalah 1 dan hardware interfacenya kurang lebih sama yaitu `left_wheel` dan `right_wheel` yang di *joint*.

Tabel 5.32 Program Parameter *Hoverboard Driver*

No.	Kode Program
	hoverboard_driver/config/joint_limits.yaml
1	joint_limits:
2	right_wheel_joint:
3	has_position_limits: false
4	min_position: 0
5	max_position: 0
6	has_velocity_limits: true
7	max_velocity: 2.62
8	min_velocity: -2.62
9	has_acceleration_limits: false
10	max_acceleration: 0.0
11	has_jerk_limits: false
12	max_jerk: 0
13	has_effort_limits: false
14	max_effort: 0
15	min_effort: 0
16	
17	left_wheel_joint:
18	has_position_limits: false
19	min_position: 0
20	max_position: 0
21	has_velocity_limits: true
22	max_velocity: 2.62
23	min_velocity: -2.62
24	has_acceleration_limits: false
25	max_acceleration: 0.0
26	has_jerk_limits: false
27	max_jerk: 0
28	has_effort_limits: false
29	max_effort: 0
30	min_effort: 0

Tabel di atas merupakan kode program untuk melakukan pemetaan. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1 hingga ke-15, merupakan parameter objek dengan nama `right_wheel_joint`. Fungsinya adalah melakukan limitasi pada proses join dari roda sebelah kanan dengan badan robot sehingga posisi robot dan TF roda akan berjalan dengan sempurna.
2. Baris ke-17 hingga ke-30, merupakan parameter objek dengan nama `left_wheel_joint`. Fungsinya adalah melakukan limitasi pada proses join dari roda sebelah kiri dengan badan robot sehingga posisi robot dan TF roda akan berjalan dengan sempurna.

5.2.2.5 Implementasi Program Navigasi

Implementasi program navigasi merupakan salah satu dari proses program utama. Implementasi dari program ini utamanya memanfaatkan paket *Move Base*, *EKF*, *AMCL*, dan *Map Server*. Program pemetaan terdiri dari *file launcher* yang memanggil *file launcher* lainnya. Implementasi proses *launcher* navigasi

memanggil beberapa *file* lain dan *node* dari *driver* dan paket navigasi lainnya. Semua kode yang berhubungan dengan pemetaan dapat dilihat pada Tabel 5.33.

Tabel 5.33 Program Navigasi

No.	Kode Program
	<i>autonomous_mobile_robot_navigation/launch/autonomous_nav.launch</i>
1	<?xml version="1.0"?>
2	<launch>
3	<!-- Declare launch rviz first -->
4	<arg name="use_rviz" default="true"/>
5	<param name="/use_sim_time" value="false"/>
6	
7	<!-- Static transformation for TF -->
8	<node pkg="tf" type="static_transform_publisher"
	name="base_laser_broadcaster" args="0 0.06 0.02 0 0 0
	base_link laser 100" />
9	<node pkg="tf" type="static_transform_publisher"
	name="imu_broadcaster" args="0 0.06 0.02 0 0 0 base_link
	imu 100" />
10	<node pkg="tf" type="static_transform_publisher"
	name="dummy_broadcaster" args="0 0 0 0 0 0 base_link dummy
	100" />
11	<node pkg="tf" type="static_transform_publisher"
	name="caster_broadcaster" args="0 0 0 0 0 0 base_link
	caster_wheel 100" />
12	<node pkg="tf" type="static_transform_publisher"
	name="base_link_broadcaster" args="0 0 0.09 0 0 0
	base_footprint base_link 100" />
13	
14	<!-- map to odom will be provided by the AMCL -->
15	<node pkg="tf" type="static_transform_publisher"
	name="map_to_odom" args="0 0 0 0 0 0 map odom_combined 100"
	/>
16	
17	<!-- Hoverboard usb port -->
18	<param name="port" type="str" value="/dev/ttyHOVER"/>
19	
20	<!-- Run RPLIDAR and IMU driver launch -->
21	<include file="\$(find
	hoverboard_driver)/launch/hoverboard.launch" />
22	<include file="\$(find rplidar_ros)/launch/rplidar.launch"
	/>
23	<include file="\$(find
	mpu6050_driver)/launch/mpu6050_driver.launch" />
24	
25	<!-- Declare map file -->
26	<arg name="map_file" default="\$(find
	autonomous_mobile_robot_navigation)/maps/basemen_g_per4_nic
	eeeeee_perfecttoooooo.yaml"/>
27	<!-- <arg name="map_file" default="\$(find
	autonomous_mobile_robot_navigation)/maps/lab.yaml"/> -->
28	
29	<!-- Robot model urdf -->
30	<arg name="model" default="\$(find
	autonomous_mobile_robot_navigation)/urdf/mobile_robot_v2.xa
	cro"/>

```

31 <param name="robot_description" command="$(find
xacro)/xacro $(arg model)" />
32
33 <!-- Robot join state -->
34 <rosparam file="$(find
autonomous_mobile_robot_navigation)/config/joint_limits.yam
l" command="load"/>
35 <node name="robot_state_publisher"
pkg="robot_state_publisher" type="robot_state_publisher"/>
36
37 <!-- Rviz config param declare -->
38 <!-- Subscribe: -->
39 <!-- Publish: -->
40 <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
autonomous_mobile_robot_navigation)/rviz/nav_config.rviz"/>
41
42 <!-- Map server -->
43 <!-- Subscribe: /map -->
44 <!-- Publish: /map, /map_metadata -->
45 <node name="map_server" pkg="map_server"
type="map_server" args="$(arg map_file)" respawn="true" />
46
47 <!-- Extended Kalman Filter from robot_pose_ekf Node-->
48 <!-- Subscribe: /odom, /imu_data, /vo -->
49 <!-- Publish: /robot_pose_ekf/odom_combined -->
50 <remap from="odom"
to="/hoverboard_velocity_controller/odom" />
51 <remap from="imu" to="/imu/data" />
52 <node pkg="robot_pose_ekf" type="robot_pose_ekf"
name="robot_pose_ekf">
53 <param name="output_frame" value="odom_combined"/>
54 <param name="base_footprint_frame"
value="base_footprint"/>
55 <param name="freq" value="100.0"/>
56 <param name="sensor_timeout" value="1.0"/>
57 <param name="odom_used" value="true"/>
58 <param name="imu_used" value="true"/>
59 <param name="vo_used" value="false"/>
60 <param name="gps_used" value="false"/>
61 <param name="debug" value="false"/>
62 <param name="self_diagnose" value="false"/>
63 </node>
64
65 <!-- Add AMCL example for differential drive robots for
Localization -->
66 <!-- Subscribe: /scan, /tf, /initialpose, /map -->
67 <!-- Publish: /amcl_pose, /particlecloud, /tf -->
68 <node pkg="amcl" type="amcl" name="amcl" output="screen">
69 <remap from="scan" to="scan"/>
70 <param name="odom_frame_id" value="odom_combined"/>
71 <param name="odom_model_type" value="diff-corrected"/>
72 <param name="base_frame_id" value="base_link"/>
73 <param name="update_min_d" value="0.1"/>
74 <param name="update_min_a" value="0.2"/>
75 <param name="min_particles" value="500"/>
76 <param name="global_frame_id" value="map"/>
77 <param name="tf_broadcast" value="true" />
78 <param name="initial_pose_x" value="0.0"/>
79 <param name="initial_pose_y" value="0.0"/>

```

```

80     <param name="initial_pose_a" value="0.0"/>
81 </node>
82
83     <!-- Activated Dijkstra (commented if doesn't used)-->
84     <!-- <node pkg="unit2_pp" name="dijkstra_solution"
type="unit2_solution_server.py" output="screen"/> -->
85
86     <!-- Import move base -->
87     <arg name="scan_topic" default="/scan"/>
88     <arg name="cmd_vel_topic"
default="/hoverboard_velocity_controller/cmd_vel"/>
89     <arg name="odom_topic" default="/odom_combined"/>
90
91     <!-- Global path planner -->
92     <arg name="base_global_planner"
default="navfn/NavfnROS"/>
93     <!-- <arg name="base_global_planner"
default="srv_client_plugin/SrvClientPlugin"/> -->
94     <!-- <arg name="base_global_planner"
default="global_planner/GlobalPlanner"/> -->
95
96     <!-- Local path planner -->
97     <arg name="base_local_planner"
default="dwa_local_planner/DWAPlannerROS"/>
98
99     <node pkg="move_base" type="move_base" respawn="false"
name="move_base" output="screen">
100
101         <param name="base_global_planner" value="$(arg
base_global_planner)"/>
102         <param name="base_local_planner" value="$(arg
base_local_planner)"/>
103
104         <!-- Costmap params -->
105         <rosparam file="$(find
autonomous_mobile_robot_navigation)/param/costmap_common_pa
rams.yaml" command="load" ns="global_costmap" />
106         <rosparam file="$(find
autonomous_mobile_robot_navigation)/param/costmap_common_pa
rams.yaml" command="load" ns="local_costmap" />
107         <rosparam file="$(find
autonomous_mobile_robot_navigation)/param/global_costmap_pa
rams.yaml" command="load" />
108         <rosparam file="$(find
autonomous_mobile_robot_navigation)/param/local_costmap_par
ams.yaml" command="load" />
109
110         <!-- Global & Local Planner -->
111         <rosparam file="$(find
autonomous_mobile_robot_navigation)/param/local_planner_par
ams.yaml" command="load"/>
112         <rosparam file="$(find
autonomous_mobile_robot_navigation)/param/global_planner_pa
rams.yaml" command="load"/>
113
114         <!-- Remap possition -->
115         <remap from="cmd_vel" to="$(arg cmd_vel_topic)"/>
116         <remap from="odom" to="$(arg odom_topic)"/>
117         <remap from="scan" to="$(arg scan_topic)"/>

```

118	</node>
119	</launch>

Tabel di atas merupakan kode program untuk melakukan pemetaan. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1 hingga ke-2, merupakan deklarasi penggunaan versi 1.0 *xml* dan deklarasi *tag* pembuka *launch file*
2. Baris ke-4, merupakan argumen untuk deklarasi penggunaan *RViz = true*
3. Baris ke-5, merupakan parameter untuk menggunakan waktu pada simulasi, berikan nilai *false* jika tidak menggunakan simulasi
4. Baris ke-8 hingga ke 15, merupakan inisialisasi *node static transformation* untuk mendeklarasikan secara manual TF dari setiap *node* yang ada.
 - a. Baris ke-8, posisi TF sensor laser RPLIDAR terhubung (child) terhadap *base_link* (parent), agar posisi laser mengikuti pergerakan badan robot,
 - b. Baris ke-9, posisi TF sensor IMU terhubung (child) terhadap *base_link* (parent), agar posisi IMU mengikuti pergerakan badan robot,
 - c. Baris ke-10, posisi TF sensor dummy terhubung (child) terhadap *base_link* (parent), agar posisi dummy mengikuti pergerakan badan robot,
 - d. Baris ke-11, posisi TF sensor caster wheel terhubung (child) terhadap *base_link* (parent), agar posisi caster wheel mengikuti pergerakan badan robot,
 - e. Baris ke-12, posisi TF *base_link* terhubung (child) terhadap *base_footprint* (parent) yang merupakan bagian paling dasar dekat tanah pada robot, agar *base_link* mengetahui posisi permukaan,
 - f. Baris ke-15, posisi TF *odom_combined* terhubung (child) terhadap *map* (parent).
5. Baris ke-18, merupakan redefinisi parameter penamaan *port* USB *hoverboard*.
6. Baris ke-21 hingga ke-23, merupakan pemanggilan *file launcher* dari *driver hoverboard_driver*, *rplidar_ros*, dan *mpu6050_driver* yang dapat dilihat pada Tabel 5.21, Tabel 5.22 dan Tabel 5.24.
7. Baris ke-26 hingga ke-27, merupakan deklarasi *file* hasil pemetaan
8. Baris ke-30 hingga ke-31, merupakan deklarasi parameter dan argumen untuk menggunakan model robot yang ada pada *file urdf*. Kode keseluruhan paket *urdf* ini dapat dilihat pada lampiran.
9. Baris ke-34 hingga ke-35, merupakan pemanggilan *node joint state* untuk inisialisasi posisi dan hubungan dari setiap kerangka *urdf*.

10. Baris ke-40, merupakan pemanggilan aplikasi Rviz dengan menggunakan konfigurasi yang ada pada file `nav_config` yang dapat dilihat pada lampiran.
11. Baris ke-45, merupakan pemanggilan *node* map server berdasarkan nilai parameter dari argumen sebelumnya. *Node* ini yang bertugas untuk mempublish topik map sehingga dapat digunakan pada navigasi.
12. Baris ke-50 hingga ke-63, merupakan deklarasi dan inisialisasi paket *Extended Kalman Filter* untuk lokalisasi. Dengan melakukan *remap* pada TF odom yang diambil dari `"/hoverboard_velocity_controller/odom"` dan remap pada TF imu yang diambil dari `"/imu/data"` untuk menginisialisasi ulang topik dengan TF nya. Kemudian mendeklarasikan parameter-parameter yang digunakan pada penelitian ini, seperti `output_frame = odom_combined`, `odom_used = true` dan sebagainya seperti yang sudah ada pada perancangan sebelumnya.
13. Baris ke-68 hingga ke-81, merupakan pemanggilan *node* AMCL untuk melakukan lokalisasi terhadap posisi laser LIDAR. AMCL menggunakan data laser dan *odometry* dalam melakukan rekonfigurasi posisi robot.
14. Baris ke-67, merupakan pemanggilan file launcher `move_base.launch`. Paket ini digunakan sebagai kendali kontrol utama dari robot agar bisa mencapai target tertentu, tujuannya agar proses *mapping* selain menggunakan `rqt` tetapi juga dapat dikendalikan langsung dengan memberikan target tujuan.
15. Baris ke-87 hingga 89, merupakan deklarasi argumen untuk remap topik.
16. Baris ke-92, merupakan deklarasi argumen *global planner* dengan menggunakan `navfn/NavfnROS` sebagai algoritme *global planning* yang ada di dalam paket `move base` secara default.
17. Baris ke-97, merupakan deklarasi argumen *local planner* dengan menggunakan `dwa_local_planner/DWAPlannerROS` sebagai algoritmenya.
18. Baris ke-99 hingga ke-118, merupakan pemanggilan *node* `move_base` dengan beberapa konfigurasi parameter di dalamnya.
 - a. Baris ke-101, merupakan inisialisasi *global planner* dengan `Navfn`
 - b. Baris ke-102, merupakan inisialisasi *local planner* dengan `DWA`
 - c. Baris ke-105 hingga ke-106, merupakan inisialisasi konfigurasi *costmap* secara default untuk masing-masing global dan *local* pada file `costmap_common_params.yaml`
 - d. Baris ke-107, merupakan inisialisasi konfigurasi *costmap* secara global pada file `global_costmap_params.yaml`
 - e. Baris ke-108, merupakan inisialisasi konfigurasi *costmap* secara *local* pada file `local_costmap_params.yaml`
 - f. Baris ke-111, merupakan insialisasi konfigurasi *local planner* yang ada pada file `local_planner_params.yaml`

- g. Baris ke-112, merupakan inisialisasi konfigurasi global planner yang ada pada file `global_planner_params.yaml`
- h. Baris ke-115 hingga ke-117, merupakan *remap* untuk topik yang di *subscribe* oleh *move base* untuk menggunakan topik yang sudah dideklarasikan pada *argument*.

19. Baris ke-119, merupakan penutup dari tag `launch`.

Berdasarkan pembahasan di atas mengenai navigasi, terdapat beberapa poin dari program yang mengambil nilai dari konfigurasi suatu file dengan ekstensi `yaml`. Maka dari itu di bawah ini dijelaskan bentuk implementasi dari parameter-parameter yang digunakan

5.2.2.6 Implementasi Parameter Konfigurasi *Common Costmap*

Implementasi parameter konfigurasi *common costmap* mencakup *costmap* yang ada pada *global* dan *local* secara *default*, jadi secara bawaan baik *global* maupun *local* memiliki parameter bawaan dan pada *file* inilah kedua parameter tersebut dideklarasikan. Hasil dari implementasi dari konfigurasi ini dapat dilihat pada Tabel 5.34.

Tabel 5.34 Parameter Konfigurasi *Common Costmap*

No.	Kode Program
	<i>autonomous_mobile_robot_navigation/param/common_costmap_params.yaml</i>
1	<code>obstacle_range: 10.0 # Maximum range sensor reading in meters</code>
2	<code>that results in an obstacle being placed in the costmap</code>
3	<code>raytrace_range: 10.0 # Outside of this range is considered free</code>
4	<code>space</code>
5	<code># Footprint of the robot with (0,0) being the center (in meters)</code>
6	<code>footprint: [[-0.20, -0.22], [-0.20, 0.22], [0.20, 0.22], [0.20, -</code>
7	<code>0.22]]</code>
8	<code>footprint_padding: 0.05</code>
9	<code># The static map created using SLAM is being published to this</code>
10	<code>topic</code>
11	<code>map_topic: /map</code>
12	<code>map_type: costmap</code>
13	<code>transform_tolerance: 1.0 # Delay in transform tf data that is</code>
14	<code>tolerable in seconds</code>
15	<code>publish_voxel_map: false</code>
16	<code>always_send_full_costmap: true</code>
17	<code>plugins:</code>
18	<code>- { name: obstacle_layer, type: "costmap_2d::ObstacleLayer" }</code>
19	<code>- { name: inflation_layer, type: "costmap_2d::InflationLayer" }</code>
20	<code>obstacle_layer:</code>
21	<code> observation_sources: laser_scan_sensor</code>
22	<code> laser_scan_sensor:</code>
23	<code> {</code>
24	<code> sensor_frame: laser,</code>
25	<code> data_type: LaserScan,</code>

26	topic: scan,
27	marking: true,
28	clearing: true,
29	}
30	
31	inflation_layer:
32	inflation_radius: 1.5 # was 1.0
33	cost_scaling_factor: 1.0 # was 0.5

Tabel di atas merupakan kode program untuk melakukan pemetaan. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1, merupakan jarak ambang batas atau *thresholds* dari halangan atau *obstacle* sebenarnya. Yang berarti robot akan membaca jarak *obstacle* dengan tambahan 2 meter dari jarak sebenarnya, dengan begini *path planning* akan diperbaharui asal tidak melewati nilai radiasi dari parameter ini.
2. Baris ke-2, merupakan jarak *raytraced* yang berarti rentang yang diberikan robot untuk membaca daerah *free space*. Dengan mengaturnya ke 3 meter berarti robot akan mencoba membersihkan ruang di depannya hingga 3 meter dengan pembacaan sensor.
3. Baris ke-5, merupakan footprint yang berarti garis tepi dari bentuk robot yang digunakan. Karena permukaan robot yang digunakan berbentuk persegi panjang maka nilai untuk lebar -0.20 sampai dengan 0.20 dan -0.22 sampai 0.22 untuk panjangnya dengan satuan meter.
4. Baris ke-6, merupakan parameter untuk padding dari footprint robot atau jarak aman robot terhadap benda sekitar, yaitu 0.05 meter.
5. Baris ke-9, merupakan parameter dari topik map.
6. Baris ke-10, merupakan parameter map dengan tipe *costmap*
7. Baris ke-11, merupakan parameter jarak *origin z* yaitu 0.0
8. Baris ke-12, merupakan parameter toleransi dari TF posisi robot yang dapat diterima.
9. Baris ke-13, merupakan deklarasi untuk tidak melakukan *publish voxel* map
10. Baris ke-14, merupakan parameter untuk deklarasi selalu mengirimkan *costmap* secara penuh, yaitu dengan nilai *true*.
11. Baris ke-16 hingga ke-18, merupakan *plugins* yang digunakan pada *costmap* yaitu *obstacle layer* dan *inflation layer*.
12. Baris ke-20, merupakan deklarasi *plugins* buatan dengan nama *obstacle layer*.
 - a. Baris ke-21, merupakan inisialisasi sumber dari observasi halangan,
 - b. Baris ke-22, merupakan deklarasi sumber observasi halangan dengan nilai parameter tertentu,

- c. Baris ke-24, merupakan inisialisasi *frame* dari sensor yang digunakan, yaitu laser,
 - d. Baris ke-25, merupakan inisialisasi tipe data halangan, yaitu LaserScan,
 - e. Baris ke-26, merupakan inisialisasi topik yang digunakan pada *layer obstacle* ini, yaitu *scan*
 - f. Baris ke-27 hingga ke-28, merupakan deklarasi *marking* dan *clearing* bernilai *true* agar halangan dapat diperbaharui. *Marking* berfungsi untuk masukan informasi hambatan ke dalam peta biaya, sedangkan *clearing* berfungsi untuk hapus informasi hambatan dari peta biaya (Pyo, dkk., 2017).
13. Baris ke-31 hingga ke-33, merupakan deklarasi parameter untuk *layer inflasi*, di mana radius yang diberikan akan memberikan jeda terhadap posisi halangan sebenarnya.

5.2.2.7 Implementasi Parameter Konfigurasi *Global Costmap*

Implementasi parameter konfigurasi *global costmap* diperlukan untuk membuat perencanaan global dari data peta yang sudah didapatkan pada proses pemetaan. Hasil dari implementasi dari konfigurasi ini dapat dilihat pada Tabel 5.35.

Tabel 5.35 Parameter Konfigurasi *Global Costmap*

No.	Kode Program
	<i>autonomous_mobile_robot_navigation/param/global_costmap_params.yaml</i>
1	<pre> global_costmap: global_frame: map # Global reference frame for the costmaps robot_base_frame: base_footprint # Base frame of the robot update_frequency: 5.0 publish_frequency: 5.0 width: 70.0 height: 70.0 resolution: 0.1 origin_x: 0.0 #init robot position coordinate x origin_y: 0.0 # init robot position coordinate y static_map: true rolling_window: false #you do not use the costmap to represent your complete environment, but only to represent your local surroundings plugins: - { name: static_layer, type: "costmap_2d::StaticLayer" } - { name: inflater_layer, type: "costmap_2d::InflationLayer" } inflater_layer: inflation_radius: 0.2 # was 0.1 cost_scaling_factor: 1.0 # was 0.5 static_layer: </pre>
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	

23	<code>map_topic: /map</code>
24	<code>subscribe to updates: false</code>

Tabel di atas merupakan kode program untuk melakukan pemetaan. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1, merupakan deklarasi sub nama dari global *costmap*
2. Baris ke-2, merupakan deklarasi dari parameter *frame* secara global untuk *costmap* global, yakni map
3. Baris ke-3, merupakan deklarasi dari parameter basis robot, yaitu *base_footprint*
4. Baris ke-4 hingga ke-5, merupakan deklarasi parameter *update* dan *publish* frekuensi dari halangan global
5. Baris ke-6 hingga ke-7, merupakan deklarasi parameter untuk batas lebar dan panjang *costmap*, yakni 70 meter x 70 meter.
6. Baris ke-8, merupakan deklarasi parameter dari resolusi *costmap*
7. Baris ke-9 hingga ke-10, merupakan deklarasi parameter untuk menentukan posisi robot saat pertama kali dijalankan dengan koordinat x dan y.
8. Baris ke-11, merupakan deklarasi untuk menggunakan peta *static*.
9. Baris ke-12, merupakan deklarasi untuk tidak menggunakan peta global *costmap* sebagai representasi dari keseluruhan lingkungan, diberikan nilai *false* karena *file* map sudah merepresentasikan *costmap* global secara menyeluruh.
10. Baris ke-14 hingga ke-16, merupakan deklarasi *plugins* buatan yang digunakan pada *costmap* global.
11. Baris ke-18 hingga ke-20, merupakan deklarasi *plugin inflater layer*.
12. Baris ke-22 hingga ke-24, merupakan deklarasi untuk *plugin static layer*

5.2.2.8 Implementasi Parameter Konfigurasi *Local Costmap*

Implementasi parameter konfigurasi *local costmap* diperlukan untuk membuat perencanaan secara lokal ketika jalur global terhalang oleh halangan. Hasil dari implementasi dari konfigurasi ini dapat dilihat pada Tabel 5.36.

Tabel 5.36 Parameter Konfigurasi *Local Costmap*

No.	Kode Program
	<i>autonomous_mobile_robot_navigation/param/local_costmap_params.yaml</i>
1	<code>local_costmap:</code>
2	<code> robot_base_frame: base_footprint</code>
3	<code> update_frequency: 5.0</code>
4	<code> publish_frequency: 5.0</code>
5	<code> origin_x: 0.0</code>
6	<code> origin_y: 0.0</code>

7	resolution: 0.05 # was 0.0067
8	static_map: false
9	rolling_window: true #you do not use the costmap to represent your complete environment, but only to represent your local surroundings
10	width: 1.0
11	height: 1.0
12	
13	plugins:
14	- { name: inflater_layer, type: "costmap_2d::InflationLayer" }
15	
16	inflater_layer:
17	inflation_radius: 0.05 # was 0.1
18	cost_scaling_factor: 0.5 # was 0.5

Tabel di atas merupakan kode program untuk melakukan pemetaan. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1, merupakan deklarasi sub name dari *local costmap*
2. Baris ke-2, merupakan deklarasi parameter dari robot *base frame*, yaitu *base_footprint*
3. Baris ke-3 hingga ke-4, merupakan deklarasi untuk parameter *publish* dan *update* frekuensi
4. Baris ke-5 hingga ke-6, merupakan deklarasi untuk parameter yang menyatakan posisi robot saat pertama kali di inisialisasi
5. Baris ke-7, merupakan deklarasi untuk resolusi dari *local costmap*
6. Baris ke-8, merupakan deklarasi untuk parameter apakah map akan berjalan dengan *static*, diberikan nilai *false* karena *local costmap* bergerak secara dinamis
7. Baris ke-9, merupakan parameter untuk memberikan nilai representasi bahwa *costmap* secara lokal tidak merepresentasikan lingkungan globalnya, melainkan lingkungan lokalnya maka dari itu diberikan nilai *true*
8. Baris ke-10 hingga ke-11, merupakan deklarasi untuk lebar dan panjang area lokal yang dibaca oleh sensor
9. Baris ke-13 hingga ke-15, merupakan deklarasi untuk parameter *plugins* yang digunakan
10. Baris ke-17 hingga-19, merupakan deklarasi untuk parameter *plugin inflater layer*

5.2.2.9 Implementasi Parameter Konfigurasi *Global Planner*

Implementasi parameter konfigurasi global *planner* yang terdiri dari konfigurasi untuk tiap-tiap parameter yang dipakai pada perencanaan sebelumnya, jika tidak terdefinisi maka nilai yang dipakai adalah nilai *default*.

Semua kode yang berhubungan dengan Parameter Konfigurasi *Global Planner* dapat dilihat pada Tabel 5.37.

Tabel 5.37 Parameter Konfigurasi Global Planner

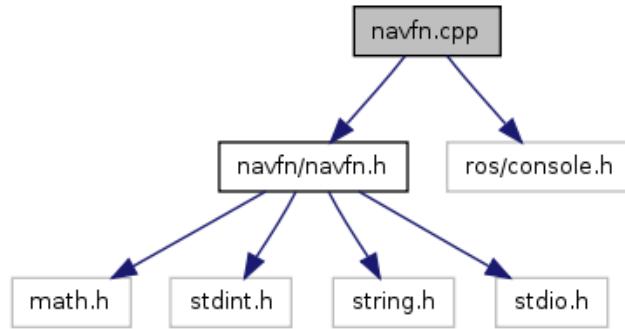
No.	Kode Program
	<i>autonomous_mobile_robot_navigation/param/global_planner.yaml</i>
1	# GLOBAL PLAN
2	NavfnROS:
3	allow_unknown: true #Specifies whether or not to allow navfn to create plans that traverse unknown space
4	planner_window_x: 0.1 #Specifies the x size of an optional window to restrict the planner to
5	planner_window_y: 0.1 #Specifies the y size of an optional window to restrict the planner to
6	default_tolerance: 0.1 #If the goal is in an obstacle, the planer will plan to the nearest point in the radius of default_tolerance
7	visualize_potential: true #for visualize the grid

Tabel di atas merupakan kode program untuk melakukan pemetaan. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-2, merupakan deklarasi sub *name* dengan nama NavfnROS, jadi ketika digunakan *plugins* untuk global *planner* dengan NavfnROS maka sub parameter yang ada di dalamnya akan ikut terpanggil
2. Baris ke-3, merupakan deklarasi untuk menentukan apakah Navfn diperbolehkan untuk membuat perencanaan di luar dari daerah yang tidak diketahui
3. Baris ke-4, merupakan deklarasi untuk menentukan pembatasan dari perencanaan jalur yang dibuat pada koordinat x. Ini berguna untuk membuat perencanaan jalur pada daerah yang sempit
4. Baris ke-5, merupakan deklarasi untuk menentukan pembatasan dari perencanaan jalur yang dibuat pada koordinat y. Ini berguna untuk membuat perencanaan jalur pada daerah yang sempit
5. Baris ke-6, merupakan deklarasi untuk menentukan jarak toleransi antara titik tujuan dengan halangan di sekitarnya
6. Baris ke-7, merupakan deklarasi untuk mengaktifkan visualisasi sehingga kita dapat melihat proses *grid* per *grid* pembacaan dan penentuan jalur global.

5.2.2.10 Implementasi Algoritme Navfn

Implementasi dari algoritme Navfn dibuat ke dalam beberapa kode yang berasal dari paket *default* dari program ROS *Navigation*. Gambar 5.34 merupakan dependensi yang ada pada *file node navfn.cpp*



Gambar 5.34 Dependensi *Node Global Planner Navfn*

Kelas NavfnROS didefinisikan di `navfn_ros.cpp`, dan kelas NavFn didefinisikan di `navfn.cpp`. Aturan penamaan untuk seluruh paket ROS Navigation adalah bahwa kelas dengan akhiran ROS melengkapi kerangka kerja dan data dari sub-proses dan keseluruhan serta proses lainnya. Sirkulasikan, selesaikan pekerjaan sebenarnya dari bagian ini di kelas tanpa akhiran ROS, dan jadilah anggota kelas dengan akhiran ROS. Kode pemrograman dari implementasi Navfn salah satunya seperti yang terlihat pada tabel-tabel yang ada di bawah ini.

Tabel 5.38 *Library Navfn ROS, cost map, msgs, dan plugin*

No	Kode Program
	navfn.cpp
1	<code>#include <navfn/navfn_ros.h></code>
2	<code>#include <pluginlib/class_list_macros.h></code>
3	<code>#include <costmap_2d/cost_values.h></code>
4	<code>#include <costmap_2d/costmap_2d.h></code>
5	<code>#include <sensor_msgs/point_cloud2_iterator.h></code>
6	
7	<code>PLUGINLIB_EXPORT_CLASS(navfn::NavfnROS, nav_core::BaseGlobalPlanner)</code>

Tabel di atas merupakan kode *library* Navfn ROS. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1 hingga ke-5, merupakan *import file library* dari internal *library* ROS, seperti `navfn_ros.h`, `class_list_macros.h`, `cost_val`, `costmap_2d.h`, `point_cloud2_iterator.h`. Dengan akses ke library ini kita dapat menggunakan fungsi-fungsi penting di dalamnya untuk proses navigasi dengan Navfn.
2. Baris ke-7, merupakan export class untuk plugin BaseGlobalPlanner

Tabel 5.39 Program Navfn Kelas Utama *Compute Potensial*

No	Kode Program
	navfn.cpp
1	<code>bool NavfnROS::computePotential(const geometry_msgs::Point& world_point){</code>
2	<code>if(!initialized){</code>
3	<code>ROS_ERROR("This planner has not been initialized yet, but it is being used, please call initialize() before use");</code>

4	return false;
5	}
6	planner_>setNavArr(costmap_>getSizeInCellsX(), costmap_>getSizeInCellsY());
7	planner_>setCostmap(costmap_>getCharMap(), true, allow_unknown_);
8	
9	unsigned int mx, my;
10	if(!costmap_>worldToMap(world_point.x, world_point.y, mx, my))
11	return false;
12	
13	int map_start[2];
14	map_start[0] = 0;
15	map_start[1] = 0;
16	
17	int map_goal[2];
18	map_goal[0] = mx;
19	map_goal[1] = my;
20	
21	planner_>setStart(map_start);
22	planner_>setGoal(map_goal);
23	
24	return planner_>calcNavFnDijkstra();
25	}

Tabel di atas merupakan kode program Navfn dengan fungsi *computePotential*. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1, merupakan deklarasi fungsi *computePotential* dengan parameter *world_point*.
2. Baris ke-2 hingga ke-5, merupakan seleksi kondisi jika variabel global *initialize == false* maka cetak pesan error dan return false.
3. Baris ke-6 hingga ke-7, merupakan inisialisasi planner untuk melakukan set navigation array dan set costmap dengan menggunakan parameter pointer *costmap*.
4. Baris ke-10 hingga ke-11, merupakan seleksi kondisi jika nilai pointer *costmap-> worldToMap == false*, maka return false. Namun jika true berarti poin x, y, mx, dan my di inisialisasi.
5. Baris ke-13 hingga ke-19, merupakan deklarasi dan inisialisasi nilai start dengan 0 dan goal dengan nilai yang baru di dapat dari mx dan my yang sebelumnya.
6. Baris ke-21 hingga ke-22, merupakan inisialisasi planner dengan *setStart* dan *setGoal* dengan nilai *map_start* dan *map_goal*. Dengan ini koordinat awal dan akhir sudah ditentukan.
7. Baris ke-24, merupakan inisialisasi return fungsi pointer planner *calcNavFnDijkstra()* sehingga didapatkan bobot tiap sel.

Di sini pertama-tama inisialisasi peta dengan peta biaya yang diteruskan oleh parameter. Fungsi ini juga menginisialisasi kelas anggota NavFn, kelas ini akan menyelesaikan perhitungan aktual dari rencana global.

Tabel 5.40 Program Navfn Kelas *Initialize*

No.	Kode Program
	navfn.cpp
1	void NavfnROS::initialize(std::string name,
2	costmap_2d::Costmap2D* costmap, std::string global_frame){
3	if(!initialized){
4	costmap_ = costmap;//Global cost map
5	global_frame_ = global_frame;
6	planner_ = boost::shared_ptr<NavFn>(new NavFn(costmap_ -
7	>getSizeInCellsX(), costmap_->getSizeInCellsY()));
8	ros::NodeHandle private_nh("~/ " + name);
9	plan_pub_ = private_nh.advertise<nav_msgs::Path>("plan",
10	1);
11	private_nh.param("visualize_potential",
12	visualize_potential_, false);
13	if(visualize_potential_)
14	potarr_pub_.advertise(private_nh, "potential", 1);
15	private_nh.param("allow_unknown", allow_unknown_, true);
16	private_nh.param("planner_window_x", planner_window_x_,
17	0.0);
18	private_nh.param("planner_window_y", planner_window_y_,
19	0.0);
20	private_nh.param("default_tolerance",
21	default_tolerance_, 0.0);
22	ros::NodeHandle prefix_nh;
23	tf_prefix_ = tf::getPrefixParam(prefix_nh);
24	
25	//Publish the service of make_plan
26	make_plan_srv_ = private_nh.advertiseService("make_plan",
27	&NavfnROS::makePlanService, this);
	//The initialization flag is set to true
	initialized_ = true;
	}
	else
	ROS_WARN("This planner has already been initialized, you
	can't call it twice, doing nothing");
	}
	}

Tabel di atas merupakan kode program Navfn dengan fungsi *initialize*. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1, merupakan deklarasi fungsi initialize dengan parameter name, costmap, dan global_frame.
2. Baris ke-2 hingga ke-27, merupakan seleksi kondisi jika variabel pointer initialize == false maka lakukan inisialisasi global frame, planner, private_nh, plan_pub, parameter, visual potential, tf_prefix, dan make_plan_srv. Terakhir inisialisasi initialize dengan true. Namun jika

ternyata initialize == true, maka return pesan warning kalau planner sudah di inialisasi dan tidak lakukan apapun.

makePlan adalah fungsi yang dipanggil ke perencana global di Movebase. Ini adalah fungsi kunci dari kelas NavfnROS. Ini bertanggung jawab untuk memanggil fungsi termasuk anggota kelas Navfn untuk menyelesaikan perhitungan aktual dan mengontrol seluruh proses rencana global. Bagian terpenting dari inputnya adalah posisi saat ini dan target. Pada tahap persiapan berisi pembersihan rencana sebelum perencanaan, tunggu tf, simpan posisi awal saat ini dan ubah ke sistem koordinat peta, dan atur sel awal pada peta biaya global ke FREE_SPACE.

Tabel 5.41 Program Navfn Kelas *makePlan*

No.	Kode Program
	navfn.cpp
1	bool NavfnROS::makePlan(const geometry_msgs::PoseStamped&
	start, const geometry_msgs::PoseStamped& goal, double
	tolerance, std::vector<geometry_msgs::PoseStamped>& plan){
2	boost::mutex::scoped_lock lock(mutex_);
3	//Check if it is initialized
4	if(!initialized){
5	ROS_ERROR("This planner has not been initialized yet,
	but it is being used, please call initialize() before use");
6	return false;
7	}
8	
9	plan.clear();
10	ros::NodeHandle n;
11	
12	if(tf::resolve(tf_prefix_, goal.header.frame_id) !=
	tf::resolve(tf_prefix_, global_frame_)){
13	ROS_ERROR("The goal pose passed to this planner must be
	in the %s frame. It is instead in the %s frame.",
	tf::resolve(tf_prefix_, global_frame_).c_str(),
	tf::resolve(tf_prefix_, goal.header.frame_id).c_str());
14	return false;
15	}
16	
17	if(tf::resolve(tf_prefix_, start.header.frame_id) !=
	tf::resolve(tf_prefix_, global_frame_)){
18	ROS_ERROR("The start pose passed to this planner must be
	in the %s frame. It is instead in the %s frame.",
	tf::resolve(tf_prefix_, global_frame_).c_str(),
	tf::resolve(tf_prefix_, start.header.frame_id).c_str());
19	return false;
20	}
21	
22	//Starting pose wx, wy
23	double wx = start.pose.position.x;
24	double wy = start.pose.position.y;
25	
26	unsigned int mx, my;
27	if(!costmap_->worldToMap(wx, wy, mx, my)){
28	ROS_WARN("The robot's start position is off the global
	costmap. Planning will always fail, are you sure the robot
	has been properly localized?");

29	return false;
30	}
31	tf::Stamped<tf::Pose> start_pose;
32	tf::poseStampedMsgToTF(start, start_pose);
33	clearRobotCell(start_pose, mx, my);
34	}

Tabel di atas merupakan kode program Navfn dengan fungsi *makePlan*. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1, merupakan deklarasi fungsi *makePlan* dengan parameter *start*, *goal*, *tolerance* dan *plan*.
2. Baris ke-2 hingga ke-5, merupakan seleksi kondisi jika variabel global *initialize == false* maka cetak pesan error dan return false.
3. Baris ke-9 hingga ke-10, merupakan fungsi untuk clear plan yang sudah dibuat.
4. Baris ke-12 hingga ke-20, merupakan seleksi kondisi pengecekan untuk memeriksa apakah TF dari global frame sudah sesuai atau belum
5. Baris ke-23 hingga ke-26, merupakan deklarasi pose *wx*, *wy*, *mx*, dan *my*.
6. Baris ke-27 hingga ke-34, merupakan seleksi kondisi untuk return false atau keluar program dan mengirim pesan warning jika hasil return dari *worldToMap* menghasilkan false. Selanjutnya pose tersebut inisialisasi ke TF dan ROS Message.

Fungsi *calcNavFnDijkstra* ini melengkapi seluruh proses penghitungan jalur dan memanggil beberapa fungsi sub-bagian secara berurutan.

Tabel 5.42 Program Navfn Kelas *calcNavFnDijkstra*

No.	Kode Program
	navfn.cpp
1	bool NavFn::calcNavFnDijkstra(bool atStart){
2	#if 0
3	static char costmap_filename[1000];
4	static int file_number = 0;
5	snprintf(costmap_filename, 1000, "navfn-dijkstra-
6	costmap-%04d", file_number++);
7	savemap(costmap_filename);
8	#endif
9	setupNavFn(true);
10	
11	// calculate the nav fn and path
12	propNavFnDijkstra(std::max(nx*ny/20,nx+ny),atStart);
13	
14	// path
15	int len = calcPath(nx*ny/2);
16	
17	if (len > 0) // found plan
18	{
19	ROS_DEBUG("[NavFn] Path found, %d steps\n", len);
20	return true;

21	}
22	else
23	{
24	ROS_DEBUG("[NavFn] No path found\n");
25	return false;
26	}
27	}

Tabel di atas merupakan kode program Navfn dengan fungsi calcNavFnDijkstra. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1, merupakan deklarasi fungsi calcNavFnDijkstra dengan parameter yang dibutuhkan adalah atStart.
2. Baris ke-2 hingga ke-8, merupakan seleksi kondisi jika state costmap == 0 maka inialisasi costmap dan save costmap.
3. Baris ke-9, merupakan pemanggilan fungsi setupNavFn dengan true. Sehingga dimungkinkan akses ke variabel pointer di fungsi setupNavFn
4. Baris ke-12, merupakan pemanggilan fungsi propNavFnDijkstra dengan parameter nilai maksimal dari nx*ny/20,nx+ny dan aStart
5. Baris ke-15 hingga ke-27, merupakan seleksi kondisi ketika plan sudah ditemukan sehingga kirimkan pesan "Path Found". Namun jika tidak maka kirimkan pesan "No path found".

Fungsi SetupNavFv ini melakukan pengaturan marginal dan pemrosesan lainnya pada array costarr yang dihasilkan oleh "translation", menginisialisasi array potarr dan array gradien gradx serta grady. Selanjutnya, atur nilai sel di empat sisi costarr ke COST_OBS dan tutup peta di sekitar untuk mencegah lintasan di luar batas.

Tabel 5.43 Program Navfn Kelas SetupNavFv

No.	Kode Program
	navfn.cpp
1	void NavFn::setupNavFn(bool keepit){
2	//Reset the value of the potential field matrix potarr
3	for (int i=0; i<ns; i++)
4	{
5	potarr[i] = POT_HIGH;
6	if (!keepit) costarr[i] = COST_NEUTRAL;
7	gradx[i] = grady[i] = 0.0;
8	}
9	COSTTYPE *pc;
10	pc = costarr;
11	
12	for (int i=0; i<nx; i++)
13	*pc++ = COST_OBS;
14	
15	pc = costarr + (ny-1)*nx;
16	for (int i=0; i<nx; i++)
17	*pc++ = COST_OBS;
18	pc = costarr;
19	for (int i=0; i<ny; i++, pc+=nx)

20	<code>*pc = COST_OBS;</code>
21	<code>pc = costarr + nx - 1;</code>
22	
23	<code>for (int i=0; i<ny; i++, pc+=nx)</code>
24	<code> *pc = COST_OBS;</code>
25	<code> //Limited buffer</code>
26	<code> curT = COST_OBS;</code>
27	<code> curP = pb1;</code>
28	<code> curPe = 0;</code>
29	<code> nextP = pb2;</code>
30	<code> nextPe = 0;</code>
31	<code> overP = pb3;</code>
32	<code> overPe = 0;</code>
33	<code> memset(pending, 0, ns*sizeof(bool));</code>
34	<code> //k is the index of the target cell</code>
35	<code> int k = goal[0] + goal[1]*nx;</code>
36	<code> initCost(k,0);</code>
37	<code> pc = costarr;</code>
38	<code> int ntot = 0;</code>
39	<code> for (int i=0; i<ns; i++, pc++)</code>
40	<code> {</code>
41	<code> if (*pc >= COST_OBS)</code>
42	<code> ntot++;</code>
43	<code> }</code>
44	<code> nobs = ntot;</code>
45	<code>}</code>

Tabel di atas merupakan kode program Navfn dengan fungsi setupNavFn. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1, merupakan deklarasi fungsi setupNavFn dengan parameter yang dibutuhkan adalah keepit.
2. Baris ke-3 hingga ke-8, merupakan perulangan untuk menyetel ulang nilai matriks medan potensial potarr.
3. Baris ke-9 hingga ke-13, merupakan deklarasi dan inisialisasi variabel pointer pc = costarr.
4. Baris ke-12 hingga ke-24, merupakan perulangan inisialisasi nilai pc dan di increment lalu inisialisasi dengan COST_OBS.
5. Baris ke-26 hingga ke-38, merupakan inisialisasi beberapa variabel untuk memberikan batasan buffer.
6. Baris ke-39 hingga ke-45, merupakan deklarasi dan inisialisasi nilai k sebagai indeks dari target sel. Selanjutnya panggil fungsi initCost sehingga cost map dapat terdeteksi. Selanjutnya inisialisasi variabel nobs dengan ntot yang sudah diproses sebelumnya.

Fungsi ini mengambil titik target (nilai Potensial telah diinisialisasi ke 0) sebagai titik awal, menyebar ke sel seluruh peta, dan mengisi array potarr hingga titik awal ditemukan

Tabel 5.44 Program Navfn Kelas propNavFnDijkstra

No.	Kode Program
	navfn.cpp
1	bool NavFn::propNavFnDijkstra(int cycles, bool atStart){
2	int nwv = 0;
3	int nc = 0;
4	int cycle = 0;
5	
6	int startCell = start[1]*nx + start[0];
7	pb = curP;
8	i = curPe;
9	while (i-- > 0)
10	updateCell(*pb++);
11	
12	if (displayInt > 0 && (cycle % displayInt) == 0)
13	displayFn(this);
14	
15	curPe = nextPe;
16	nextPe = 0;
17	pb = curP; // swap buffers
18	curP = nextP;
19	nextP = pb;
20	
21	if (curPe == 0)
22	{
23	curT += priInc;
24	curPe = overPe;
25	overPe = 0;
26	pb = curP; // swap buffers
27	curP = overP;
28	overP = pb;
29	}
30	if (atStart)
31	if (potarr[startCell] < POT_HIGH)
32	break;
33	}
34	ROS_DEBUG("[NavFn] Used %d cycles, %d cells visited
35	(%d%%), priority buf max %d\n",
36	cycle,nc,(int)((nc*100.0)/(ns-nobs)),nwv);
37	if (cycle < cycles) return true; // finished up here
38	else return false;
39	}

Tabel di atas merupakan kode program Navfn dengan fungsi *propNavFnDijkstra*. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1, merupakan deklarasi fungsi *propNavFnDijkstra* dengan parameter yang dibutuhkan adalah *cycle* dan *atStart*.
2. Baris ke-2 hingga ke-8, merupakan deklarasi dan inisialisasi variabel yang dibutuhkan.
3. Baris ke-9 hingga ke-34, merupakan perulangan dalam array *potarr* untuk dapat mencerminkan "seberapa jauh telah pergi" dan "situasi rintangan

terdekat" dan memberikan dasar untuk perhitungan jalur akhir. Perbarui potarr secara berulang dalam satu lingkaran, dan tentukan kondisinya.

4. Baris ke-35 hingga 39, merupakan seleksi kondisi dimana Jika himpunan saat ini sedang dipropagasi dan himpunan yang dipropagasi berikutnya kosong, itu berarti perambatan tidak dapat dilanjutkan, mungkin ada rintangan yang tidak dapat dilewati atau keadaan lain, dan keluar.

updateCell digunakan untuk memperbarui nilai Potensial sel tunggal, pertama-tama dapatkan nilai potensial dari titik-titik tetangga di sekitar sel saat ini, dan ambil nilai terkecil dan simpan di ta. Selanjutnya, lakukan penilaian hanya ketika sel saat ini bukan penghalang fatal, itu akan menyebar ke sekitarnya, jika tidak maka akan berhenti dan tidak menyebar. Saat menghitung nilai Potensial titik saat ini, ada dua situasi, yaitu, perlu untuk membandingkan "nilai absolut dari perbedaan antara nilai pot minimum dari tetangga kiri dan kanan dan nilai pot minimum dari sisi atas dan bawah. tetangga yang lebih rendah" dan "nilai costarr dari sel saat ini".

Tabel 5.45 Program Navfn Kelas updateCell

No.	Kode Program
	navfn.cpp
1	inline void NavFn::updateCell(int n){
2	// get neighbors
3	float u,d,l,r;
4	l = potarr[n-1];
5	r = potarr[n+1];
6	u = potarr[n-nx];
7	d = potarr[n+nx];
8	float ta, tc;
9	if (l<r) tc=l; else tc=r;
10	if (u<d) ta=u; else ta=d;
11	// do planar wave update
12	if (costarr[n] < COST_OBS) // don't propagate into
13	obstacles
14	{
15	float hf = (float)costarr[n]; // traversability factor
16	float dc = tc-ta; // relative cost between ta,tc
17	if (dc < 0) // ta is lowest
18	{
19	dc = -dc;
20	ta = tc;
21	}
22	// do planar wave update
23	if (costarr[n] < COST_OBS)
24	{
25	float hf = (float)costarr[n];
26	float dc = tc-ta;
27	if (dc < 0) // ta is lowest
28	{
29	dc = -dc;
30	ta = tc;
31	}
32	// calculate new potential
33	float pot;

34	if (dc >= hf)
35	pot = ta+hf;
36	else
37	{
38	float d = dc/hf;
39	float v = -0.2301*d*d + 0.5307*d + 0.7040;
40	pot = ta + hf*v;
41	}
42	// now add affected neighbors to priority blocks
43	if (pot < potarr[n])
44	{
45	float le = INVSQRT2*(float)costarr[n-1];
46	float re = INVSQRT2*(float)costarr[n+1];
47	float ue = INVSQRT2*(float)costarr[n-nx];
48	float de = INVSQRT2*(float)costarr[n+nx];
49	potarr[n] = pot;
50	if (pot < curT) // low-cost buffer block
51	{
52	if (l > pot+le) push_next(n-1);
53	if (r > pot+re) push_next(n+1);
54	if (u > pot+ue) push_next(n-nx);
55	if (d > pot+de) push_next(n+nx);
56	}
57	else // overflow block
58	{
59	if (l > pot+le) push_over(n-1);
60	if (r > pot+re) push_over(n+1);
61	if (u > pot+ue) push_over(n-nx);
62	if (d > pot+de) push_over(n+nx);
63	}
64	}
65	}
66	}

Tabel di atas merupakan kode program Navfn dengan fungsi *updateCell*. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1, merupakan deklarasi fungsi *updateCell* dengan parameter yang dibutuhkan adalah a.
2. Baris ke-2 hingga ke-10, merupakan deklarasi dan inisialisasi variabel yang diambil dari potarr dengan nilai [n-1], [n+1], [n-nx], dan potarr[n+nx]. Variabel ini berguna untuk mencari tetangga terdekat di ke empat sisi terdekat. Kemudian deklarasi variabel ta dan tc bertujuan untuk menemukan nilai yang terendah dan tetangganya yang terendah.
3. Baris ke-33 hingga ke-49, merupakan proses untuk melakukan perhitungan ke potensial yang baru. Dengan seleksi kondisi jika planar terlalu besar gunakan ta-only update, namun jika terlalu kecil maka update 2 interpolasi tetangga. Kemudian gunakan perhitungan quadratic approximation untuk mempercepat melalui pencarian tabel.
4. Baris ke-50 hingga ke-66, merupakan bagian untuk menambahkan tetangga sekarang yang terpengaruh ke blok prioritas dan proses tetangga tersebut.

Fungsi *Calc Path* ini bertanggung jawab untuk memilih beberapa titik sel berdasarkan *array potarr* untuk menghasilkan jalur perencanaan global akhir, mulai dari titik target dan mencari lintasan optimal ke titik awal sepanjang arah gradien *cost* berjalan optimal.

Tabel 5.46 Program Navfn Fungsi *Calc Path*

No.	Kode Program
	navfn.cpp
1	int NavFn::calcPath(int n, int *st) {
2	if (npathbuf < n){
3	if (pathx) delete [] pathx;
4	if (pathy) delete [] pathy;
5	pathx = new float[n];
6	pathy = new float[n];
7	npathbuf = n;
8	}
9	if (st == NULL) st = start;
10	int stc = st[1]*nx + st[0];
11	float dx=0;
12	float dy=0;
13	npath = 0;//Path point index
14	for (int i=0; i<n; i++){
15	int nearest_point=std::max(0, std::min(nx*ny-
16	1, stc+(int) round(dx)+(int) (nx*round(dy))));
17	if (potarr[nearest_point] < COST_NEUTRAL){
18	pathx[npath] = (float)goal[0];
19	pathy[npath] = (float)goal[1];
20	return ++npath; // done!
21	}
22	if (stc < nx stc > ns-nx){
23	ROS_DEBUG("[PathCalc] Out of bounds");
24	return 0;
25	}
26	pathx[npath] = stc%nx + dx;//x index
27	pathy[npath] = stc/nx + dy;//y index
28	npath++;
29	bool oscillation_detected = false;
30	if(npath > 2 &&
31	pathx[npath-1] == pathx[npath-3] &&
32	pathy[npath-1] == pathy[npath-3])
33	{
34	ROS_DEBUG("[PathCalc] oscillation detected, attempting
35	fix.");
36	oscillation_detected = true;
37	}
38	int stcnx = stc+nx;
39	int stcpn = stc-nx;
40	if (potarr[stc] >= POT_HIGH
41	potarr[stc+1] >= POT_HIGH
42	potarr[stc-1] >= POT_HIGH
43	potarr[stcnx] >= POT_HIGH
44	potarr[stcnx+1] >= POT_HIGH
45	potarr[stcnx-1] >= POT_HIGH
46	potarr[stcpn] >= POT_HIGH
	potarr[stcpn+1] >= POT_HIGH
	potarr[stcpn-1] >= POT_HIGH

```

47         oscillation_detected)
48     {
49         ROS_DEBUG("[Path] Pot fn boundary, following grid
50         (%0.1f/%d)", potarr[stc], npath);
51         int minc = stc;
52         int minp = potarr[stc];
53         int st = stcp - 1;
54         if (potarr[st] < minp) {minp = potarr[st]; minc=st; }
55         st++;
56         if (potarr[st] < minp) {minp = potarr[st]; minc=st; }
57         st++;
58         if (potarr[st] < minp) {minp = potarr[st]; minc=st; }
59         st = stc-1;
60         if (potarr[st] < minp) {minp = potarr[st]; minc=st; }
61         st = stcnx-1;
62         if (potarr[st] < minp) {minp = potarr[st]; minc=st; }
63         st++;
64         if (potarr[st] < minp) {minp = potarr[st]; minc=st; }
65         st++;
66         if (potarr[st] < minp) {minp = potarr[st]; minc=st; }
67         stc = minc;
68         dx = 0;
69         dy = 0;
70         ROS_DEBUG("[Path] Pot: %0.1f pos: %0.1f,%0.1f",
71         potarr[stc], pathx[npath-1], pathy[npath-1]);
72         if (potarr[stc] >= POT_HIGH){
73             ROS_DEBUG("[PathCalc] No path found, high
74             potential");
75             return 0;
76         }
77         }else {
78             gradCell(stc);
79             gradCell(stc+1);
80             gradCell(stcnx);
81             gradCell(stcnx+1);
82             float x1=(1.0-dx)*gradx[stc] + dx*gradx[stc+1];
83             float x2=(1.0-dx)*gradx[stcnx] + dx*gradx[stcnx+1];
84             float x=(1.0-dy)*x1 + dy*x2; // interpolated x
85             float y1=(1.0-dx)*grady[stc] + dx*grady[stc+1];
86             float y2=(1.0-dx)*grady[stcnx] + dx*grady[stcnx+1];
87             float y=(1.0-dy)*y1 + dy*y2; // interpolated y
88             ROS_DEBUG("[Path]%0.2f,%0.2f%0.2f,%0.2f
89             %0.2f,%0.2f %0.2f,%0.2f; final x=%0.3f, y=%0.3f\n",
90             gradx[stc], grady[stc], gradx[stc+1], grady[stc+1],
91             gradx[stcnx], grady[stcnx], gradx[stcnx+1], grady[stcnx+1],
92             x, y);
93             if (x == 0.0 && y == 0.0)
94             {
95                 ROS_DEBUG("[PathCalc] Zero gradient");
96                 return 0;
97             }
98             // move in the right direction
99             float ss = pathStep/hypot(x, y);
100             dx += x*ss;
101             dy += y*ss;
102             if (dx > 1.0) { stc++; dx -= 1.0; }
103             if (dx < -1.0) { stc--; dx += 1.0; }

```


98	if (dy > 1.0) { stc+=nx; dy -= 1.0; }
99	if (dy < -1.0) { stc-=nx; dy += 1.0; }
100	}
101	}
102	ROS_DEBUG("[PathCalc] No path found, path too long");
	return 0;
103	}

Tabel di atas merupakan kode program Navfn dengan fungsi *calcPath*. Berikut ini penjelasan mengenai kode program tersebut,

1. Baris ke-1, merupakan deklarasi fungsi *calcPath* dengan parameter yang dibutuhkan adalah a.
2. Baris ke-2 hingga ke-8, merupakan proses pengecekan jalur dengan array-array yang sudah di operasikan.
3. Baris ke-9 hingga ke-13, merupakan proses setup posisi awal dan indeks rekaman titik awal stc. Kemudian mengatur offset dan indeks array untuk titik jalur.
4. Baris ke-14 hingga ke-20, merupakan perulangan berdasarkan indeks cycles. Kemudian cek apakah perencanaan jalur sudah mendekati tujuan, jika ya maka return ++npath yang berarti pencarian jalur telah selesai.
5. Baris ke-21 hingga ke-35, merupakan pengecekan jika baris pertama atau terakhir di luar batas. Kemudian juga proses menambahkan ke way point dan deteksi getaran, apakah posisi langkah tertentu sama dengan langkah sebelumnya.
6. Baris ke-36 hingga 103, merupakan seleksi kondisi apakah 8 node di sekitar node yang saat ini mencapai node memiliki nilai hambatan, jika demikian, langsung arahkan stc ke node dengan nilai potensial terendah di antara 8 node. Kemudian periksa nilai terkecil di antara delapan titik tetangga. Selanjutnya seleksi kondisi jika ada gradien yang baik, hitung gradiennya secara langsung dan temukan simpul berikutnya di sepanjang arah gradien. Selanjutnya periksa gradien nol apakah gagal atau tidak, *move* ke arah yang benar, dan periksa overflow dari nilai dx dan dy. Terakhir return pesan "No path found, path too long" dan return 0.

BAB 6 PENGUJIAN

Pada bab pengujian, penulis menunjukkan bagaimana sistem ini dapat diimplementasikan dengan pengujian untuk menilai apakah sistem yang dirancang telah sesuai dengan tujuan maupun rumusan masalah yang diajukan. Hasil dari pengujian juga digunakan sebagai hasil penarikan kesimpulan pada bab selanjutnya, berikut merupakan tahapan pengujian pada penelitian ini.

6.1 Pengujian Kinerja *Hoverboard*

Pengujian ini dilakukan dengan mengontrol *motherboard hoverboard* untuk memutar motor sehingga didapatkan data berupa kecepatan minimum dan maksimum. Nilai kecepatan diukur berdasarkan data *odometry* yang dihasilkan oleh sensor *encoder*. Selain kecepatan, pengujian ini juga melakukan uji ketahanan baterai dengan mengisi penuh baterai yang kemudian dihabiskan. Kemudian dicatat berapa lama baterai dapat bertahan berdasarkan aktivitas yang dilakukan.

6.1.1 Tujuan Pengujian

Pengujian ini bertujuan untuk mencari tahu apakah *hoverboard* sebagai sistem penggerak utama layak digunakan sebagai aktuator dari sistem. Hasil keluaran yang diharapkan dari pengujian ini adalah informasi kecepatan yang dapat dicapai oleh *hoverboard* dalam satuan radian per sekon. Selain kecepatan, ketahanan baterai juga menjadi poin penting pada pengujian ini.

6.1.2 Prosedur Pengujian

Prosedur pengujian kinerja *hoverboard* dilakukan dengan melakukan prosedur yang telah disesuaikan sebagai berikut.

1. Menyambungkan perangkat *hoverboard motherboard* dengan Jetson Nano yang masing-masing dihubungkan ke sumber daya yang sudah penuh.
2. Menghidupkan perangkat *hoverboard motherboard* dan Jetson Nano yang sudah terpasang.
3. Menjalankan kode *hoverboard driver launcher* dan *rqt robot steering* untuk menjalankan dan menampilkan data *odometry*.
4. Memasukkan nilai kecepatan (m/s) minimum hingga maksimum melalui perintah pada aplikasi *rqt robot steering*.
5. Mencatat aktivitas dan lama baterai *hoverboard* digunakan.
6. Melakukan analisis perbandingan dan perhitungan nilai radian per sekon dari setiap kecepatan yang diujikan.
7. Melakukan analisis ketahanan dan perhitungan pada hasil pengujian daya tahan baterai.

6.1.3 Hasil Pengujian

Hasil pengujian dilakukan dengan mencari nilai maksimum sebanyak 20 kali percobaan dan daya tahan baterai sistem sebanyak 10 kali percobaan.

6.1.3.1 Hasil Pengujian Kinerja *Hoverboard* Berdasarkan Kecepatan Minimum dan Maksimum Motor

Hasil pengujian dari kinerja *hoverboard* berdasarkan kecepatan minimum dan maksimum motor dapat dilihat pada Tabel 6.1.

Tabel 6.1 Data Hasil Uji Kinerja *Hoverboard* Berdasarkan Kecepatan Minimum dan Maksimum Motor

No.	Label	Masukan		Keterangan tercapai	
		Kecepatan (m/s)	RPM	Ya	Tidak
1	Minimum	0.1	11	✓	
2	Minimum	0.1	11	✓	
3	Minimum	0.1	11	✓	
4	Minimum	0.1	11	✓	
5	Minimum	0.1	11	✓	
6	Minimum	0.2	22	✓	
7	Minimum	0.2	22	✓	
8	Minimum	0.2	22	✓	
9	Minimum	0.2	22	✓	
10	Minimum	0.2	22	✓	
11	Maksimum	4	466	✓	
12	Maksimum	4	466	✓	
13	Maksimum	4	466	✓	
14	Maksimum	4	466	✓	
15	Maksimum	4	466	✓	
16	Maksimum	8	932	✓	
17	Maksimum	8	932	✓	
18	Maksimum	8	932	✓	

19	Maksimum	8	932	✓	
20	Maksimum	8	932	✓	
Jumlah				20	0

6.1.3.2 Hasil Pengujian Kinerja *Hoverboard* Berdasarkan Daya Tahan Baterai

Untuk hasil pengujian dari kinerja *hoverboard* berdasarkan daya tahan baterai dapat dilihat pada Tabel 6.2.

Tabel 6.2 Data Hasil Uji Kinerja *Hoverboard* Berdasarkan Daya Tahan Baterai

No.	Aktivitas motor (m/s)	Charge		Discharge	
		Voltase	Lama waktu pengisian (menit)	Voltase	Lama waktu pemakaian (menit)
1	0.2	42.2	102	32.4	140
2	0.2	42.2	101	32.5	142
3	0.5	42.2	103	32.6	130
4	0.5	42.2	102	32.2	128
5	1	42.2	105	32.4	110
6	1	42.2	104	32.1	108
7	2	42.2	106	32.3	82
8	2	42.2	107	32.2	81
9	4	42.2	108	32.4	62
10	4	42.2	110	32.5	64
Rata-rata		42.2	103	32.36	104.7

6.1.4 Analisis Pengujian

Analisis pengujian dilakukan dengan mengolah hasil uji untuk mendapatkan informasi dengan menganalisis perhitungan dan kinerja sistem.

6.1.4.1 Analisis Pengujian Kinerja *Hoverboard* Berdasarkan Kecepatan Minimum dan Maksimum Motor

Pada hasil pengujian kinerja *hoverboard* yang dilakukan untuk mencari tahu kecepatan minimum dan maksimum yang dapat dicapai oleh sistem, dari 20 kali

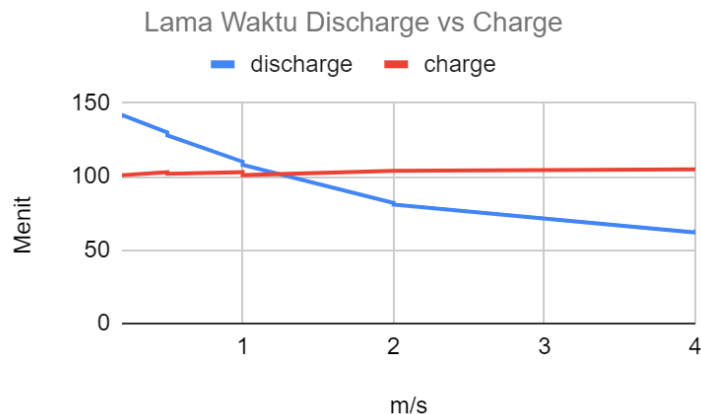
pengujian dengan kecepatan yang berbeda-beda didapatkan nilai akurasi yang ditunjukkan pada persamaan 6.1.

$$Akurasi = \frac{20-0}{20} \times 100 = 100\% \quad (6.1)$$

Didapatkan hasil akurasi 100% dari pengujian kinerja *hoverboard* berdasarkan keberhasilan kecepatan motor dalam mencapai kecepatan 0.1 m/s – 0.2 m/s dan 4 m/s – 8 m/s. Dari 20 kali percobaan, didapatkan minimum putaran roda 0.1 m/s atau sekitar 11 RPM dan maksimum putaran roda hingga 8 m/s atau sekitar 932 RPM. Hasil uji yang positif ini menunjukkan bahwa dalam kecepatan minimum dan maksimum, *hoverboard* dapat dijadikan sebagai motor penggerak utama. Hal ini dikarenakan kemampuannya untuk berjalan dengan RPM yang sangat rendah atau sangat tinggi sehingga dimungkinkannya penempatan sistem di berbagai tempat, baik yang luas ataupun yang sempit.

6.1.4.2 Analisis Pengujian Kinerja *Hoverboard* Berdasarkan Daya Tahan Baterai

Pada hasil pengujian kinerja *hoverboard* yang dilakukan untuk mencari tahu daya tahan baterai (*life battery*), dari 10 kali pengujian dengan aktivitas yang berbeda-beda didapatkan informasi mengenai *discharge* dan *charge* baterai. Data ini disajikan dalam bentuk grafik pada Gambar 6.1.



Gambar 6.1 Uji Daya Tahan Baterai

Pada Gambar 6.1 terlihat bahwa aktivitas kecepatan roda mempengaruhi waktu *discharge*, sedangkan waktu *charge* cenderung stabil. Pada waktu *discharge* diketahui dapat mencapai 150 menit dengan aktivitas 0.2 m/s dan terus menurun ke 60 menit dengan aktivitas 4 m/s. Hal ini dikarenakan semakin cepat motor berputar maka semakin besar pula daya (watt) yang dibutuhkan. Sedangkan untuk waktu *charge* diketahui memiliki nilai yang cenderung stabil di 100 - 110 menit.

6.2 Pengujian Daya Angkut Sistem

Pengujian ini dilakukan dengan memberikan *stress weight* atau tekanan beban pada bagian atas sistem untuk mendapatkan maksimum daya angkut dengan

mempertimbangkan kecepatan dari sistem. Nilai daya angkut diukur berdasarkan total beban yang diduduki oleh barang.

6.2.1 Tujuan Pengujian

Pengujian ini bertujuan untuk mencari tahu batasan berat yang dapat diangkut oleh sistem. Hasil keluaran yang diharapkan dari pengujian ini adalah informasi mengenai berat maksimum yang dapat diberikan, dengan mempertimbangkan kecepatan sistem saat beroperasi. Dengan mengetahui maksimum daya angkut dari sistem, kita dapat mengetahui batasan dari berat yang dapat diangkut robot.

6.2.2 Prosedur Pengujian

Prosedur pengujian fungsionalitas dari *odometry* dilakukan dengan melakukan prosedur yang telah disesuaikan sebagai berikut.

1. Menyambungkan perangkat *hoverboard motherboard* dengan Jetson Nano yang masing-masing dihubungkan ke sumber daya yang sudah penuh.
2. Menghidupkan perangkat *hoverboard motherboard* dan Jetson Nano yang sudah terpasang.
3. Menjalankan kode *hoverboard driver launcher* dan rqt robot *steering* untuk menjalankan dan menampilkan data *odometry*.
4. Memasukkan nilai kecepatan (m/s) minimum antar 0.1 m/s dan 0.2 m/s melalui perintah pada aplikasi rqt robot *steering*.
5. Memberikan beban di bagian alas penutup sistem dengan 25kg, 35kg, 45kg, dan 55kg.
6. Melakukan analisis perhitungan daya angkut sistem.

6.2.3 Hasil Pengujian

Hasil pengujian dari daya angkut sistem dapat dilihat pada Tabel 6.3.

Tabel 6.3 Data Hasil Uji Daya Angkut Sistem

No.	Masukan		Beban 25 kg		Beban 3 kg		Beban 45 kg		Beban 55 kg	
	Kecepatan (m/s)	Jarak tempuh (m)	Ya	Tidak	Ya	Tidak	Ya	Tidak	Ya	Tidak
1	0.1	5	✓		✓			✓		✓
2	0.1	5	✓		✓			✓		✓
3	0.2	5	✓		✓		✓			✓
4	0.2	5	✓		✓		✓			✓
5	0.3	5	✓		✓		✓			✓

6	0.3	5	✓		✓		✓			✓
7	0.4	5	✓		✓		✓			✓
8	0.4	5	✓		✓		✓		✓	
9	0.5	5	✓		✓		✓		✓	
10	0.5	5	✓		✓		✓		✓	
Jumlah			10	0	10	0	8	2	3	7

6.2.4 Analisis Pengujian



Gambar 6.2 Pengujian Daya Angkut Sistem

Dapat dilihat pada Gambar 6.2 merupakan dokumentasi dari pengujian daya angkut sistem, yakni dengan menaiki bagian atas robot menggunakan berat bervariasi mulai dari 25 kilogram hingga 55 kilogram. Penentuan berat maksimum ini ditentukan berdasarkan spesifikasi dari *hoverboard* yang mampu menerima beban maksimal 50 kg dan roda *caster* 3 inci yang mampu menerima beban maksimal 30 kg. Titik temu dari maksimal beban diputuskan dari total jumlah roda dibagi 2 yang menghasilkan 40 kg, sehingga rentang berat ditentukan di antara nilai 40 kg.

$$Akurasi = \frac{10+10+8+3}{40} \times 100 = 77.5\% \quad (6.2)$$

Pada hasil pengujian daya angkut sistem yang terdapat pada Tabel 6.3. Dilakukan uji untuk mencari tahu berat maksimum yang dapat diangkut oleh sistem, dari 10 kali pengujian dengan kecepatan yang berbeda-beda didapatkan jumlah keberhasilan sebesar 33 dari 40. Berdasarkan perhitungan akurasi dari persamaan 6.2 didapatkan akurasi 77.5% antara waktu tempuh tanpa beban dan waktu dengan beban. Dari pengujian ini dapat disimpulkan bahwa beban maksimal aman yang dapat di angkut oleh sistem adalah 35 kilogram. Tidak menutup kemungkinan untuk mengangkat beban 45 kilogram akan tetapi kemungkinan sistem mati karena terlalu berat mengangkat beban akan terjadi.

6.3 Pengujian Akurasi *Odometry* Pada Sensor *Encoder* Motor BLDC

Penelitian ini diuji dengan menggunakan sebuah purwarupa *autonomous mobile robot* dengan ukuran yang disesuaikan dengan kebutuhan di gudang penyimpanan. Pengujian nantinya diuji berdasarkan fungsionalitas dan kinerja. Pengujian ini dilakukan untuk mengetahui tingkat ketelitian dari nilai putaran *odometry* pada motor BLDC dengan tujuan akhir untuk mencapai jarak tertentu dengan *error* yang kecil.

6.3.1 Tujuan Pengujian

Tujuan dari pengujian fungsionalitas *odometry encoder* motor BLDC yaitu untuk mengetahui kinerja dari motor BLDC dalam berputar sebanyak putaran tertentu yang sudah disesuaikan dengan *input* dari program rqt. Besarnya *error* dari motor umumnya disebabkan kesalahan konfigurasi motor *hoverboard*, seperti ukuran diameter roda robot dan jarak satu roda ke roda lainnya, sehingga kesalahan perhitungan menyebabkan putaran motor dalam mencapai jarak tertentu tidak sesuai dengan yang ada divisualisasi RViz.

6.3.2 Prosedur Pengujian

Prosedur pengujian fungsionalitas dari *odometry* dilakukan dengan melakukan prosedur yang telah disesuaikan sebagai berikut:

7. Menyambungkan perangkat Motor BLDC dengan motor *driver*, yaitu *hoverboard motherboard* dan Jetson Nano yang masing-masingnya disambungkan sumber daya baterai.
8. Menghidupkan perangkat *hoverboard motherboard* dan Jetson Nano yang sudah terpasang dengan sistem operasi Linux Tegra.
9. Memasang penanda pada tanda titik di motor BLDC untuk digunakan menghitung putaran roda dengan RPM yang sudah dihitung.
10. Menjalankan kode *hoverboard driver launcher* dan rqt robot *steering* untuk menjalankan dan menampilkan data *odometry*.
11. Memasukkan nilai kecepatan (m/s) melalui perintah pada aplikasi rqt robot *steering* untuk menggerakkan motor dengan kecepatan yang bervariasi.
12. Mengamati jumlah putaran *per cycle* selama waktu observasi, yaitu 20 detik dan mengamati nilai *odometry* yang ditampilkan oleh *topic /odom*.
13. Melakukan analisis perbandingan dan perhitungan nilai RPM menggunakan *RPM converter* dari setiap kecepatan yang diujikan.

6.3.3 Hasil Pengujian

Hasil pengujian dari *odometry encoder* motor BLDC disajikan pada tabel dan juga terdapat dokumentasi pada Tabel 6.4 untuk menghitung jumlah putaran per detik dari nilai *odometry* motor BLDC.

Tabel 6.4 Pengujian *Odometry* Motor BLDC

Uji akurasi sensor <i>Odometry</i>						
No.	Input	<i>Odometry</i>		Observasi		
	Kecepatan (m/s)	Kecepatan rotasi (rad/s)	RPM sebenarnya	Waktu observasi (s)	Jumlah putaran roda (cyc)	RPM
1	0.2	2.30384	22	20	7.5	22.5
2	0.4	4.7124	45	20	15.14	45.4
3	0.6	7.12096	68	20	23.5	70.5
4	0.8	9.63424	92	20	31.4	94.2
5	1	12.0428	115	20	39	117
6	1.2	14.45136	138	20	46.8	140.4
7	1.4	16.7552	160	20	54.6	163.8
8	1.6	19.26848	184	20	62.3	186.9
9	1.8	21.78176	210	20	70.7	212.1
10	2	24.39976	233	20	79	237

Tabel 6.5 Hasil Akurasi Pengujian Fungsionalitas Nilai *Odometry*

Pengukuran kesalahan	Total RPM <i>odometry</i>	Total RPM observasi
	1267	1289.8
WAPE	1.80%	
Nilai akurasi	98.20%	

6.3.4 Analisis Pengujian

Dari hasil pengujian yang dilakukan untuk mencari tahu ketepatan nilai *odometry* motor BLDC terhadap nilai putaran sebenarnya, dari 10 kali pengujian dengan kecepatan yang berbeda-beda didapatkan nilai akurasi sebesar 98.20% seperti yang dapat dilihat pada Tabel 6.5. Hasil akurasi ini dihitung berdasarkan persamaan 2.23 dan 2.24 mengenai WAPE dan persentase keberhasilan. Berdasarkan hasil akurasi yang didapatkan disimpulkan bahwa kinerja dan konfigurasi *odometry* dari motor BLDC yang dipakai oleh sistem tergolong sangat baik dan presisi.

6.4 Pengujian Akurasi Sensor RPLIDAR A1

Pengujian fungsionalitas data laser *point cloud* yang menggunakan sensor RPLIDAR A1 sebagai *input* dari sistem yang menghasilkan data berupa jarak. Dalam pengujian ini dilakukan serangkaian uji terhadap sistem dari laser RPLIDAR secara langsung. Pengujian ini dilakukan dengan pengambilan data dari keluaran data *point cloud* yang dihasilkan sesuai dengan kondisi lingkungan sekitar.

6.4.1 Tujuan Pengujian

Tujuan dari pengujian fungsionalitas laser *point cloud* dari sistem ini adalah untuk mengetahui bagaimana kinerja dari perangkat RPLIDAR yang digunakan dalam pengambilan data jarak berdasarkan kondisi lingkungan sekitar.

6.4.2 Prosedur Pengujian

Prosedur pengujian fungsionalitas dari RPLIDAR dilakukan dengan melakukan prosedur yang telah disesuaikan sebagai berikut:

1. Menyambungkan perangkat RPLIDAR A1 dengan Jetson Nano menggunakan adapter UART ke *port* USB.
2. Menghidupkan perangkat Jetson Nano yang sudah terpasang dengan sistem operasi *Linux Tegra*.
3. Menghubungkan laptop dengan Jetson Nano menggunakan VNC Viewer.
4. Menjalankan kode pada *driver* `rplidar_ros` untuk menampilkan grafik bentuk laser dan jarak dari pengambilan laser.
5. Mengamati hasil data yang diambil dari *point cloud* laser RPLIDAR untuk mengetahui kinerja dan ketepatan nilai pada perangkat lunak dengan jarak sebenarnya.
6. Melakukan analisis perbandingan dan perhitungan data jarak perangkat lunak `rplidar_ros` dengan pengukuran jarak sebenarnya dari setiap jarak yang diujikan.

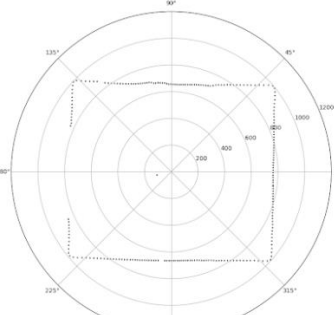
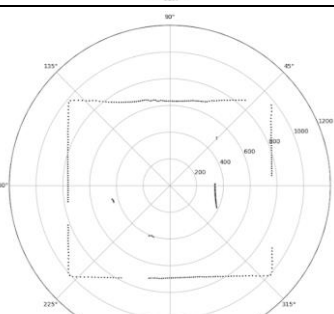
6.4.3 Hasil Pengujian

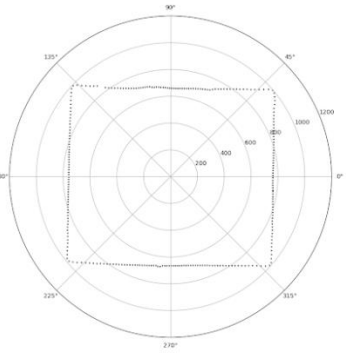
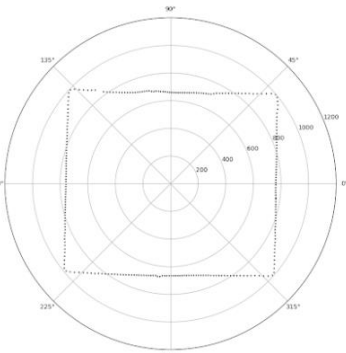
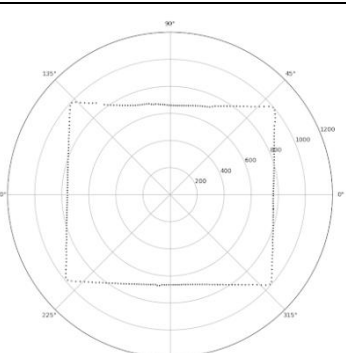
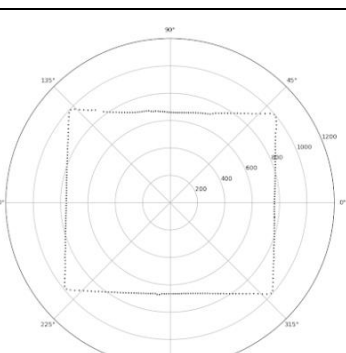
Hasil pengujian dari fungsionalitas perangkat RPLIDAR dilakukan sebanyak sepuluh kali percobaan dan hasil dari pengujian bisa dilihat pada Gambar 6.3 dan Tabel 6.6.



Gambar 6.3 Kondisi Lingkungan Sekitar Pada Pengujian Jarak Data RPLIDAR

Tabel 6.6 Pengujian Fungsionalitas RPLIDAR

Uji akurasi jarak sensor RPLIDAR A1						
No.	Hasil Pengamatan	Halangan	Pengukuran Observasi		RPLIDAR A1	
			Sudut ($^{\circ}$)	Jarak (cm)	Sudut ($^{\circ}$)	Jarak (cm)
1		Terdekat	180 $^{\circ}$	9.7	180	0
2		Terjauh	0 $^{\circ}$	6000	180	5998

3		Tembok	0°	76	0	77
4		Tembok	90°	65	90	66.3
5		Tembok	180°	75	180	76.4
6		Tembok	270°	65	270	66.5

7		Objek 1 (kertas)	180°	43	180	43.7
8		Objek 1 (kertas)	0°	60	0	61.3
9		Objek 2 (besi)	135°	50	141	51.5
10		Objek 2 (besi)	315°	42	330	43.5

Tabel 6.7 Hasil Akurasi Pengujian Fungsionalitas RPLIDAR

Pengukuran kesalahan	Total jarak observasi	Total jarak RPLIDAR
	6485.7	6474.2
WAPE	0.18%	
Nilai akurasi	99.82%	

6.4.4 Analisis Pengujian

Dari hasil pengujian yang dilakukan untuk mencari tahu kesesuaian pembacaan jarak oleh sensor RPLIDAR terhadap jarak sebenarnya. Dari 10 kali pengujian dengan jarak benda yang berbeda-beda didapatkan nilai akurasi sebenar 99.82% seperti yang dapat dilihat pada Tabel 6.6. Hasil akurasi ini dihitung berdasarkan persamaan 2.23 dan 2.24 mengenai WAPE dan persentase keberhasilan. Berdasarkan hasil akurasi yang didapatkan dapat disimpulkan bahwa kinerja dan konfigurasi dari sensor RPLIDAR yang dipakai oleh sistem tergolong sangat baik dan sangat presisi.

6.5 Pengujian Akurasi Sensor IMU GY-521

Pengujian fungsionalitas data IMU menggunakan sensor GY-521 sebagai *input* dari sistem yang menghasilkan data berupa akselerasi. Dalam pengujian ini dilakukan serangkaian uji terhadap sistem dari sensor GY-521 secara langsung. Pengujian ini dilakukan dengan pengambilan data dari keluaran data IMU berupa akselerasi sudut x, y, dan z yang dihasilkan sesuai dengan percepatan yang dilakukan oleh sistem.

6.5.1 Tujuan Pengujian

Tujuan dari pengujian fungsionalitas data IMU dari sistem ini adalah untuk mengetahui bagaimana kinerja dari perangkat GY-521 yang digunakan dalam pengambilan perubahan kecepatan atau percepatan berdasarkan pergerakan sistem.

6.5.2 Prosedur Pengujian

Prosedur pengujian fungsionalitas dari GY-521 dilakukan dengan melakukan prosedur yang telah disesuaikan sebagai berikut:

1. Menyambungkan perangkat sensor GY-521 dengan Jetson Nano menggunakan *port* I2C yang terletak pada *port* 3 dan 5.
2. Menghidupkan perangkat Jetson Nano yang sudah terpasang dengan sistem operasi Linux Tegra .
3. Menghubungkan laptop dengan Jetson Nano secara virtual menggunakan VNC Viewer.

4. Memastikan bahwa sensor GY-521 sudah dilakukan kalibrasi sebelum dilakukan pengujian.
5. Menjalankan kode `mpu6050_driver.launch` pada *terminal* di folder *driver* `mpu6050_driver` untuk menjalankan *node* `mpu6050` dan `i2c` sehingga melakukan *publish* topik `/imu/data`.
6. Melakukan *rostopic echo* pada topik `/imu/data` sehingga dapat mengamati perubahan nilai percepatan.
7. Mengamati hasil data yang diambil dari IMU GY-521 pada saat diam ataupun bergerak robot dengan kecepatan yang sudah ditentukan untuk mengetahui kinerja dan ketepatan nilai pada perangkat lunak dengan jarak sebenarnya.
8. Melakukan analisis perbandingan dan perhitungan data jarak perangkat lunak yang didapat dari data IMU dengan pengukuran jarak sebenarnya dari setiap jarak yang diujikan menggunakan data yang didapat dari sensor IMU *smartphone*.

6.5.3 Hasil Pengujian

Hasil pengujian dari fungsionalitas perangkat GY-521 dilakukan sebanyak sepuluh kali percobaan dengan percepatan diam dan bergerak masing-masing 0.2 m/s, 0.4 m/s, dan 0.6 m/s. Hasil dari pengujian dapat dilihat pada Gambar 6.4 dan Tabel 6.8.

```
orientation:
  x: 0.0
  y: 0.0
  z: 0.0
  w: 0.0
orientation_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
angular_velocity:
  x: 0.0018652373692
  y: -0.00253139366396
  z: -0.00359724345617
angular_velocity_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
linear_acceleration:
  x: 0.102985844016
  y: 0.0694555714726
  z: 9.72018718719
linear_acceleration_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
---
```

Gambar 6.4 Pembacaan Dari Sensor GY-521 Pada Data Akselerasi

Tabel 6.8 Pengujian Fungsionalitas Sensor GY-521

Uji akurasi sensor GY-521 IMU 6050						
No.	Smartphone			Sensor GY-521		
	Xi (m/s)	Yi (m/s)	Zi (m/s)	Xj (m/s)	Yj (m/s)	Zj (m/s)
1	0.06	0.05	9.74	0.103	0.152	9.821
2	0.07	0.04	9.68	0.037	0.151	9.820
3	0.06	0.04	9.67	0.028	0.029	9.831

4	0.04	0.07	9.71	0.025	0.014	9.804
5	0.05	0.07	9.72	0.038	0.152	9.959
6	0.06	0.06	9.71	0.038	0.017	9.808
7	0.04	0.07	9.71	0.028	0.018	9.794
8	0.06	0.09	9.71	0.030	0.012	9.808
9	0.05	0.08	9.72	0.041	0.022	9.833
10	0.05	0.09	9.76	0.038	0.018	9.822

Tabel 6.9 Hasil Akurasi Pengujian Fungsionalitas GY-521

Pengukuran kesalahan	Total nilai sensor vs Total nilai <i>smartphone</i>					
	Xi	Yi	Zi	Xj	Yj	Zj
	0.406	0.584	98.299	0.54	0.66	97.13
WAPE tiap sumbu	24.84%		11.49%		11.49%	
WAPE keseluruhan	12.51%					
Nilai akurasi	87.49%					

6.5.4 Analisis Pengujian

Dari hasil pengujian yang dilakukan untuk mencari tahu ketetapan pembacaan percepatan oleh sensor GY-521 terhadap percepatan robot sebenarnya untuk mendapatkan keakuratan ketika melakukan lokalisasi. Dari 10 kali pengujian dengan variasi percepatan masing-masing diam atau 0 m/s, 0.2 m/s, 0.4 m/s, dan 0.6 m/s didapatkan nilai akurasi sebesar 87.49% seperti yang dapat dilihat pada Tabel 6.9. Hasil akurasi dihitung berdasarkan persamaan 2.23 dan 2.24 mengenai WAPE dan persentase keberhasilan. Berdasarkan hasil akurasi yang didapatkan dapat disimpulkan bahwa kinerja dan konfigurasi dari sensor GY-521 yang dipakai oleh sistem tergolong sangat baik dan presisi.

6.6 Pengujian Bentuk Peta Dari Hasil Pemetaan Menggunakan Algoritme Hector SLAM

Pada pengujian pemetaan menggunakan Hector SLAM dilakukan dengan melakukan beberapa kali proses pengambilan data peta pada sistem. Proses pengambilan data peta ini disimpan ke dalam *file* dengan ekstensi khusus yang nantinya dapat digunakan pada proses navigasi. Pengujian ini dilakukan dengan mengaktifkan program pemetaan dan melakukan manuver robot secara manual menggunakan *rqt robot steering* untuk mendapatkan hasil yang maksimal.

6.6.1 Tujuan Pengujian

Tujuan dari pengujian pemetaan menggunakan Hector SLAM ini adalah untuk mengetahui tingkat akurasi proses pemetaan dari sistem yang dibuat

menggunakan algoritme SLAM yang dipilih, yakni Hector SLAM. Selain itu pengujian ini juga bertujuan untuk mengetahui kecocokan penggunaan metode algoritme ini terhadap studi area yang digunakan, yaitu basemen gedung G FILKOM yang memiliki luas lebih lebar dibanding dengan batas maksimal laser, sehingga diharapkan didapatkan kesimpulan berupa kesesuaian penggunaan algoritme untuk pemetaan berdasarkan kondisi lingkungan yang dijadikan bahan studi.

6.6.2 Prosedur Pengujian

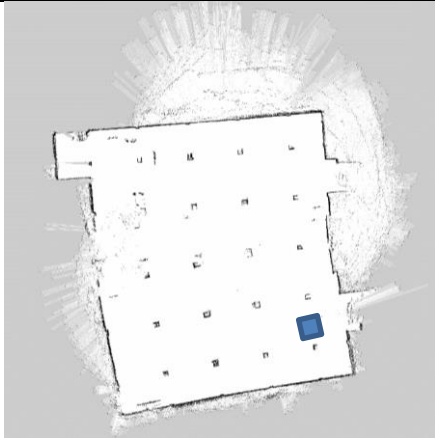
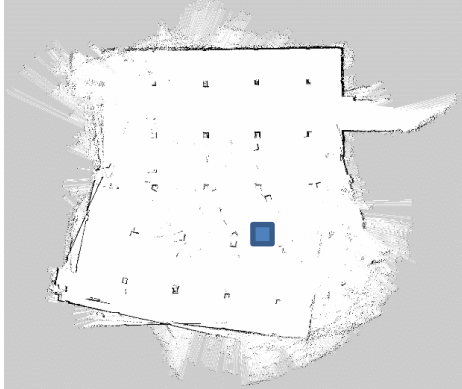
Prosedur pengujian pemetaan menggunakan Hector SLAM dilakukan dengan melakukan prosedur yang telah disesuaikan sebagai berikut:

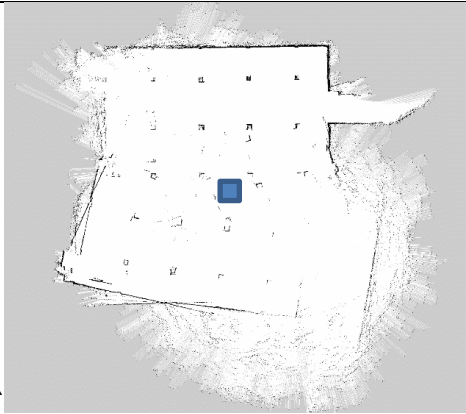
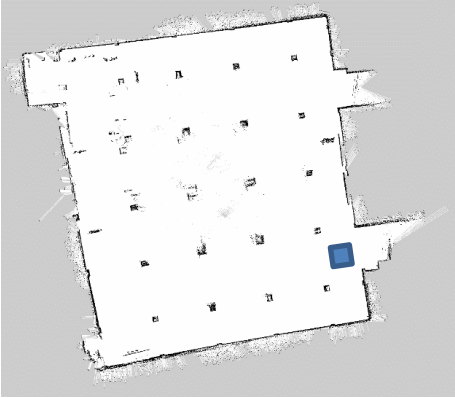
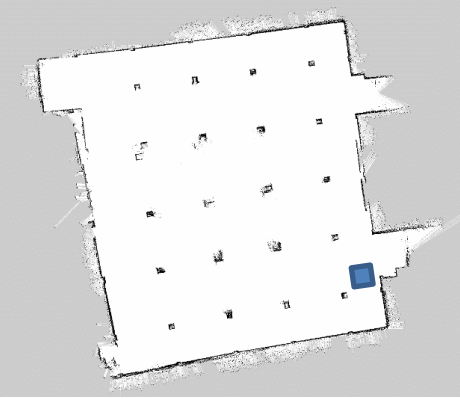
1. Meletakkan posisi robot pada posisi yang sudah ditandai sebagai titik awal pada basemen gedung G FILKOM.
2. Menyambungkan perangkat sensor GY-521 dengan Jetson Nano menggunakan *port* I2C yang terletak pada *port* 3 dan 5.
3. Menghubungkan perangkat RPLIDAR dengan Jetson Nano menggunakan *port* USB.
4. Menghidupkan perangkat Jetson Nano yang sudah terpasang dengan sistem operasi Linux Tegra.
5. Menghidupkan perangkat *hoverboard motherboard* yang sudah terpasang dengan Jetson Nano melalui UART TTL *Converter*.
6. Menghubungkan laptop dengan Jetson Nano secara virtual menggunakan VNC *Viewer*.
7. Menjalankan kode program `map_hector.launch` pada *terminal* untuk menjalankan semua *node* yang dibutuhkan pada proses pemetaan.
8. Melakukan kontrol terhadap pergerakan robot dengan kecepatan bervariasi antara 0.2 m/s hingga 0.6 m/s pada aplikasi *rqt robot steering* yang muncul bersamaan dengan aplikasi Rviz karena sudah terpenggil menjadi satu pada *file launcher*.
9. Mengamati hasil data yang sudah diperoleh pada aplikasi visualisasi Rviz hingga semua area basemen selesai di pindai oleh laser.
10. Menyimpan hasil pemetaan ke dalam folder "*maps*" dengan ekstensi *pgm*.
11. Melakukan perulangan pada pembuatan peta dengan melakukan perbedaan posisi memulai pemetaan, yakni dari pojok kiri, tengah, dan pojok kanan.
12. Melakukan analisis terhadap peta yang sudah dihasilkan dengan membandingkan dari tiap hasil peta terhadap denah basemen gedung G FILKOM.

6.6.3 Hasil Pengujian

Hasil pengujian dari pemetaan menggunakan Hector SLAM dilakukan sebanyak 5 kali percobaan dengan posisi awal pemetaan yang berbeda-beda. Kotak berwarna biru menandakan posisi awal robot sebagai penanda. Hasil dari pengujian dapat dilihat pada Tabel 6.10.

Tabel 6.10 Hasil Pengujian Pemetaan Menggunakan Hector SLAM

No.	Skenario Pengujian	Hasil Pemetaan Hector SLAM	Sesuai	
			Ya	Tidak
1	Memulai pemetaan dengan lokasi seperti pada kotak biru dengan track perjalanan langsung menuju ke tengah basemen hingga kembali ke posisi awal dengan kecepatan rata-rata 0.2 m/s		✓	
2	Memulai pemetaan dengan lokasi seperti pada kotak biru dengan track perjalanan dari tengah bawah langsung menuju area pinggir dan kembali lagi ke tengah bawah dengan kecepatan rata-rata 0.4 m/s		✓	

3	Memulai pemetaan dengan lokasi seperti pada kotak biru dengan track perjalanan dari tengah langsung menuju area pinggir dan kembali lagi ke tengah dengan kecepatan rata-rata 0.3 m/s			✓
4	Memulai pemetaan dengan lokasi seperti pada kotak biru dengan track perjalanan dari pojok kanan berputar di antara pojok dinding dengan kecepatan rata-rata 0.4 m/s		✓	
5	Memulai pemetaan dengan lokasi seperti pada kotak biru dengan track perjalanan dari pojok kanan berputar di antara pojok dinding dengan kecepatan rata-rata 0.3 m/s		✓	
Jumlah			4	1

6.6.4 Analisis Pengujian

Berdasarkan pengujian yang dilakukan dari proses pemetaan area basemen gedung G FILKOM yang dilakukan sebanyak 5 kali percobaan dengan posisi dan kecepatan yang berbeda-beda, didapatkan hasil pengujian berupa 5 buah peta dengan hasil gambar yang berbeda-beda berdasarkan perbedaan kondisi yang dilakukan. Berdasarkan bentuk dari denah gedung G FILKOM yang terlihat pada dapat disimpulkan bahwa dari kelima percobaan yang dilakukan 4 dari 5 kali percobaan sesuai bentuk peta dengan denah yang ada. Kemudian dihitung akurasi

dari 5 kali percobaan berdasarkan persamaan 2.24, didapatkan hasil akurasi yang terdapat pada persamaan 6.3.

$$Akurasi = \frac{4}{5} \times 100 = 80\% \quad (6.3)$$

6.7 Pengujian Akurasi dan Waktu Komputasi Dari Proses Navigasi Global Menggunakan Algoritme Navfn

Pada hasil akurasi manuver dan waktu komputasi pada proses navigasi global dan lokal menggunakan algoritme Navfn saat tidak ada halangan, dilakukan dengan melakukan beberapa kali proses percobaan navigasi pada sistem. Pengujian ini dilakukan sebanyak 5 kali percobaan dengan titik *goal* yang berbeda-beda. Keberhasilan dari pengujian diwakilkan dengan kecepatan waktu komputasi yang dilakukan oleh algoritme Navfn dan akurasi manuver robot saat berjalan dari titik awal menuju titik akhir. Pada pengujian ini waktu manuver robot untuk sampai ke titik tujuan tidak dihitung karena hal tersebut berhubungan dengan pengaturan kecepatan robot. Dalam pengujian ini kecepatan robot diatur dengan nilai minimal 0 m/s dan maksimal 1 m/s sehingga bagaimanapun kondisinya kecepatan manuver robot bergantung pada hasil akurasi manuvernya terhadap jalur yang sudah dibuat, sehingga karena hal tersebut diputuskan untuk menguji hasil manuvernya saja.

6.7.1 Tujuan Pengujian

Tujuan dari pengujian ini yaitu untuk mengetahui seberapa besar akurasi manuver pergerakan robot dan waktu komputasi yang dilakukan oleh sistem ketika melakukan navigasi secara *autonomous* tanpa kemudi dari *user* yang hanya memasukkan *input* berupa titik tujuan. Selain itu pengujian ini juga sebagai penentu apakah robot AMRN ini layak untuk digunakan pada area gudang kurir karena jika pergerakannya tidak stabil maka berakibat tabrakan dengan benda lain.

6.7.2 Prosedur Pengujian

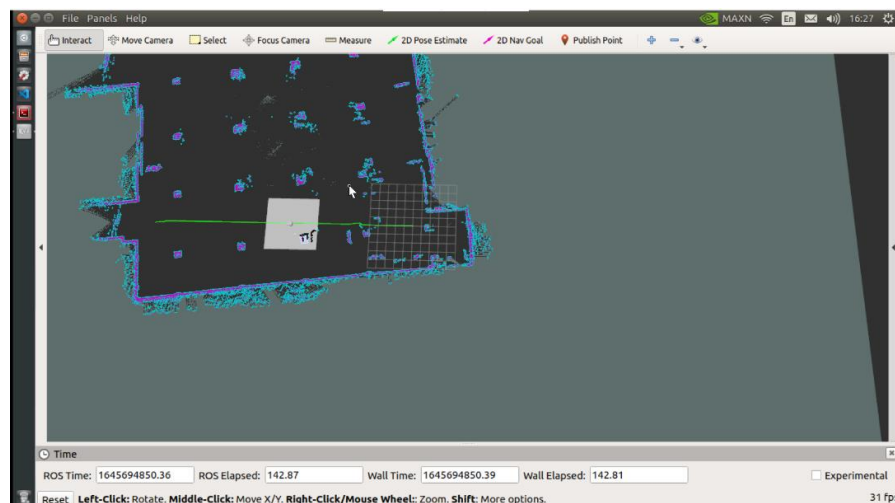
Prosedur pengujian fungsionalitas dari RPLIDAR dilakukan dengan melakukan prosedur yang telah disesuaikan sebagai berikut:

1. Meletakkan posisi robot pada posisi yang sudah ditandai sebagai titik awal pada basemen gedung G FILKOM.
2. Menyambungkan perangkat sensor GY-521 dengan Jetson Nano menggunakan *port* I2C yang terletak pada *port* 3 dan 5.
3. Menghubungkan perangkat RPLIDAR dengan Jetson Nano menggunakan *port* USB.
4. Menghidupkan perangkat Jetson Nano yang sudah terpasang dengan sistem operasi Linux Tegra.

5. Menghidupkan perangkat *hoverboard motherboard* yang sudah terpasang dengan Jetson Nano melalui UART TTL *Converter*.
6. Menghubungkan laptop dengan Jetson Nano secara virtual menggunakan VNC *Viewer*.
7. Menjalankan kode program *autonomous_nav.launch* pada *terminal* untuk menjalankan semua *node* yang dibutuhkan pada proses navigasi.
8. Memasukkan *input* estimasi posisi dan *nav goal* yang ada pada *tools* RViz.
9. Mengamati kecepatan waktu komputasi yang dilakukan oleh sistem dalam membuat perencanaan jalur global.
10. Mengamati pergerakan manuver robot untuk mengetahui apakah robot bergerak sudah sesuai dengan jalur global yang sudah dibuat.
11. Melakukan perulangan pada proses navigasi dengan posisi awal dan tujuan yang berbeda-beda.
12. Melakukan analisis terhadap semua hasil navigasi yang sudah dihasilkan seperti analisis pergerakan robot dan waktu komputasi pada perencanaan globalnya.

6.7.3 Hasil Pengujian

Hasil pengujian dari navigasi robot menggunakan perencanaan global Navfn dilakukan sebanyak 5 kali percobaan dengan posisi awal navigasi yang berbeda-beda. Hasil pengujian direpresentasikan dengan gambar posisi awal dan akhir untuk melihat seberapa jauh posisi robot berubah dari yang seharusnya. Pada Gambar 6.5 merupakan antarmuka aplikasi RViz yang digunakan sebagai visualisasi dari proses pengujian navigasi menggunakan algoritme Navfn. Pada Tabel 6.11 dan Tabel 6.12 merupakan hasil dari pengujian yang dilakukan.

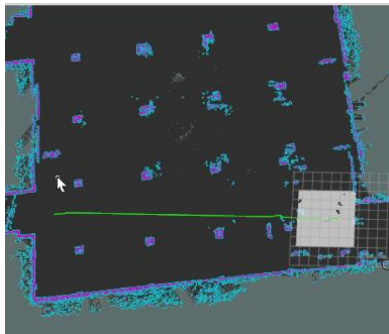
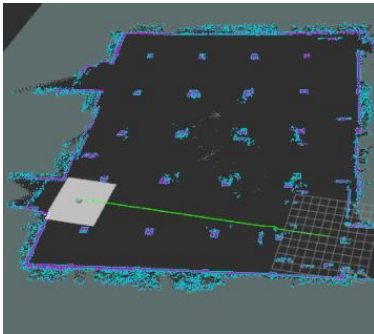
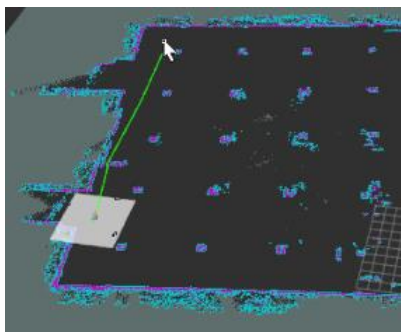
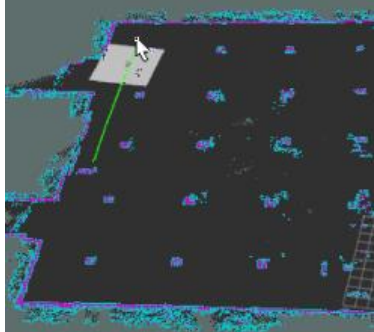
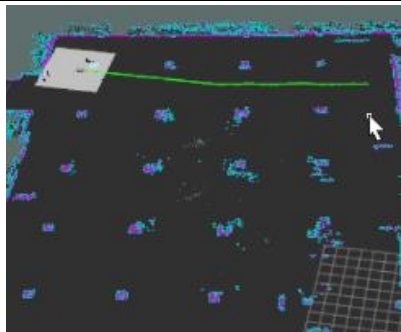
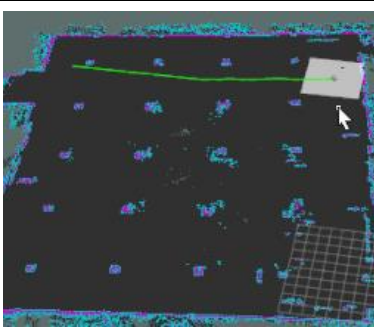


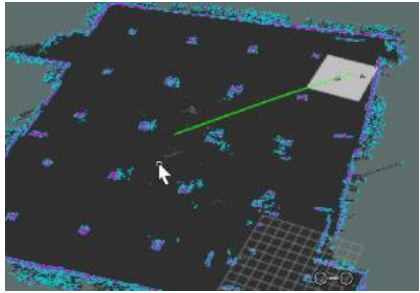
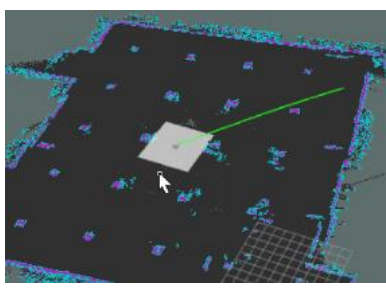
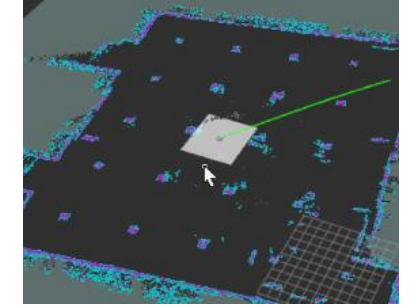
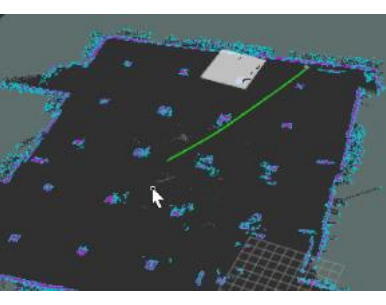
Gambar 6.5 Navigasi Global Menggunakan Aplikasi RViz untuk *Input* Goals

Tabel 6.11 Pengujian Waktu Komputasi Algoritme Navfn

No.	Waktu pemberian <i>goals</i> (detik)	Selesai membuat <i>path</i> (detik)	Selisih
1	64.34	65.24	0.9
2	96.74	97.37	0.63
3	191.18	191.79	0.61
4	64.38	64.96	0.58
5	25.89	26.4	0.51
Rata-rata waktu komputasi (detik)			0.646 detik

Tabel 6.12 Pengujian Akurasi Algoritme Navfn

No.	Gambar posisi awal robot	Gambar posisi akhir robot	Keterangan
1			Berhasil
2			Berhasil
3			Berhasil

4			Berhasil
5			Gagal
Keberhasilan			80%

6.7.4 Analisis Pengujian

Pengujian akurasi dan waktu komputasi ini dilakukan sebanyak 5 kali pada masing-masing perbedaan posisi untuk mencari tahu akurasi dan waktu komputasi yang dibutuhkan oleh algoritme Navfn dalam menentukan jalur global. Kemudian dihitung akurasi dari 5 kali percobaan berdasarkan persamaan 2.24, didapatkan hasil akurasi yang terdapat pada persamaan 6.4.

$$Akurasi = \frac{4}{5} \times 100 = 80\% \quad (6.4)$$

Dari perhitungan tersebut hasil dari pengujian menghasilkan akurasi yang cukup baik berkat algoritme yang digunakan dan lokalisasi yang digunakan, yakni mencapai 80%. Untuk waktu komputasi, berdasarkan persamaan 6.7 yang diambil selisih dari setiap sampel pengujian dan kemudian dirata-rata. Didapatkan menghasilkan 0.646 detik atau sekitar 646 *millisecond*.

$$Mean\ waktu\ komputasi = \frac{(w1+w2+w3+w4+w5)}{5} \quad (6.5)$$

$$Mean\ waktu\ komputasi = \frac{0.9+0.63+0.61+0.58+0.51}{5} \quad (6.6)$$

$$Mean\ waktu\ komputasi = 0.646 \quad (6.7)$$

Nilai rata-rata dari waktu komputasi ini menunjukkan bahwa penggunaan algoritme Navfn tidak memerlukan *resource* yang besar serta waktu komputasi yang cepat tidak lebih dari 1 detik untuk menentukan jalur global.

BAB 7 PENUTUP

Dalam bab penutup, penulis menyampaikan hasil yang telah didapatkan selama melakukan penelitian ini. Adanya hasil tersebut, memungkinkan penulis untuk menarik kesimpulan atas rumusan masalah yang sebelumnya diutarakan. Serta ada pemberian saran untuk penelitian ini agar ke depannya dapat dikembangkan kembali menjadi lebih baik.

7.1 Kesimpulan

Atas hasil pengujian yang penulis dapatkan selama melakukan penelitian ini, maka penulis dapat menarik kesimpulan untuk menjawab rumusan masalah yang ada. Berikut adalah kesimpulan dari penulis:

1. Pada pengujian kinerja *hoverboard* berdasarkan kecepatan minimum - maksimum, telah dilakukan sebanyak 20 kali dan menghasilkan nilai akurasi 100%. Pada pengujian ini dapat disimpulkan bahwa kecepatan minimum yang dijalankan sistem adalah 0.1 m/s dan kecepatan maksimum yang dapat dijalankan sistem adalah 8 m/s. Untuk kinerja *hoverboard* berdasarkan daya tahan baterai, telah dilakukan sebanyak 10 kali dan menghasilkan informasi berupa rata-rata lama pengisian baterai adalah 103 menit dan rata-rata lama pemakaian baterai adalah 104.7 menit tergantung pada aktivitas motor saat sistem digunakan. Sedangkan pada pengujian daya angkut sistem, telah dilakukan sebanyak 10 kali dan menghasilkan nilai selisih sebesar 75.5%. Dari pengujian ini ditentukan berat maksimum adalah 35 kg. Pengujian ini bertujuan untuk mengetahui apakah sistem dapat berjalan dengan baik ketika sedang membawa barang dengan beban maksimum 35 kg. Dengan kemampuan sistem yang bergerak hingga 8 m/s atau sekitar 1000 RPM, memiliki daya tahan baterai yang kuat dan mampu membawa beban hingga 35 kg. Disimpulkan bahwa perangkat *hoverboard* layak dan cocok digunakan sebagai sistem penggerak utama pada sistem navigasi otonom untuk mendistribusikan paket di dalam gudang ekspedisi.
2. Pada pengujian akurasi pembacaan sensor terhadap keluaran data *raw* yang di hasilkan pada tiap sensor, yakni *encoder*, GY-521, dan RPLIDAR telah dilakukan sebanyak 10 kali dan menghasilkan nilai akurasi pada tiap sensornya. Untuk *odometry* didapatkan nilai akurasi sebesar 98.20%, untuk IMU didapatkan nilai akurasi sebesar 87.49%, dan untuk RPLIDAR didapatkan nilai akurasi sebesar 99.82%. Dari ketiga hasil akurasi tersebut dapat disimpulkan bahwa penggunaan sensor dan perangkat lunak sistem berjalan dengan sangat baik dan dapat menunjang sistem *autonomous mobile robot*.
3. Pada pengujian pemetaan menggunakan algoritme Hector SLAM, telah dilakukan sebanyak 5 kali dan menghasilkan nilai akurasi sebesar 80%. Hanya terdapat satu buah peta yang gagal dibuat dikarenakan posisi

permulaan saat membuat peta. Ditemukan permasalahan dalam pemosisian saat memulai jika robot. Ketika robot berada di luar area di sekitar tembok atau area sekitar tidak dapat terbaca oleh laser RPLIDAR, maka hasil peta cenderung buruk dikarenakan algoritme Hector SLAM tidak dapat memberikan data TF (*transformation*) dengan akurat, sehingga posisi robot pada sistem tidak sinkron terhadap posisi sebenarnya. Namun selama area sekitar robot terdapat halangan berupa tembok, maka proses pemetaan berjalan dengan baik.

4. Pada pengujian navigasi statis menggunakan perencanaan global dengan algoritme Navfn, telah dilakukan sebanyak 5 kali dan menghasilkan akurasi sebesar 80%. Dari 5 kali percobaan navigasi, terdapat 1 kali percobaan yang gagal. Hal ini dikarenakan pada percobaan tersebut posisi awal navigasi, dimulai dari tengah basemen yang mana RPLIDAR tidak dapat membaca area tembok terdekat, sehingga lokalisasi dengan AMCL mengalami *error*. Sedangkan untuk waktu komputasi, pemilihan algoritme Navfn untuk perencanaan global sudah tepat dikarenakan rendahnya rata-rata waktu komputasi yang dibutuhkan dan sedikitnya *resource* yang digunakan untuk melakukan perencanaan global.

7.2 Saran

Berdasarkan dengan kesimpulan yang telah dijabarkan sebelumnya dari penelitian ini, ada beberapa saran untuk mengembangkan purwarupa sistem ini di masa depan sehingga kekurangan dari penelitian ini bisa disempurnakan. Saran dari penelitian ini dijabarkan kembali sebagai berikut.

1. Penggunaan sensor RPLIDAR dengan spesifikasi luas pembacaan yang lebih tinggi, dapat digunakan dalam mengatasi permasalahan jangkauan laser ketika ditempatkan di dalam gudang yang lebih luas dan sedikit halangan statis.
2. Penggunaan sistem kemudi *steering Ackerman* pada sistem *autonomous mobile robot* untuk mempermudah manuver ketika berada diposisi yang sempit dan terdapat banyak halangan.
3. Tidak menggunakan algoritme Hector SLAM dan AMCL untuk melakukan pemetaan ataupun navigasi pada ruangan atau area yang minim dengan tembok, karena ketika Hector SLAM tidak membaca area tembok sekitar maka hasil lokalisasi akan menjadi buruk.
4. Menambahkan variasi pada pengujian pemetaan dan navigasi sehingga didapatkan lebih banyak informasi mengenai algoritme yang digunakan.
5. Menambahkan perencanaan jalur lokal untuk menghindari halangan bergerak dinamis agar sistem dapat lebih fleksibel dan bisa menyesuaikan dengan perubahan lingkungan sekitarnya.