TUGAS AKHIR – EE 184801

SISTEM PERENCANAAN RUTE PADA DOMESTIC SERVICE ROBOT UNTUK PENCARIAN OBJEK RUMAH TANGGA

Muhammad Sholahuddin Al-Ayubi
NRP 07111640000078

Dosen Pembimbing
Dr. Ir. Hendra Kusuma, M.Eng.Sc.
Muhammad Attamimi, B.Eng. M.Eng. Ph. D.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

TUGAS AKHIR – EE 184801

SISTEM PERENCANAAN RUTE PADA *DOMESTIC SERVICE ROBOT* UNTUK PENCARIAN OBJEK RUMAH TANGGA.

Muhammad Sholahuddin Al-Ayubi
NRP 07111640000078

Dosen Pembimbing
Dr. Ir. Hendra Kusuma, M.Eng.Sc.
Muhammad Attamimi, B.Eng. M.Eng. Ph. D.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

FINAL PROJECT – **EE 184801**

PATH PLANNING SYSTEM IN DOMESTIC SERVICE ROBOT FOR HOUSEHOLD OBJECT SEARCH.

Muhammad Sholahuddin Al-Ayubi
NRP 07111640000078

Supervisor
Dr. Ir. Hendra Kusuma, M.Eng.Sc.
Muhammad Attamimi, B.Eng. M.Eng. Ph. D.

DEPARTMENT OF ELECTRICAL ENGINEERING
Faculty of Electrical Technology and Intelligence Informatics
Institut Teknologi Sepuluh Nopember
Surabaya 2020

PERNYATAAN KEASLIAN TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul :

"Sistem Perencanaan Rute pada *Domestic Service Robot* untuk Pencarian Objek Rumah Tangga."

adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diizinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

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

Surabaya, 31 Mei 2020

Muhammad Sholahuddin Al-Ayubi
NRP. 07111640000078

Halaman ini sengaja dikosongkan

SISTEM PERENCANAAN RUTE PADA *DOMESTIC SERVICE ROBOT* UNTUK PENCARIAN OBJEK RUMAH TANGGA

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagai Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik

Pada

Bidang Studi Elektronika
Departemen Teknik Elektro
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Menyetujui:

Dosen Pembimbing I

Dr. Ir. Hendra Kusuma, M.Eng.Sc.
NIP 196409021989031003

SURABAYA
Juli 2020

SISTEM PERENCANAAN RUTE PADA *DOMESTIC SERVICE ROBOT* UNTUK PENCARIAN OBJEK RUMAH TANGGA

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan Untuk
Memperoleh Gelar Sarjana Teknik

Pada

Bidang Studi Elektronika
Departemen Teknik Elektro
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Menyetujui:

Dosen Pembimbing II

Muhammad Attamimi, B.Eng., M.Eng., Ph.D.
NIP. 198503272019031006

SURABAYA
Juli 2020

Halaman ini sengaja dikosongkan

SISTEM PERENCANAAN RUTE PADA *DOMESTIC SERVICE ROBOT* UNTUK PENCARIAN OBJEK RUMAH TANGGA.

Muhammad Sholahuddin Al-Ayubi
07111640000078

Dosen Pembimbing : 1. Dr. Ir. Hendra Kusuma, M.Eng.Sc.
: 2. Muhammad Attamimi, B.Eng, M.Eng., Ph.D

ABSTRAK

Domestic Service Robot merupakan robot otonom yang tugas utamanya melakukan pekerjaan rumah tangga. Dalam melakukan tugasnya, salah satu kemampuan penting yang perlu dimiliki oleh *Domestic Service Robot* yaitu adalah kemampuannya dalam menentukan rute yang dituju. Untuk dapat mencapai suatu posisi yang dituju, *Domestic Service Robot* membutuhkan suatu sistem perencanaan rute yang dapat menentukan rute yang efisien, aman dan bebas benturan.

Perencanaan rute yang dilakukan merupakan perencanaan rute gerak robot dari posisi robot berada menuju posisi target tujuan. Target tujuan yang digunakan dalam penelitian ini merupakan objek rumah tangga berdasarkan dari *dataset* yang telah diambil sebelumnya. Objek rumah tangga yang menjadi target tujuan robot akan dideteksi dan diidentifikasi menggunakan jaringan saraf konvolusi dan posisi relatif antara robot dan objek target dapat diperoleh dari pengolahan citra *depth* yang diambil dari kamera RGB-D. Penelitian ini menggunakan kamera RGB-D karena kemampuannya untuk memperoleh informasi jarak dari suatu lingkungan dengan mudah. Sistem ini akan diuji di ruangan *indoor* dan diterapkan pada robot beroda *omnidirectional* dalam merencanakan rute dan medekati objek rumah tangga.

Hasil dari penelitian ini merupakan grafik perbandingan hasil yang diperoleh dari program dan pada sistem yang sesungguhnya dengan RMS sebesar 80% dan waktu rata-rata respon sistem terhadap algoritma selama 12,15 detik.

Kata Kunci : *Path Planning, Object Search, Object Detection*.

Halaman ini sengaja dikosongkan

PATH PLANNING SYSTEM IN DOMESTIC SERVICE ROBOT FOR HOUSEHOLD OBJECT SEARCH.

Muhammad Sholahuddin Al-Ayubi
07111640000078

Supervisors : 1. Dr. Ir. Hendra Kusuma, M.Eng.Sc.
: 2. Muhammad Attamimi, B.Eng., M.Eng., Ph.D.

ABSTRACT

Domestic Service Robot is an autonomous robot whose main task is to do household chores. In carrying out their duties, one of the important capabilities that Domestic Service Robot needs to have is its ability to determine the intended route. To be able to reach a targeted position, the Domestic Service Robot requires a path planning system that can determine routes that are efficient, safe, and free from collisions.

Path planning is a robot motion path planning from the position of the robot to the target destination position. The target objectives used in this thesis are household objects based on the dataset that was taken previously. Household objects that are targeted by the robot will be detected and identified using a convolution neural network and the relative position between the robot and the target object can be obtained from processing depth images taken from an RGB-D camera. This study uses an RGB-D camera because of its ability to easily obtain distance information from an environment. This system will be tested in an indoor room and applied to omnidirectional wheeled robots in planning routes and finding household objects.

As a result of this research are graphs of the results obtained from the program and the system received with an RMS of 80% and the average response time of the system to the algorithm for 12.15 seconds.

Keywords : *Path Planning, Object Search, Object Detection.*

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji Penulis mengucapkan Syukur atas berkat Rahmat Tuhan Yang Maha Esa sehingga penulis mampu untuk menyelesaikan penelitian tugas akhir ini yang berjudul Sistem Perencanaan Rute pada Domestic Service Robot untuk Pencarian Objek Rumah Tangga.

Penelitian dan penulisan Tugas Akhir ini merupakan persyaratan untuk dapat menyelesaikan pendidikan program Strata-Satu di Departemen Teknik Elektro, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember Surabaya.

Tugas Akhir ini didasarkan pada teori dan praktik yang telah didapat melalui perkuliahan, pengalaman penulis sebagai salah satu anggota tim robot, serta berbagai literatur penunjang lainnya.

Atas banyaknya pihak-pihak yang telah membantu penulis untuk menyelesaikan penelitian ini, penulis ingin mengucapkan banyak terima kasih khususnya kepada:

1. Bapak dan Ibu penulis yang selalu menemani penulis dalam pengerjaan Tugas Akhir ini.
2. Dr. Ir. Hendra Kusuma, M.Eng.Sc. Sebagai dosen pembimbing 1 atas segala bimbingan, arahan dan petunjuk yang diberikan dalam proses pengerjaan Tugas Akhir ini.
3. Muhammad Attamimi, B.Eng., M.Eng., Ph.D. Sebagai dosen pembimbing 2 atas segala bimbingan, arahan dan petunjuk yang diberikan dalam proses pengerjaan Tugas Akhir ini.
4. Dimas Anton Asfani S.T., M.T.,Ph.D. selaku Sekretaris Departemen 1 Teknik Elektro ITS Surabaya.
5. Seluruh dosen bidang studi elektronika.
6. Teman - teman Laboratorium B-202 yang selalu mendukung dalam pengerjaan Tugas Akhir ini.
7. Seluruh rekan serta dosen pembimbing Tim Robotika ITS yang telah memberikan tempat untuk mengembangkan diri selama menjalani masa perkuliahan.
8. Team ABU Robocon ITS yang telah banyak memberikan pengalaman untuk mengembangkan diri penulis dalam bidang robotika.
9. Semua pihak yang telah membantu baik secara langsung maupun tidak langsung dalam proses penyelesaian Tugas Akhir ini.

Penulis sangat menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Untuk itu besar harapan penulis atas kritik dan saran yang membangun. Semoga hasil dari penelitian ini dapat terus dikembangkan.

.

Surabaya, 31 Mei 2020

Penulis

Halaman ini sengaja dikosongkan

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

DAFTAR TABEL

Halaman ini sengaja dikosongkan

BAB I
PENDAHULUAN

Pada Bab I dijelaskan mengenai latar belakang dari penelitian yang telah dilakukan. Berawal dari permasalahan sistem yang terjadi, peneliti berkeinginan untuk mengembangkan sistem melalui algoritma-algoritma yang ada untuk dilakukan pengujian dari beberapa percobaan. Pada penelitian ini diawali dengan melakukan studi literatur, perancangan sistem, pengujian dan analisis. Dalam Bab ini akan dijelaskan mengenai latar belakang, perumusan masalah, batasan masalah, tujuan, sistematika, serta metodologi yang digunakan dalam penelitian.

1.1 Latar Belakang

Terdapat banyak tantangan yang ditemui *Domestic Service Robot* dalam melakukan tugasnya. Salah satu tantangan yang mungkin ditemui adalah bagaimana merencanakan gerak robot dari satu titik ke titik lain. Permasalahan tersebut diharapkan dapat diterapkan di dunia nyata dimana *Domestic Service Robot* dijalankan di sebuah lingkungan *indoor* untuk mendekati objek rumah tangga. Selain menggunakan perencanaan rute, kemampuan yang harus dimiliki oleh *Domestic Service Robot* adalah pencarian objek. Bidang yang dikuasai adalah bidang penelitian aktif di bidang robotika dan visual komputer, namun realisasinya jarang dieksplorasi dalam sistem robot yang memungkinkan robot tidak hanya untuk mendeteksi objek tetapi juga untuk mendekati objek dengan menghindari rintangan.

Dalam tugas akhir ini penulis mencoba mengembangkan sistem perencanaan rute pada *Domestic Service Robot* yang beroperasi pada lingkungan *Indoor* untuk mendekati objek rumah tangga. Awalnya, robot diberi perintah objek dan estimasi lokasinya yang sudah ditentukan. Lalu perencanaan rute membuat rute yang harus dilalui robot untuk sampai ke target lokasi dan mulai mendekati objek. Jika objek dideteksi, robot diperintah jalan mendekat posisi target. Jika target yang dideteksi hilang selama mendekati proses, proses pencarian berulang sampai robot mencapai tujuan target atau kriteria pemutusan terpenuhi.

Dalam tugas akhir ini, robot perlu menghindari rintangan dan mendeteksi target di lingkungan *indoor*. Saat ini, kamera RGB-D sudah banyak tersedia dengan harga terjangkau, di mana informasi citra RGB dan citra kedalaman ditangkap secara serempak. Informasi yang kedalaman diperoleh dengan menggunakan sensor *Time of* Flight (TOF)

yang terdapat dalam kamera RGB-D. Sensor jenis ini muncul sebagai sensor alternatif yang menarik dan telah menarik perhatian peneliti di bidang visi komputer. Karena biaya rendah dan naiknya minat pada kamera RGB-D merupakan salah satu latar belakang yang mendukung tugas akhir ini. Tugas akhir ini akan menggunakan Kamera RGB-D, dan informasi citra RGB dan kedalamannya digunakan untuk mendeteksi objek dan mengestimasi posisi relatif robot terhadap objek.

1.2 Perumusan Masalah

Permasalahan yang dibahas dalam tugas akhir ini adalah :

1. Mengaplikasikan perencanaan rute dan kontrol gerak untuk mengendalikan pergerakan dari *Domestic Service Robot*.
2. Perancangan perencanaan rute yang menghasilkan keputusan berupa rute terbaik dari titik posisi robot menuju titik tujuan target.
3. Mengaplikasikan visi komputer dan pembelajaran mesin pada *Domestic Service Robot* untuk mengetahui posisi objek dan mengidentifikasi objek.
4. Mengintegrasikan perencanaan rute dan visi komputer pada *Domestic Service Robot* sehingga dapat mendekati objek.

1.3 Batasan Masalah

Dalam penelitian ini permasalahan yang dibahas memiliki batasan masalah sebagai berikut :

1. Objek yang dicari merupakan benda-benda rumah tangga berdasarkan dataset yang telah diambil sebelumya.
2. Sistem sudah mendapatkan informasi posisi robot dan halangan
3. Sistem perencanaa rute yang dibuat pada lingkungan area dua dimensi dengan halangan statis.
4. Robot berjalan pada permukaan yang rata.

1.4 Tujuan Penelitian

Tujuan yang ingin dicapai dalam tugas akhir ini adalah :

1. Mampu mengaplikasikan perencanaan rute dan kontrol gerak pada *Domestic Service Robot* untuk mengendalikan pergerakan dari *Domestic Service Robot*.
2. Sistem dapat membuat perencanaan rute yang dapat menghindari rintangan
3. Sistem dapat membuat rute terbaik

4. Mampu mengaplikasikan visi komputer dan pembelajaran mesin pada *Domestic Service Robot* untuk mengetahui posisi objek dan mengidentifikasi objek.

5. Mampu mengintegrasikan perencanaan rute dan visi komputer pada *Domestic Service Robot* sehingga dapat mendekati objek.

1.5 Metodologi

Pada penelitian ini dilakukan melalui beberapa tahap yaitu sebagai berikut :

1. Studi Literatur

Pada tahap ini dilakukan pengumpulan dan mengkaji dasar teori yang menunjang dalam pembuatan Tugas Akhir ini. Sumber literasi ini dapat diambil dari buku, jurnal, *paper*, *datasheet* yang sudah distandarisasi nasional ataupun internaisonal, artikel-artikel di internet dan forum-forum diskusi internet.

2. Perancangan *Software*

Perancangan *software* dilakukan dengan pembuatan *source code* yang meliputi pembuatan platform robot yang dapat menjalankan fungsi-fungsi pada sistem yang sebenarnya, penentuan titik awal dimana robot berada, titik target dimana objek yang dicari berada, arah orientasi robot menghadap ke objek dan perintah untuk membuat perencanaan rute gerak robot secara otomatis. Titik target ditentukan dengan menggunakan algoritma perhitungan untuk mendapatkan titik koordinat robot terhadap lingkungan dengan memanfaatkan parameter titik koordinat dimana robot berada, orientasi robot dan jarak objek target terhadap robot. Kemudian, *source code* yang dibuat akan mengirimkan perintah untuk membuat perencanaan rute yang akan dilalui robot menuju titik target.

3. Perancangan *Hardware*

Perancangan *hardware*, meliputi pembuatan *platform* robot. Robot yang dibuat merupakan robot *holonomic* dengan tiga roda *omni-directional* dengan *depth camera* sebagai sensor yang akan mendeteksi objek rumah tangga, *rotary-encoder* sebagai sensor posisi dan sensor *ultrasonic* sebagai sensor jarak. Kamera dipasang pada bagian kepala robot, *rotary-encoder* pada bagian bawah robot dan dihubungkan ke roda dan sensor *ultrasonic* dipasang mengelilingi badan robot. Kamera, *rotary-encoder* dan *ultrasonic* tersebut kemudian dihubungkan ke unit pemrosesan yang kemudian diolah agar dapat mengendalikan robot secara otomatis.

4. Pengujian Sistem

Pengujian alat dilakukan untuk menguji keandalan dari sistem yang telah dirancang. Pengujian dilakukan untuk melihat apakah *software* dan *hardware* yang telah dirancang dapat berfungsi secara optimal. Pengujian dapat dilakukan dengan beberapa tahap. Pertama adalah pengujian fungsionalitas untuk menguji apakah setiap sensor dari robot bekerja sesuai ekspektasi atau tidak. Kedua adalah pengujian implementasi algoritma yang telah dibuat ke dalam gerak robot yang sebenarnya.

5. Analisa

Analisa dilakukan terhadap hasil dari pengujian sehingga dapat ditentukan karakteristik dari *software* dan *hardware* yang telah dibuat. Apabila karakteristik dari algoritma yang telah diprogram dan diimplementasikan masih belum sesuai, maka perlu dilakukan perancangan ulang pada sistem dan kemudian diuji kembali.

6. Penyusunan Laporan Tugas Akhir

Tahap penulisan laporan tugas akhir adalah tahapan terakhir dari proses pengerjaan tugas akhir ini. Laporan tugas akhir berisi seluruh hal yang berkaitan dengan tugas akhir yang telah dikerjakan yaitu meliputi pendahuluan, tinjauan pustaka dan teori penunjang, perancangan sistem, pengujian, dan penutup.

1.6 Sistematika Penulisan

Pada penelitian ini disusun sistematika penulisan sesuai dengan penjelasan sebagai berikut :

BAB 1 PENDAHULUAN

Dalam pendahuluan akan dijelaskan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi, sistematika penulisan, serta relevansi dari penelitian.

BAB 2 TEORI PENUNJANG

Dalam teori penunjang dijelaskan mengenai teori dasar dari penelitian yang dilakukan. Teori dasar yang mendasari konsep dari perencanaan rute, deteksi objek serta *library* pendukung yang digunakan untuk proses kalibrasi serta pengolahan data.

BAB 3 PERANCANGAN SISTEM

Perancangan sistem mencakup penjelasan dari sistem yang dirancang. Penjelasan ini dapat berupa diagram blok, *flowchart*, hingga kode pendukung.

BAB 4 PENGUJIAN DAN ANALISIS

Pada Pengujian dan Analisis akan dilakukan percobaan pada sistem yang dirancang dalam menyelesaikan beberapa masalah yang disebutkan pada bagian rumusan masalah.

BAB 5 KESIMPULAN

Hasil dari percobaan yang dilakukan akan diterangkan dalam beberapa poin yang merepresentasikan semua percobaan yang telah dilakukan.

BAB II
TINJAUAN PUSTAKA

Dalam Bab ini dijelaskan mengenai teori-teori dasar untuk membangun sistem ini dan pernah diimplementasikan oleh penulis sebelumnya. Perencanaan rute, detektor objek serta teori-teori pendukung lainnya dijelaskan juga dengan tujuan untuk memberikan gambaran umum mengenai tugas akhir ini. Dasar teori ini akan digunakan sebagai dasar untuk perancangan sistem pada bab III.

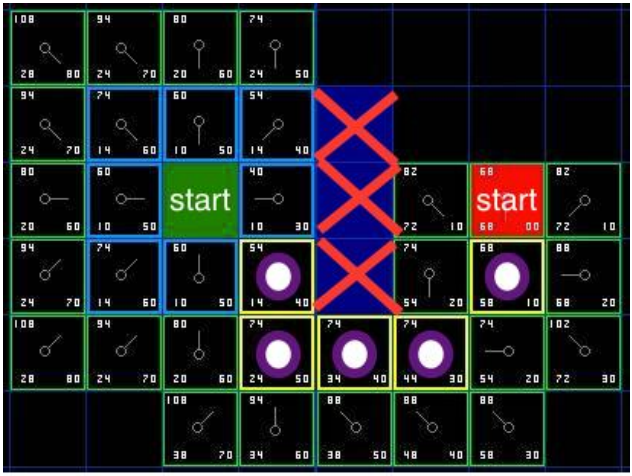
2.1 Perencanaan Rute

Perencanaan rute gerak robot adalah penentuan rute yang harus dilalui oleh robot untuk melewati setiap titik pada luasan area tertentu menuju ke titik target yang diinginkan[1]. Umumnya, perencanaan rute gerak robot digunakan untuk memecahkan permasalahan ketika robot berada dalam *indoor*. Algoritma perencanaan rute dirancang untuk memberikan keputusan rute yang harus dilalui robot menuju titik target yang diinginkan dengan mempertimbangkan posisi *obstacle* sehingga dapat menghindarinya.

Metode perencanaan rute yang telah dikembangkan sebelumnya dikelompokkan menjadi dua jenis yaitu pencarian buta atau tana informasi (*blind/uninformed search*) dan pencarian heuristic atau dengan informasi (*heuristic/informed search*). Metode pencarian heuristik merupakan metode pencarian dengan mengolah informasi yang telah didapatkan sebelumnya. Metode ini menggunakan fungsi perhitungan untuk menghitung biaya perkiraan dari suatu titik simpul menuju ke titik simpul tujuan yang dikenal sebagai fungsi heuristik (*heuristic*).

2.1.1 Algoritma A* (A-Star)

Algoritma A* merupakan algoritma pencarian jalur dari satu titik awal yang menuju titik akhir yang telah ditentukan dalam sebuah area bebas beraturan. Algoritma A* menggunakan metode pencarian heuristik *h(x)* yang memberikan nilai peringkat pada tiap titik koordinat dengan cara memperkirakan rute terbaik untuk dapat dilalui. Algoritma A* merupakan implementasi dari salah satu contoh dari metode *best-first search.* Contoh dari algoritma A* dapat dilihat pada gambar 2.1. Perhitungan biaya dapat dihitung dari penjumlahan biaya sebenarnya dan biaya perkiraan, sehingga dapat dirumuskan sebagai berikut:



Gambar 2.1 Contoh Algoritma A*(Star)

$$\boldsymbol{f}\,(\mathbf{n}) \;=\; \mathbf{g}\,(\mathbf{n}) \;+\; \mathbf{h}\,(\mathbf{n}) \qquad (2.1)$$

Algoritma A* merupakan algortima yang *optimal*. Hal ini dapat dibuktikan dengan penjelasan berikut. Dimisalkan titik *G* merupakan titik target dengan harga jalur *f** dan *G2* merupakan suboptimal *goal state*, yang berarti goal state dengan harga *g(G2) > f**. Keadaan ini dapat diimajinasikan bahwa algortima A* memilih *G2* dari antrian. Karena *G2* merupakan goal state, maka akan mengakhiri pencarian dengan solusi suboptimal. Hal ini tidak mungkin terjadi karena *node* n adalah *node* pada jalur optimal menuju *G*. Maka, harus ada beberapa *node* lagi, kecuali jalur telah selesai diperluas sepenuhnya dan algoritma mengembalikan nilai *G*. Selanjutnya, karena fungsi *h* diterima, maka *r >= f(n)*. Selain itu, jika *n* tidak dipilih untuk mengekspansi *G2*, maka *f(n) >= f(G2)*. Sehingga *r > f(G2)*. Tetapi karena *G2* merupakan *goal state*, maka *h(G2) = 0*. Sehingga *f(G2) = g(G2)*. Oleh karena itu, dapat diasumsikan bahwa *f* >= g(G2)*. Hal ini bertentangan dengan asumsi bahwa *G2* adalah suboptimal, sehingga dapat diketahui bahwa algoritma A* tidak memilih tujuan suboptimal untuk ekspansi. Oleh karena itu, Algoritma A* merupakan

algoritma yang *optimal* karna hanya akan mengembalikan nilai solusi setelah memilih *goal state* untuk ekspansi.

Algoritma A* juga merupakan algoritma yang *complete* yang artinya algoritma A* mengekspansi *node* untuk menaikan fungsi *f* untuk mencapai goal state. Hal ini merupakan hal yang tepat, kecuali tak terhingganya node dengan $f(n) < f^*$. Satu-satunya cara jika terdapat *node* yang tak terbatas adalah dengan cara membuat factor percabangan yang tak terbatas atau dengan adanya jalur dengan harga yang terbatas namun dengan jumlah *node* yang tak terbatas pada sepanjang jalur tersebut. Dengan demikian algoritma A* merupakan algoritma yang *complete* pada grafik dengan factor percabangan terbatas [2].

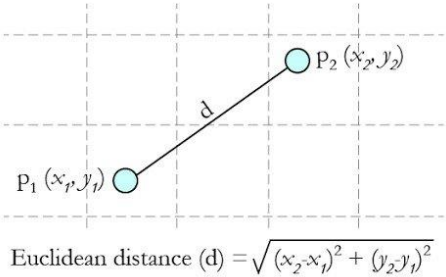
Prinsip algoritma A* adalah memperluas setiap simpul yang mungkin dari awal ke tujuan dan membandingkan biaya masing-masing jalur. Begitu ada hambatan yang membuat jalur saat ini lebih mahal daripada jalur lain yang tersedia, A* akan kembali ke jalur terendah yang baru dan mengembangnya lagi. Setelah mengulangi proses pencarian ini, A* akhirnya akan menemukan jalan setapak yang harganya paling murah sehingga mengembang ke tujuan. Seperti ditunjukkan pada gambar 2.1, algoritma A* mulai memperluas node dari A ke tujuan B, dan *grid* dengan tanda silang adalah hambatannya. Setiap grid yang diperluas berisi informasi: F di kiri atas, G di kiri bawah, H di kanan bawah, dan panah yang menunjuk induknya. Dalam contoh ini, fungsi heuristik adalah Euclidean.

2.1.2 Fungsi Heuristik Jarak Euclidian

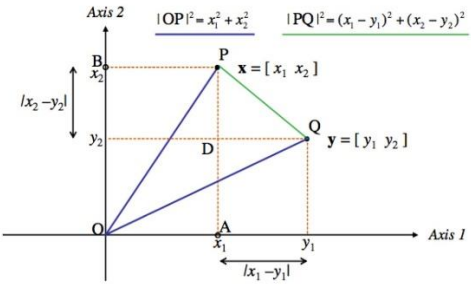
Fungsi heuristik memiliki peranan penting salam metode pencarian yang termasuk kedalam pencarian heuristik. Fungsi heuristik tersebut hanya dapat diterima apabila fungsi tersebut menghasilkan estimasi harga yang tidak melebihi harga sebenarnya. Karena ketika suatu fungsi heuristik menghasilkan estimasi harga yang melebihi dari harga sebenarnya atau biasa disebut *overestimate*, maka proses pencarian bisa tersesat dan membuat pencarian heuristik menjadi tidak optimal. Fungsi heuristik bisa dikatakan baik jika fungsi tersebut dapat menghasilkan estimasi harga yang mendekati harga sebenarnya [2]. Salah satu fungsi heuristik yang dapat digunakan adalah fungsi heuristic jarak *euclidean* seperti pada **gambar 2.2**.

Fungsi heuristik jarak *euclidean* merupakan salah satu metode perhitungan jarak antara dua titik dalam sebuah area yang biasanya

disebut *euclidean space*. Fungsi jarak *euclidean* berkaitan dengan prinsip teorema *pythagoras*, yaitu hasil dari kuadrat sisi miring didapatkan dari penjumlahan kuadrat antara sisi – sisi lainnya. Penjelasan tentang cara memperoleh jarak *euclidean* dapat dibuktikan dengan mengaplikasikan teorema *pythagoras* pada luasan dua dimensi seperti pada **gambar 2.3**.



Gambar 2.2 Jarak Euclidian



Gambar 2.3 Teorema Phytagoras dalam area dua dimensi

2.2 *Domestic Service Robot*

Domestic Service Robot adalah salah satu tipe *service robot* yang memiliki karakteristik berbeda dari robot statik seperti lengan industri,

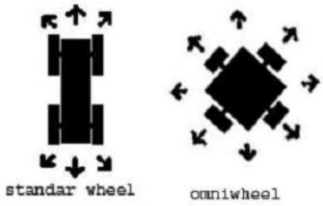
dimana *Domestic Service Robot* memiliki fungsi untuk melakukan pekerjaan rumah tangga yang biasanya dikerjakan manusia dan memiliki karakteristik bergerak bebas dari satu tempat ke tempat lain[3]. Jenis-jenisnya pun bermacam-macam sesuai dengan caranya bergerak. *Domestic Service Robot* dapat berbentuk robot beroda, ataupun jenis robot berkaki. Hal ini disesuaikan dengan kebutuhan penggunaan robot, dengan tugas yang dilakukan, lingkungan sekitar, dan fungsional dari robot sendiri. Hal yang membedakan dari *Domestic Service Robot* darat dengan *Robot* lain (UAV, USV) adalah keterbatasan *degree of freedom* pada sistem navigasinya. Untuk *Mobile Robot* beroda yang digunakan hanya di dalam ruangan, dengan permukaan yang relatif rata, hanya perlu digunakan 3 DOF (*Degrees of Freedom*) navigasi yaitu X, Y, dan Sudut *Heading*. Hal ini yang akan menjadi sebuah parameter pengujian sistem pencari objek yang penulis rancang.

2.2.1 Robot *Omnidirectional*

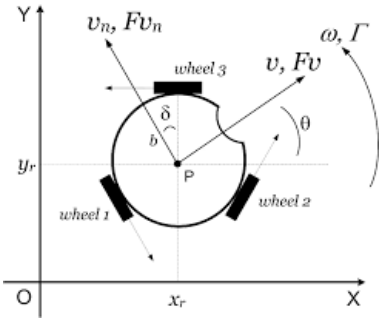
Robot *omnidirectional* adalah robot dengan sistem pergerakan yang secara langsung dapat bergerak ke segala arah dengan konfigurasi apapun. Pada umumnya robot di desain dengan pergerakan yang sudah direncanakan terlebih dahulu. Pada sistem pergerakan konvensional, pergerakan tidak mampu di kontrol pada setiap tingkat kebebasan dalam bergerak secara independen, sehingga hanya mampu bergerak ke beberapa arah yang sudah ditentukan sebelumnya. Perbandingan dari sistem gerak roda biasa dan omni digambarkan pada **gambar 2.4**. Ini disebut kendala *non-holomic* yaitu pencegahan roda kemudi dari selip, meskipun pada umumnya mampu menjangkau setiap lokasi dan orientasi dalam ruang 2 dimensi, namun memerlukan manuver dan perencanaan jalan yang rumit dan kompleks. Pada tugas akhir ini, Robot memiliki 3 DOF terhadap bidang gerak robot menggunakan roda *omni directional* yang menghadap jauh pada pusat robot seperti pada **gambar 2.5**.

Kinematika robot *holonomic* dapat dihitung berdasarkan susunan struktur mekanis dengan menggunakan pengembangan metode kinematika *forward* dan transformasi antara koordinat bidang dan koordinat robot [4]. Implementasi dari kinematika *forward* pergerakan robot dapat dituliskan dengan persamaan berikut:

$$\begin{bmatrix} \dot{\emptyset}_1 \\ \dot{\emptyset}_2 \\ \dot{\emptyset}_3 \end{bmatrix} = \begin{bmatrix} -\sin(\theta) & \cos(\theta) & R \\ -\sin(\theta+\alpha_2) & \cos(\theta+\alpha_2) & R \\ -\sin(\theta+\alpha_3) & \cos(\theta+\alpha_3) & R \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{2.2}$$



Gambar 2.4 Sistem Pergerakan Roda Standard dan Omnidirectional

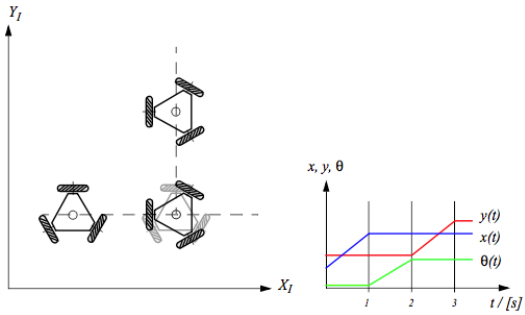


Gambar 2.5 Robot *Holonomic*

2.2.2 Trayektori dan Penentuan Rute

Dalam bidang robotika, kemampuan robot untuk mengikuti rute yang telah ditentukan untuk mencapai titik target perlu diperhitungkan. Robot harus mampu melacak rute yang harus dilalui dalam ruang geraknya [5]. Pada robot *omnidirectonal* ada tiga parameter yang perlu diperhatikan untuk menentukan arah pergerakan robot, yaitu kecepatan terhadap sumbu kartesian *x*, kecepatan terhadap sumbu kartesian *y*, dan kecepatan perubahan arah orientasi robot yang disebut kecepatan sudut θ.

Contoh lintasan robot omnidirectional dan grafik pergerakannya dapat dilihat pada **gambar 2.6.**. Dimana robot bergerak selama 1 detik dengan kecepatan konstan 1 m/s di sepanjang sumbu kartesian x. Kemudian robot melakukan perubahan orientasi berlawanan arah jarum jam sebesar 90 derajat dalam 1 detik. Kemudian robot bergerak selama 1 detik dengan kecepatan konstan 1 m/s di sepanjang sumbu kartesian y.



Gambar 2.6 Contoh lintasan pergerakan robot holonomic

2.3 Pengolahan Citra Digital

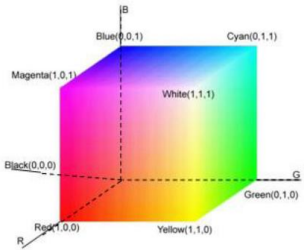
Pengolahan citra digital dapat didefinisikan sebagai pemrosesan citra dengan menggunakan komputer digital. Citra digital tersusun atas elemen dengan jumlah terbatas dan masing-masing elemen tersebut menempati koordinat tertentu dan memiliki amplitudo/nilai. Elemen-elemen tersebut kemudian disebut dengan elemen citra atau piksel. Pengolahan citra digital bertujuan untuk memperoleh informasi atau deskripsi yang terdapat pada objek [6]. Berikut hal dasar yang perlu diketahui dalam pengolahan citra digital :

2.3.1 Citra dengan Ruang Warna RGB

Ruang warna RGB adalah ruang warna yang paling umum digunakan pada komputer dan kamera digital. Ruang warna ini didasarkan pada campuran dari tiga warna primer Merah (*Red*), Hijau (*Green*) dan Biru (*Blue*) yang dinyatakan secara fisik [6].

Ruang warna RGB dapat dimodelkan dalam bentuk kubus tiga dimensi. Semua warna yang bisa didapatkan dari kombinasi warna R, G, dan B telah direpresentasikan pada model kubus RGB seperti pada **gambar 2.7**. Warna hitam dalam RGB direpresentasikan dalam nilai (0,

0, 0) untuk nilai R, G, dan B secara urut. Warna putih direpresentasikan dalam nilai maksimal yaitu (255, 255, 255) untuk nilai R, G, dan B secara urut, apabila menggunakan resolusi 8 bit. Namun hasil kombinasi ini tidak disukai karena tidak ideal dengan warna aslinya. Hal tersebut dikarenakan warna merah, hijau, dan biru sesungguhnya saling mempengaruhi untuk membuat satu warna tertentu. Sehingga penelitian terkait gambar digital mengusulkan ruang warna HSV yang merupakan representasi dari *Hue*, *Saturation*, dan *Value* [6].

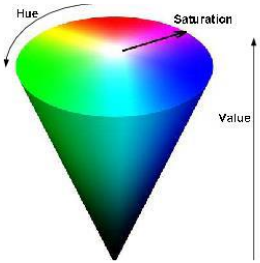


Gambar 2. 7 Model Kubus Ruang Warna RGB

2.3.2 Citra dengan Ruang Warna HSV

Citra Hasil kombinasi warna pada ruang warna RGB memiliki sebuah masalah, yaitu warna yang terbentuk dari hasil kombinasi tidak sepenuhnya merepresentasikan warna aslinya, untuk mengatasi hal itu maka diterapkanlah transformasi non-linier pada ruang warna RGB, salah satu hasilnya adalah ruang warna HSV. Ruang warna HSV merupakan ruang warna yang merepresentasikan warna seperti yang dilihat oleh mata manusia. Ruang warna HSV dinyatakan oleh tiga elemen yaitu, *hue*, *saturation*, dan *value*. Ruang warna HSV direpresentasikan dalam sebuah model kerucut terbalik. *Hue* merupakan tipe warna, seperti: merah, kuning [6]. Tipe warna ini ditentukan oleh nilai dari 0 sampai 360 derajat pada kerucut warna HSV seperti pada **gambar 2.8.** *Saturation* merupakan kemurnian warna, dimana dalam kerucut HSV ditentukan oleh nilai dari 0 sampai 100% [6]. *Value* merupakan tingkat kecerahan, dimana dalam kerucut HSV ditentukan oleh nilai dari 0 sampai 100% [6].

Ruang warna ini termasuk dalam sistem warna non-linier. Metode representasi warna HSV sesuai dengan persepsi manusia tentang warna. Setiap elemen ruang warna terpisah, sehingga cocok untuk pengolahan citra digital [6].



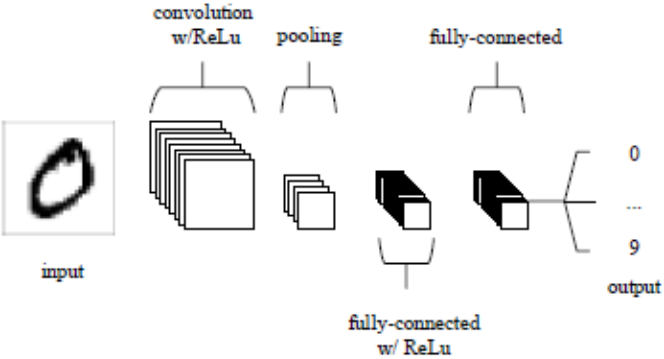
Gambar 2. 8 Model Ruang Warna HSV

2.4 *Convolutional Neural Network*

Convolutional neural network (CNN) jenis jaringan *feed-forward neural network* khusus yang mencakup lapisan *convolutional* dan *pooling* dalam arsitekturnya. Pola umum untuk arsitektur CNN adalah memiliki *input layer*, beberapa kombinasi *convolutional layer* dan *pooling layer*, dan *fully-connected layer* yang terhubung dengan *output layer* [7].

Konvolusi adalah konsep utama dibalik arsitektur CNN. Secara sederhana, konvolusi adalah operasi matematika yang menggabungkan informasi dari dua sumber untuk menghasilkan satu set informasi baru [7]. Secara khusus, ia menerapkan matriks khusus yang dikenal sebagai *kernel* ke *input*, untuk menghasilkan satu set matriks yang dikenal sebagai *feature maps*.

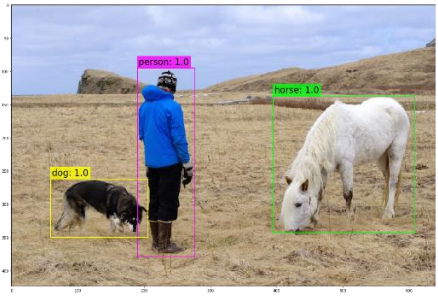
Pooling mengacu pada penghitungan statistik agregat atas wilayah dari *feature* yang dikonvolusi [7]. Dua jenis statistik agregat paling populer adalah maksimum dan rata-rata. Output dari penerapan *max-pooling* adalah maksimum wilayah yang dipilih, sedangkan output dari penerapan *average-pooling* adalah rata-rata dari angka-angka di wilayah tersebut. Pada **gambar 2.9** menunjukan ilustrasi dari arsitektur sederhana CNN.



Gambar 2.9 Arsitektur sederhana CNN

2.5 *Object Detection*

. *Object Detection* (deteksi objek) adalah metode untuk menemukan objek dalam sebuah citra dan menemukan posisinya yang direpresentasikan oleh *bounding box* (kotak pembatas) [8]. Contoh *object detection* dapat dilihat pada **gambar 2.10**. Metode ini banyak sekali digunakan dalam aplikasi visi komputer, seperti untuk mendeteksi pejalan kaki, mendeteksi mobil, mendeteksi lampu lalu lintas, dan lain-lain.



Gambar 2.10 Contoh Deteksi Objek

2.7 Odometry

Odometry merupakan salah satu metode untuk menentukan posisi pada robot. Metode *odometry* memanfaatkan nilai keluaran sistem sensor

tertentu untuk mengestimasi perpindahan robot. Pada robot yang digunakan pada tugas akhir ini, *odometry* yang digunakan untuk mengestimasi perpindahan robot memanfaatkan putaran *rotary encoder* dan orientasi kompas digital. Perpindahan translasi robot didapatkan dengan cara mengintegrasi kecepatan putaran *rotary encoder* dan orientasi kompas digital. Sedangkan perpindahan rotasi robot murni didapatkan dari orientasi kompas digital.

2.7.1 *Rotary Encoder*

Rotary encoder adalah sensor yang berfungsi untuk mengukur perpindahan rotasi suatu benda yang berputar pada porosnya. Cara menggunakan *rotary encoder* adalah dengan memasang kopling pada poros *rotary encoder* dengan poros benda berputar yang ingin diukur. *Rotary encoder* umumnya menggunakan sensor optik untuk menghasilkan serial pulsa yang dapat diartikan menjadi gerakan, posisi, dan arah. Sehingga posisi sudut suatu poros benda berputar dapat diolah menjadi informasi berupa kode digital oleh *rotary encoder* untuk diteruskan oleh rangkaian kendali.

Pada robot ini digunakan *rotary encoder* tipe *incremental*. *incremental encoder* terdiri dari dua *track* atau *single track* dan dua sensor yang disebut *channel* A dan B Ketika poros berputar, deretan pulsa akan muncul di masing-masing *channel* pada frekuensi yang proporsional dengan kecepatan putar sedangkan hubungan fasa antara *channel* A dan B menghasilkan arah putaran. Dengan menghitung jumlah pulsa yang terjadi terhadap resolusi piringan maka putaran dapat diukur. Untuk mengetahui arah putaran, dengan mengetahui *channel* mana yang leading terhadap *channel* satunya dapat kita tentukan arah putaran yang terjadi karena kedua *channel* tersebut akan selalu berbeda fasa seperempat putaran (*quadrature signal*). Bentuk fisik dari *encoder* yang digunakan yaitu *Autonics* E30S4 adalah seperti pada **Gambar 2.11**.

Gambar 2.11 Incremental Rotary Encoder Autonics E30S4

2.7.2 Kompas Digital

Dalam *odometry*, kompas digital berperan penting untuk menunjukkan orientasi robot terhadap lingkungannya. Karena setiap cuplikan kecepatan robot akan dihitung komponennya terhadap sumbu-x dan sumbu-y area berdasarkan orientasi robot. Kesalahan penghitungan komponen kecepatan akan berakibat pada output posisi bergeser (*drifting*) dari posisi yang sebenarnya.

Kompas digital terdiri dari sensor giroskop dan akselerometer mekanik yang berukuran sangat kecil. Sensor giroskop berfungsi menunjukkan kecepatan sudut benda pada porosnya sedangkan sensor giroskop berfungsi menunjukkan akselerasi benda pada sumbu tertentu. Ketelitian kompas digital ditentukan oleh resolusi keluaran dari sensor yang dapat diatur melalui konfigurasi awal ketika sensor dinyalakan. Bentuk fisik dan ilustrasi keluaran sensor dari kompas digital yang digunakan yaitu GY-52 adalah seperti pada **Gambar 2.12**.

Gambar 2.12 Kompas Digital GY-52

2.8 Mikrokontroler STM32F4 *Discovery*

STM32F4 *Discovery* adalah sebuah modul mikrokontroler yang di dalamnya menggunakan IC STM32F407VGT6 ARM Cortex-M4 yang mempunyai kecepatan sampai dengan 168 MHz, serta mampu mengeksekusi perintah hingga 210 MIPS (*Million Instruction per Second*). STM32F4 Discovery merupakan mikrokontroler 32 bit dengan arsitektur ARM. Mikrokontroler ini memiliki kapasitas 1 MByte Flash PEROM (*Flash Programmable and Eraseble Read Only Memory*), 192 Kbyte SRAM [9]. Dilengkapi dengan 100 buah pin *input output* yang mempunyai karakteristik masing-masing yaitu USART, TIMER, ADC dan I2C bentuk fisik dari mikrokontroler STM32F4 Discovery tersebut seperti pada **gambar 2.13**.

Gambar 2.13 Mikrokontroler STM32F4 *Discovery*

STM32F4 *Discovery* memiliki 14 *timer* dan 6 *Universal Asynchronous Receiver Transmitter* (UART) yang dapat digunakan secara independen. 6 dari 14 *timer* dapat digunakan untuk membaca *rotary encoder*, yaitu TIM1, TIM2, TIM3, TIM4, TIM5, dan TIM8. Sedangkan TIM6 dan TIM7 hanya dapat digunakan sebagai *counter* atau *timer* internal dikarenakan tidak mempunyai pin output fisik pada mikrokontroler. TIM9, TIM10, TIM11, TIM12, TIM13, dan TIM14 dapat digunakan sebagai pembangkit sinyal *Pulse Width Modulation* (PWM) untuk digunakan sebagai kecepatan motor pada robot. Mikrokontroler STM32F4 juga dilengkapi dengan *Direct Memory Access* (DMA) yang dapat dimanfaatkan untuk mengelola UART tanpa membebani komputasi utama sehingga diharapkan semua komputasi dapat berjalan secara *realtime*.

2.9 Komputer Mini

Komputer mini atau *single board computer* adalah kelas komputer multi pengguna yang memiliki bentuk yang lebih kecil daripada komputer personal pada umumnya. Spesifikasi atau kemampuan yang dimiliki oleh komputer mini lebih unggul dibandingkan dengan komputer personal pada umumnya. Hal ini disebabkan karena mikroprosesor yang digunakan komputer mini dalam pemrosesan data memiliki kemampuan jauh lebih mumpuni dibandingkan dengan mikroprosesor komputer personal biasa. Dalam pembuatan tugas akhir ini, komputer mini berfungsi sebagai *processing unit* utama untuk melakukan proses pengolahan data informasi yang diperoleh dari kamera, melakukan simulasi dan mengendalikan pergerakan robot. Komputer mini yang dipakai adalah CK3 Mini PC seperti pada **gambar 2.14** dengan spesifikasi Prosesor Intel® Core™ i7-4500U (6M Cache, hingga 3.50 GHz), RAM 8 GB DDR4, SSD 128GB, TDP 45 Watt, Suplai sistem 14 Volt, memiliki port HDMI, 6 buah port USB dan mendukung sistem 64-bit.

Gambar 2.14 CK3 Mini PC

2.10 Intel RealSense D435i

Kamera kedalaman D435i adalah bagian dari seri kamera Intel® RealSense ™ D400, jajaran yang membawa perangkat keras dan perangkat lunak pengindraan kedalaman terbaru Intel dan mengemasnya menjadi produk yang mudah diintegrasikan. Sempurna untuk pengembang, pembuat, dan inovator yang ingin menghadirkan penginderaan mendalam pada perangkat, kamera seri Intel® RealSense ™ D400 menawarkan integrasi sederhana dan memungkinkan generasi baru dari solusi cerdas yang dilengkapi dengan penglihatan cerdas. Menambahkan IMU memungkinkan aplikasi untuk meningkatkan kesadaran kedalamannya dalam situasi apa pun di mana kamera bergerak.

Perangkat ini juga memungkinkan peningkatan kemampuan lingkungan untuk robotika dan drone. Penggunaan IMU membuat registrasi dan kalibrasi lebih mudah untuk kasus penggunaan sistem pemindaian genggam dan juga penting dalam bidang-bidang seperti *virtual* atau *augmented reality* dan *drone*. Unit pengukuran inersia (IMU) digunakan untuk mendeteksi gerakan dan rotasi dalam 6 derajat kebebasan (6DoF). IMU menggabungkan berbagai sensor dengan giroskop untuk mendeteksi rotasi dan gerakan dalam 3 sumbu, serta pitch, yaw, dan roll [10].

Gambar 2.15 Intel RealSense D435i

2.11 Tensorflow

Tensorflow adalah pustaka perangkat lunak sumber terbuka yang dirilis pada tahun 2015 oleh Google untuk memudahkan merancang, membuat, dan melatih model *deep learning* [11]. Meskipun *Tensorflow* hanya satu dari beberapa opsi yang tersedia untuk para pengembang, *Tensorflow* memiliki fitur yang lengkap dan mudah untuk digunakan [11].

Tensorflow adalah pustaka Python yang memungkinkan pengguna untuk merepresentasikan perhitungan sebagai grafik aliran data. *Node* dalam grafik ini merepresentasikan operasi matematika, sedangkan *edge* mewakili data yang dikomunikasikan dari satu *node* ke *node* lainnya. Data dalam *Tensorflow* direpresentasikan sebagai tensor, yang merupakan larik multidimensi (mewakili vektor dengan tensor 1 dimensi, matriks dengan tensor 2 dimensi, dll.) [11]. *Tensorflow* juga menyertakan Tensorboard, alat tambahan untuk melakukan visualisasi data. Adapun logo dari *tensorflow* dapat dilihat pada **gambar 2.16**.

Gambar 2.16 Logo Tensorflow [11]

2.11.1 Tensorflow Object Detection API

Tensorflow menciptakan model pembelajaran mesin yang akurat yang mampu menentukan lokasi objek dan mengidentifikasi banyak

objek dalam satu gambar tetap, yang menjadi tantangan utama dalam visi komputer. *Tensorflow Object Detection API* adalah kerangka kerja sumber terbuka yang dibangun di atas *Tensorflow* yang membuatnya mudah untuk membangun, melatih, dan menggunakan model deteksi objek [11].

2.12 Google Colaboratory

Tensorflow Google Colaboratory atau *Google Colab* adalah layanan *cloud* gratis dan sekarang juga mendukung untuk pelatihan model menggunakan GPU gratis. *Google Colab* menggunakan *Jupyter Notebook* yang tidak lagi memerlukan pengaturan dan sepenuhnya berjalan di *cloud*. *Google Colab* dapat digunakan menulis dan mengeksekusi kode, menyimpan dan membagikan analisis, dan mengakses sumber daya komputasi yang kuat, semuanya cepat, mudah, dan gratis dari browser [12].

2.13 OpenCV

OpenCV (Open Source Computer Vision Library) adalah pustaka perangkat lunak sumber terbuka untuk aplikasi visi komputer dan pembelajaran mesin. *OpenCV* tersedia dalam berbagai macam bahasa pemrograman termasuk Python dan C++. Ada banyak fungsi pengolahan citra digital yang tersedia pada pustaka ini yang dapat sangat bermanfaat [13].

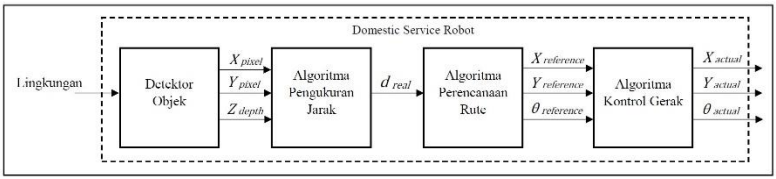
Pustaka ini memiliki lebih dari 2500 algoritma yang sudah dioptimalkan, yang mencakup visi computer dan pembelajaran mesin yang mutakhir [13]. Algoritma ini dapat digunakan untuk mendeteksi dan mengenali wajah, mengidentifikasi objek, mengklasifikasikan tindakan manusia dalam video, melacak pergerakan kamera, melacak objek bergerak, mengekstraksi model objek 3D, menyatukan citra bersama untuk menghasilkan citra resolusi tinggi, menemukan citra serupa dari basis data, menghapus mata merah dari citra yang diambil menggunakan flash, mengikuti gerakan mata, mengenali pemandangan, dll [13]. Logo *OpenCV* dapat dilihat seperti pada **gambar 2.17**.

Gambar 2.17 Logo OpenCV [16]

Halaman ini sengaja dikosongkan

BAB III
PERANCANGAN SISTEM

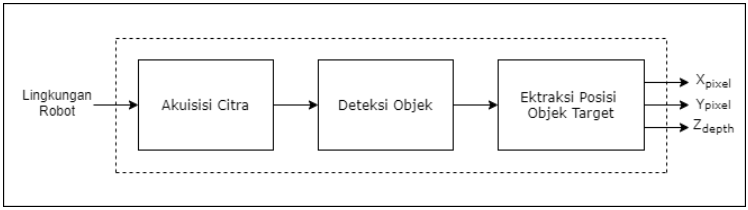
Bab ini akan dijelaskan mengenai sistem yang akan dibangun seperti perancangan hardware yang meliputi perancangan mekanik dan perancangan sistem elektronik. Selain itu juga dijelaskan mengenai perancangan *software* yang meliputi perancangan detektor objek, pengukuran jarak target, perancangan algoritma perencanaan rute, dan juga perancangan algoritma kontrol gerak.



Gambar 3.1 Diagram Blok Sistem

3.1 Detektor Objek

Detektor objek berfungsi untuk mendeteksi dan mengektraksi informasi posisi objek rumah tangga yang ingin dituju. Informasi yang didapatkan dari citra lingkungan agar dapat diolah pada unit pemrosesan. Pada sub sistem ini, terdapat beberapa proses yaitu akuisisi citra, deteksi dan ekstraksi posisi target. Proses-proses tersebut digambarkan dalam bentuk diagram blok seperti pada **gambar 3.2**. Keluaran dari sub sistem ini berupa koordinat posisi piksel dan jarak kedalaman target yang selanjutnya akan diolah pada sub sistem pengukuran jarak seperti pada diagram blok **gambar 3.6**.

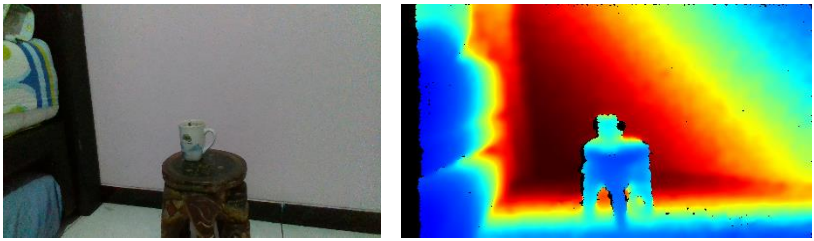


Gambar 3.2 Diagram Blok Detektor Objek

3.1.1 Akuisisi Citra

Dalam penelitian ini, kamera berfungsi sebagai pendeteksi posisi objek. Kamera yang digunakan adalah kamera RGB-D *Intel RealSense*. Pemasangan kamera pada sistem dapat dilihat pada **gambar 3.3**. Citra yang dihasilkan dari pemasangan kamera dengan desain tersebut ditunjukan pada **gambar 3.4**. Kamera RGB-D Intel RealSense sudah dilengkapi dengan API yang cukup mudah digunakan untuk berbagai keperluan, salah satunya yaitu pengambilan informasi citra warna dan citra kedalaman dalam satuan yang baku (citra kedalaman dalam satuan milimeter dan citra warna dalam satuan 8-bit RGB *channel*).

Gambar 3.3 Pemasangan kamera pada sistem



Gambar 3.4 Citra warna dan citra kedalaman yang dihasilkan oleh kamera

3.1.2 Deteksi Objek

Untuk mendeteksi dan mengekstraksi informasi posisi objek rumah tangga yang ingin dituju, dibutuhkan nilai piksel x, y yang diekstraksi dari citra warna dan nilai piksel kedalaman dari citra *depth*. Dalam penelitian

ini, deteksi objek rumah tangga memanfaatkan *Tensorflow Object Detection API* yang mampu menentukan lokasi piksel objek dalam citra warna. Objek rumah tangga yang digunakan dalam penelitian ini adalah gelas, mangkok, dan sepatu. Data objek rumah tangga yang diambil secara mandiri yang terdiri dari 3 kelas dengan masing-masing kelas diambil 300 citra dengan ukuran piksel gambar 640 x 480. Pada **Gambar 3.5** dapat dilihat objek rumah tangga yang digunakan untuk deteksi objek pada tugas akhir ini. Deteksi objek yang dihasilkan dari memanfaatkan API ini ditunjukkan pada **gambar 3.6**. Hasil dari langkah deteksi objek ini merupakan nilai piksel x,y dari titik tengah objek target yang dideteksi.

Gambar 3.5 Dataset Objek Rumah Tangga (a) Sepatu (b) Mug (c) Mangkok

Gambar 3.6 Deteksi Objek Target

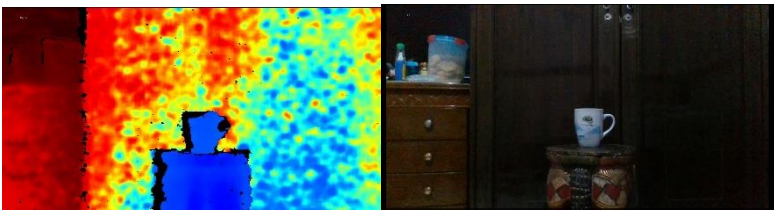
3.1.3 Pengekstraksian Posisi Objek target

Resolusi yang tidak sama antara citra warna dan citra kedalaman menyebabkan nilai piksel di lokasi yang sama pada citra merepresentasikan titik yang berbeda. Akan tetapi meskipun resolusi citra warna dan citra kedalaman sama, tidak berarti nilai piksel di lokasi yang sama pada kedua citra merepresentasikan titik yang sama. Perbedaan ini

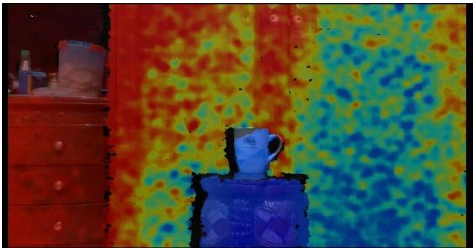
disebabkan perbedaan sudut pandang kedua citra yang berbeda, sehingga nilai dari kedua citra tidak dapat digabungkan begitu saja. Maka dari itu perlu dilakukan pengolahan untuk mencari lokasi piksel dari kedua citra yang merepresentasikan titik yang sama.

Align adalah sebuah istilah digunakan untuk membentuk gambar baru dimana nilai piksel yang sama merepresentasikan nilai titik yang sama. Untuk melakukan ini, digunakan fitur *align* pada API *librealsense*. *Align* yang dilakukan adalah menyamakan nilai kedalaman pada gambar warna sebagai acuan.

Pada **gambar 3.7** terlihat kedua citra kedalaman dan citra warna untuk objek yang sama namun tidak dilakukan *align*. Setelah dilakukan *align* maka akan terbentuk citra dengan lokasi piksel yang sama untuk objek titik yang sama pula seperti pada **gambar 3.8.** Dari hasil proses ini, kita bisa mendapatkan informasi nilai piksel x, y, dan kedalamannya.



Gambar 3.7 Citra kedalaman dan citra warna sebelum di-*align*



Gambar 3.8 Citra baru setelah di-*align*

3.1 Pengukuran Jarak

Pada proses sebelumnya, telah diketahui titik koordinat piksel objek target yang ingin dituju, langkah selanjutnya adalah mengkonversi koordinat tersebut menjadi posisi 3 dimensi relatif terhadap posisi robot

saat pengakuisisian citra. Posisi ini didapatkan dengan persamaan sebagai berikut:

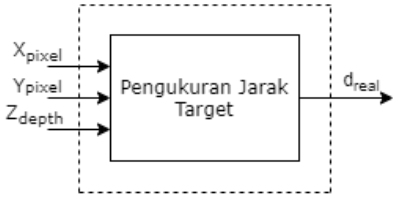
$$X_{object} = Z_{depth}(X_{pixel} - X_{pp})/fx \tag{3.1}$$

$$Y_{object} = Z_{depth}(Y_{pixel} - Y_{pp})/fy \tag{3.2}$$

Dengan X_{pixel} dan Y_{pixel} adalah koordinat piksel dari gambar yang ditangkap, *Xpp* dan *Ypp* adalah koordinat piksel *principal point* (titik tengah proyeksi), serta *fx* dan *fy* adalah *focal length* dari gambar yang diambil. Setelah mendapatkan posisi relatif objek target terhadap robot kita dapat menghitung jarak sesungguhnya objek terhadap robot dengan persamaan sebagai berikut:

$$d_{real} = \sqrt{(X_{robot} - X_{object})^2 + (Y_{robot} - Y_{object})^2} \tag{3.3}$$

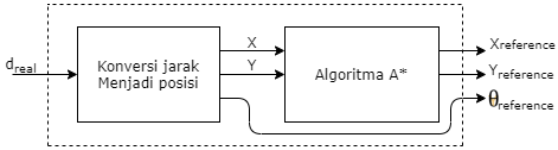
Proses-proses pada subsistem pengukuran jarak digambarkan dalam bentuk diagram blok seperti pada **gambar 3.9**.



Gambar 3.9 Diagram Blok Pengukuran Jarak

3.2 Perancangan Algoritma Perencanaan Rute

Algoritma *path planning* dirancang untuk memberikan keputusan rute yang harus dilalui robot menuju titik target yang diinginkan dengan mempertimbangkan posisi *obstacle* sehingga dapat menghindarinya. Proses-proses pada sub sistem ini digambarkan kedalam bentuk diagram blok seperti pada **gambar 3.10**. Dalam hal ini algoritma *path planning* yang digunakan adalah algoritma A* (A *Star*).



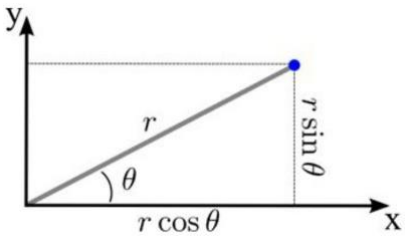
Gambar 3.10 Diagram Blok Perencanaan Rute

3.2.1 Perhitungan Konversi Jarak menjadi Koordinat

Perhitungan konversi jarak menjadi titik koordinat digunakan untuk mengkonversi jarak yang terukur dalam satuan meter ke dalam bentuk koordinat kartesian (x, y). Sedangkan nilai θ diperoleh dari pembacaan nilai keluaran dari sensor *gyro*. Perhitungan dapat dilakukan dengan menggunakan prinsip dari trigonometri yang digambarkan dengan grafik hubungan antara koordinat kartesian dengan koordinat kutub seperti pada **gambar 3.11**. Dari grafik gambar 3.7, dapat diperoleh perumusan untuk mencari koordinat x dan koordinat y sebagai berikut:

$$x = r\cos\theta \tag{3.4}$$

$$y = r\sin\theta \tag{3.5}$$



Gambar 3.11 Grafik hubungan koordinat kartesian dengan koordinat kutub

3.2.2 Perancangan Algoritma A* (A-Star)

Algoritma A* merupakan algoritma pencarian rute dari satu titik awal yang menuju titik akhir yang telah ditentukan dalam sebuah

lingkungan yang bebas beraturan. Persamaan untuk mendapatkan nilai estimasi harga terkecil untuk solusi rute dapat dirumuskan sebagai berikut:

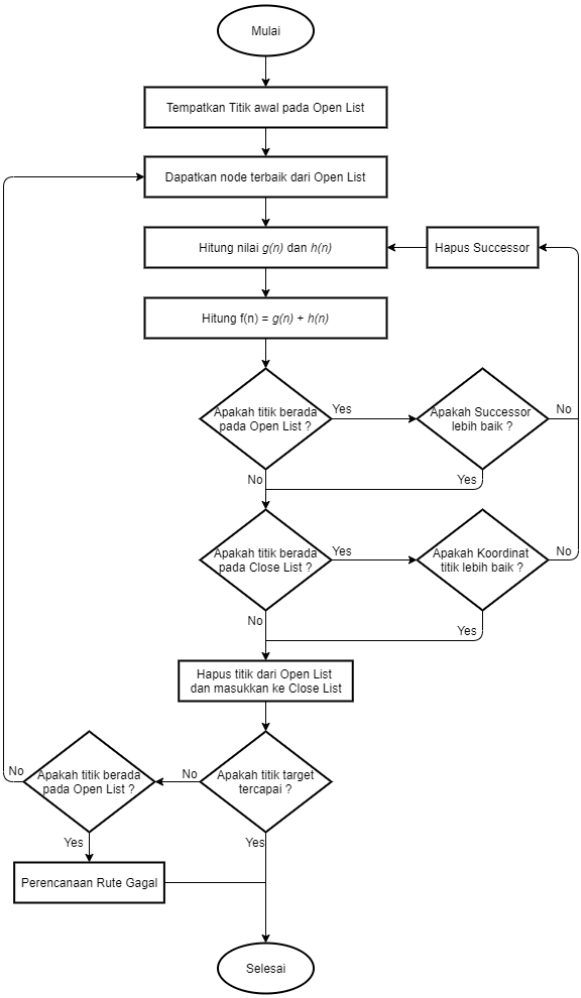
$$f(n) = g(n) + h(n) \tag{3.6}$$

Dalam hal ini, fungsi heuristik yang digunakan adalah fungsi heuristik jarak *Euclidean*. Fungsi jarak *Euclidean* berkaitan dengan prinsip teorema *Pythagoras*. Persamaan rumus jarak *Euclidean* adalah sebagai berikut:

$$d_{x,y} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3.7}$$

Gambar 3.12 menggambarkan diagram alir dari algoritma A*. Langkah-langkah dalam algoritma A* adalah sebagai berikut:

1. Tempatkan titik awal n_s pada *Open List*.
2. Jika *Open Lis*t kosong, maka proses gagal.
3. Kemudian keluarkan titik awal dari *Open List* dan tempatkan pada *Closed List* sebuah titik n_i yang memiliki fungsi *cost* (n_i) minimum.
4. Jika n_i merupakan titik akhir n_g, maka proses berhasil dengan hasil rute diambil dari proses balik, *pointer* dari n_i sampai n_s.
5. Jika sebaliknya, maka perluas n_i, buat *successor* dan letakan pada pointer balik ke n_{i-1}. Untuk setiap *successor* n_{i+1}:
   5.1. Jika n_{i+1} tidak ada dalam *Open List*, maka fungsi heuristik (n_{i+1}) dihitung sebelum total *cost* proses dari n_i sampai n_{i+1} diperoleh.
   5.2. Jika n_{i+1} telah berada dalam *Open List* atau *Closed List*, maka pointernya langsung menuju titik rute yang merupakan hasil terendah $g(n_{i+1})$.
6. Kembali ke langkah 2.

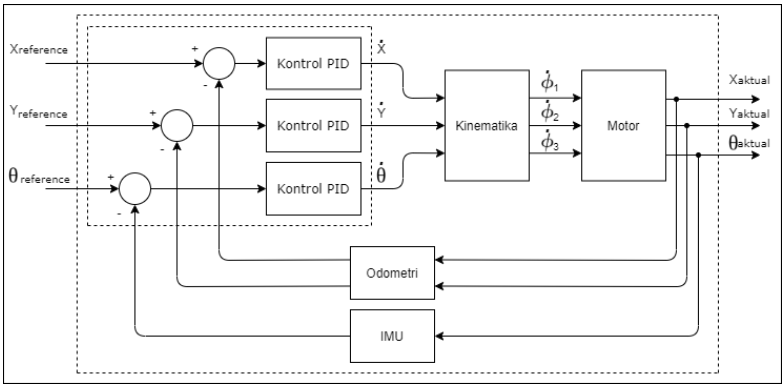


Gambar 3.12 Diagram Alir Algoritma A*

3.4 Kontrol Pergerakan Robot

Kontrol pergerakan digunakan untuk mengatur *duty cycle* sinyal PWM yang berfungsi untuk mengatur kecepatan masing-masing motor untuk menghasilkan nilai kecepatan vx, vy, dan ω. *Rotary Encoder* di

masing-masing motor akan memberikan feedback kecepatan motor. Sementara *Rotary Encoder* pada odometri akan memberikan feedback posisi robot. Diagram blok dari kontrol pergerakan robot dapat dilihat pada **gambar 3.13** :



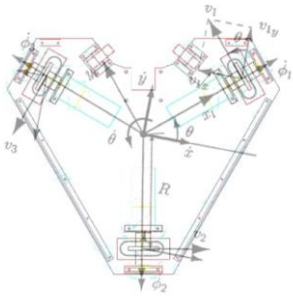
Gambar 3.13 Diagram blok kontrol pergerakan

3.4.1 Perancangan Sistem Gerak Robot

Motor yang digunakan sebagai penggerak sistem pada tugas akhir ini adalah motor DC PG-45 dengan spesifikasi suplai tegangan hingga 24 Volt dengan kecepatan 400 rpm. Motor ini dilengkapi dengan *gearbox planetary* dengan perbandingan 1:19. Direksi motor ditentukan dengan pemberian tegangan DC pada kedua input motor. Driver motor digunakan untuk menentukan kecepatan dan arah putar motor digunakan modul driver motor BTN7970. Tiga buah motor DC dipasang membentuk segitiga dengan masing-masing sudut peletakan motor DC adalah 120 derajat seperti ditunjukan pada **gambar 3.14**.

Dengan desain pemasangan motor seperti pada **gambar 3.14**, maka kinematika dari sistem dapat dirumuskan sebagai berikut :

$$\begin{bmatrix} \dot{\emptyset}_1 \\ \dot{\emptyset}_2 \\ \dot{\emptyset}_3 \end{bmatrix} = \begin{bmatrix} -\sin(\theta) & \cos(\theta) & R \\ -\sin(\theta+\alpha_2) & \cos(\theta+\alpha_2) & R \\ -\sin(\theta+\alpha_3) & \cos(\theta+\alpha_3) & R \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{3.6}$$

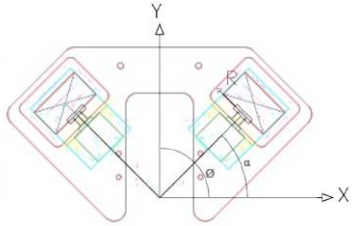


Gambar 3.14 Konfigurasi Pemasangan Motor pada Sistem

3.4.2 Perancangan Sistem Odometri

Dalam tugas akhir ini, *rotary encoder* digunakan sebagai *feedback* posisi dari robot. *Rotary encoder* yang digunakan memiliki spesifikasi kebutuhan tegangan suplai5-24VDC ± 5% dan resolusi 400 pulsa per rotasi. **Gambar 3.15** menunjukan perancangan dua buah rotary encoder yang akan dipasang pada sistem. *Rotary encoder* satu dengan yang lainnya berjarak 90 derajat. Dengan desain pemasangan *rotary encoder* seperti pada **gambar 3.15**, maka persamaan odometri dari sistem dapat dirumuskan sebagai berikut:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -\cos(225° + \theta) & -\cos(135° + \theta) \\ -\sin(225° + \theta) & -\sin(135° + \theta) \end{bmatrix} \tag{3.7}$$



Gambar 3.15 Konfigurasi Pemasangan *Rotary Encoder* pada Sistem

3.4.3 Pembangkitan Sinyal *Pulse Width Modulation* (PWM)

Pembangkitan PWM dalam mikrokontroler STM32F4 Discovery memanfaatkan *Timer/Counter*. Sehingga, *timer* pada pin yang akan dijadikan sebagai keluaran untuk membangkitkan sinyal PWM harus diaktifkan dan pin di-set sebagai AF (*Alternate Function*). Frekuensi dari *timer* yang diaktifkan dipengaruhi oleh nilai TIM *Period* dan TIM *Prescaler* dengan perumusan sebagai berikut:

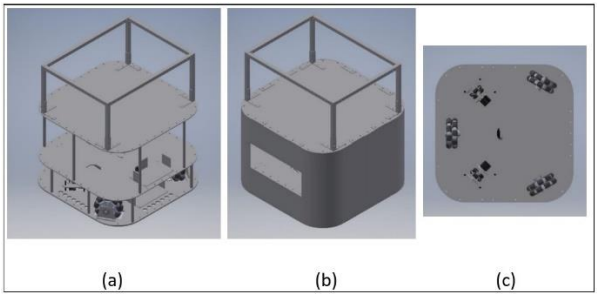
$$\frac{1}{f} = \frac{(TIM_{Period} + 1)(TIM_{Prescaler} + 1)}{\frac{System\ Clock}{APBx\ Prescaler}\ x\ 2} \tag{3.8}$$

Untuk mengatur *duty cycle* dari sinyal PWM yang dihasilkan bisa dengan cara mengatur variabel TIMx -> CCRy dengan variabel x merupakan indeks *timer* dan variabel y merupakan *indeks* kanal output sinyal PWM.

3.5 Desain Mekanik Sistem

Mekanik robot didesain menggunakan bahan dasar alumunium padat untuk bodynya dan akrilik untuk beberapa bagian lain semisal penyangga sensor dan alat elektronik lainnya Sistem didesain dengan bahan dasar alumunium padat dengan ketebalan 3 mm dan 5 mm untuk bagian kerangka badan robot dan akrilik untuk bagian penyangga sistem elektronik pada robot. Robot didesain dengan menggunakan penggerak 3

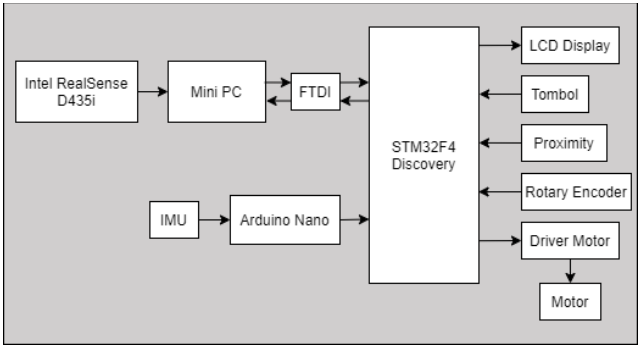
roda dengan Motor DC yang dilengkapi dengan *encoder* dan roda *omni directional*. Desain mekanik dari sistem ditunjukan pada **gambar 3.16**.



Gambar 3.16 Desain Mekanik Sistem (a) Tampak Dalam (b) Tampak Luar (c) Tampak Bawah

3.6 Perancangan Sistem Elektronik

Skema dari desain elektronik dari robot ditunjukkan pada **gambar 3.17**. Sistem elektronik yang ditunjukkan pada gambar tersebut adalah sistem elektronik keseluruhan dari platfom robot yang dibuat pada tugas akhir penulis.



Gambar 3.17 Diagram Blok Sistem Elektronik

BAB IV
PENGUJIAN DAN ANALISIS DATA

Pada bab ini akan dibahas mengenai uji coba dan analisa dari hasil perancangan sistem. Uji coba dilakukan untuk mengetahui kinerja performa dari sistem yang dibangun dengan menggunakan beberapa parameter yang telah ditentukan.

4.1 Pengujian Kecepatan Motor

Pengujian kecepatan motor dilakukan untuk mengetahui perbandingan kecepatan motor pada robot terhadap sinyal masukan *Pulse Width Modulation* (PWM) yang dikeluarkan mikrokontroler. Pengujian dilakukan menggunakan tegangan suplai 16 Volt DC. Kemudian sinyal masukan PWM diberikan kepada *driver* motor dan mengubah nilai dari persentase *duty cycle* sinyal PWM setiap 10% sehingga dapat diamati perubahan kecepatan motor terhadap sinyal masukan PWM yang diberikan. Tabel 4.1 merupakan data perbandingan antara persentase *duty cycle* sinyal masukan PWM dengan data kecepatan motor.

Tabel 4.1 Hasil pengukuran kecepatan motor dengan sinyal masukan persentase *duty cycle* dari PWM

Persentase *duty cycle* (%)	Kecepatan putar motor (rpm)
30	115
40	156
50	198
60	232
70	276
80	303
90	363

4.2 Pengujian *Rotary Encoder* Odometri

Pengujian *rotary encoder* odometri dilakukan untuk mengetahui koordinat posisi dari sistem yang dirancang. Pengujian dilakukan dengan cara menggerakan sistem pada koordinat tertentu, lalu hasil pembacaan *rotary encoder* odometri dibandingkan dengan titik koordinat sebenarnya. Pengukuran dilakukan setiap 50 cm dari mulai 0 cm hingga 500 cm. Pada

pengujian arah gerak sistem adalah maju, kiri, dan kanan. Hasil pengujian disajikan dalam **Tabel 4.2, Tabel 4.3, Tabel 4.4** sebagai berikut.

Tabel 4.2 Pengujian *rotary encoder* odometri arah gerak maju

Koordinat (cm)		Koordinat Terukur (cm)		Error (cm)	
x	y	x	y	x	y
0	50	0	52	0	2
0	100	0	98	0	2
0	150	-1	151	1	1
0	200	-3	201	3	1
0	250	-4	247	4	3
0	300	-6	299	6	1
0	350	-8	347	8	3
0	400	-11	401	11	1
0	450	-13	449	13	1
0	500	-15	497	15	3
		RMSE (cm)		2.46	1.34

Tabel 4.3 Pengujian *rotary encoder* odometri arah gerak kanan

Koordinat (cm)		Koordinat Terukur (cm)		Error (cm)	
x	y	x	y	x	y
50	0	51	4	1	4
100	0	101	9	1	9
150	0	151	13	1	13
200	0	202	18	2	18
250	0	253	23	3	23

300	0	302	29	2	29
350	0	353	35	3	35
400	0	401	41	1	41
450	0	452	48	2	48
500	0	501	54	1	54
		RMSE (cm)		1.30	5.23

Tabel 4.4 Pengujian *rotary encoder* odometri arah gerak kiri

Koordinat (cm)		Koordinat Terukur (cm)		Error (cm)	
x	y	x	y	x	y
-50	0	-49	-4	1	4
-100	0	-101	-9	1	9
-150	0	-148	-13	2	13
-200	0	-202	-19	2	19
-250	0	-249	-24	1	24
-300	0	-298	-31	2	31
-350	0	-348	-38	2	38
-400	0	-399	-47	1	47
-450	0	-451	-58	1	58
-500	0	-502	-64	2	64
		RMSE (cm)		1.22	5.54

4.3 Pengujian Sensor Giroskop

Pengujian sensor giroskop dilakukan untuk mengetahui arah hadap dari sistem. Pengujian ini dilakukan dengan memutar sensor giro dengan sudut tertentu kemudian mengamati nilai hasil keluaran dari sensor giroskop sehingga dapat diketahui perubahan nilai keluaran dari sensor

giroskop terhadap perubahan sudut yang sebenarnya. Tabel 4.5 merupakan data nilai hasil keluaran giroskop dengan perubahan sudut yang sebenarnya.

Tabel 4.5 Pengujian data nilai output sensor Giroskop

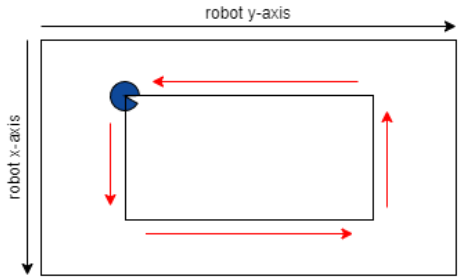
Sudut (°)	Sudut Pembacaan (°)	Error (°)
0	0.3	0.3
20	22.4	2.4
40	38.7	2.7
60	57.6	3.4
80	79.4	0.6
100	101.3	1.3
120	119.4	0.6
140	138.6	1.4
160	159.5	0.5
180	178.7	1.3
	RMSE (°)	1.20

4.4 Pengujian Algoritma Perencanaan Rute pada Sistem

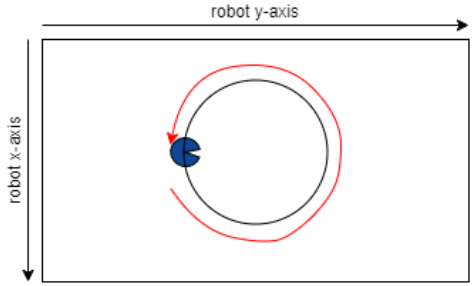
Pengujian algoritma pada sistem dilakukan dengan memberikan beberapa skenario yang identik dengan skenario yang diuji pada program perintah, yaitu dengan meletakan posisi robot dan objek berjauhan dan juga peletakan posisi *obstacle*. Trayektori yang merupakan jalur yang telah dilalui robot akan direkam, kemudian hasilnya dibandingkan dengan hasil yang diperoleh pada program perintah. Berbeda dengan program perintah, kondisi pada sistem yang sebenarnya mempertimbangkan beberapa faktor eksternal dari lingkungan, seperti tingkat kecerahan cahaya, warna dari objek dan gaya gesek dari lantai.

Skenario pertama adalah skenario tanpa adanya *obstacle* yang ditunjukkan pada **Gambar 4.1**. Robot akan membuat jalur menuju ke koordinat yang telah ditentukan. Ada 3 titik tuju yang harus dicapai oleh sistem kemudian kembali ke titik awal. Skenario kedua adalah skenario

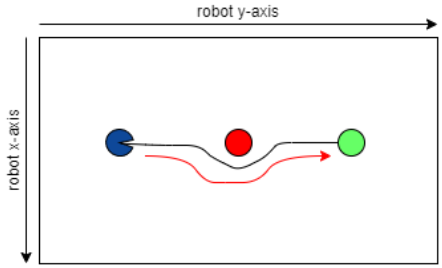
pergerakan melingkar yang ditunjukkan seperti pada **Gambar 4.2**. Skenario ketiga adalah skenario ketika robot dihadapkan dengan *obstacle* dan objek target berada di belakangnya yang ditunjukkan seperti pada **Gambar 4.3**. Dari hasil pengujian algoritma pada program dan pada sistem yang sebenernya, maka perbandingan antara pergerakan *trajectory* yang direkam pada program dan pada sistem yang sebenarnya dapat digambarkan dalam bentuk grafik.



Gambar 4.1 Skenario rute pergerakan translasi



Gambar 4.2 Skenario rute pergerakan rotasi

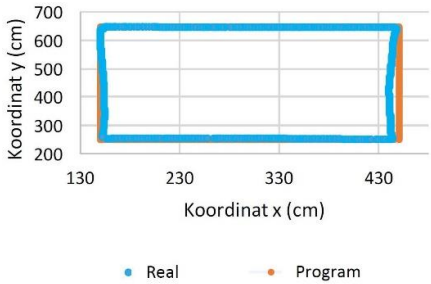


Gambar 4.3 Skenario rute menghindari obstacle

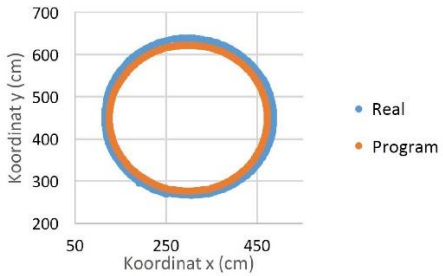
Gambar 4.4 menunjukan grafik perbandingan hasil yang diperoleh dari program dan pada sistem yang sesungguhnya dengan skenario pergerakan translasi. Grafik berwarna biru menunjukan *trajectory* yang dihasilkan oleh pergerakan robot, sedangkan grafik berwarna jingga merupakan yang dihasilkan oleh program perintah. Dari data-data yang digambarkan pada grafik **gambar 4.4** dapat diperoleh perhitungan nilai *error* RMS sebesar 6.48 cm.

Gambar 4.5 menunjukan grafik perbandingan hasil yang diperoleh dari program dan pada sistem yang sesungguhnya dengan skenario pergerakan rotasi. Grafik berwarna biru menunjukan *trajectory* yang dihasilkan oleh pergerakan robot, sedangkan grafik berwarna jingga merupakan yang dihasilkan oleh program perintah. Dari data-data yang digambarkan pada grafik **gambar 4.5** dapat diperoleh perhitungan nilai *error* RMS sebesar 10.63 cm.

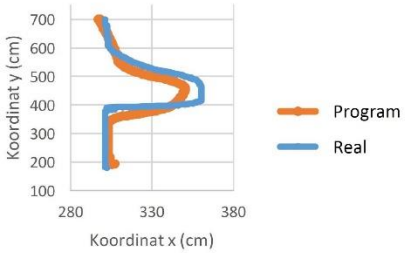
Gambar 4.6 menunjukan grafik perbandingan hasil yang diperoleh dari program dan pada sistem yang sesungguhnya dengan skenario menghindari *obstacle avoidance*. Grafik berwarna biru menunjukan *trajectory* yang dihasilkan oleh pergerakan robot, sedangkan grafik berwarna jingga merupakan yang dihasilkan oleh program perintah. Dari



Gambar 4.4 Grafik perbandingan trayektori pada program dengan sistem sebenarnya dengan skenario pergerakan translasi



Gambar 4.5 Grafik perbandingan trayektori pada program dengan sistem sebenarnya dengan skenario pergerakan rotasi



Gambar 4.6 Grafik perbandingan trayektori pada program dengan sistem sebenarnya dengan skenario menghindari *obstacle*

data-data yang digambarkan pada grafik **gambar 4.6** dapat diperoleh perhitungan nilai *error* RMS sebesar 17.78 cm.

4.5 Pengujian Respon Sistem terhadap Algoritma

Pengujian respon sistem terhadap algoritma ini dilakukan untuk mengetahui seberapa cepat komputer *mini* pada sistem dapat memproses keseluruhan algoritma yang telah dibuat dan menjalankan tugasnya dalam mendekati objek target. **Tabel 4.6** menunjukan hasil pengujian respon sistem terhadap algoritma yang telah dibuat.

Tabel 4.6 Pengujian Respon Sistem terhadap Algoritma

Titik Awal	Titik Akhir	*Obstacle*	Waktu
300, 200	300, 500	0	11.21
300, 200	300, 500	0	11.34
300, 200	300, 500	1	12.85
300, 200	300, 500	1	12.61
300, 200	300, 500	2	14.15
300, 200	300, 500	2	13.78

Pada pengujian ini, dilakukan 6 kali percobaan dengan kecepatan motor yang sama, posisi titik awal dan titik akhir pada koordinat yang sama, akan tetapi jumlah dan peletakan *obstacle* yang berbeda-beda. Dari data-data tersebut, maka dapat dihitung hasil rata-rata dari waktu yang ditempuh dari titik target menuju titik akhir adalah sebagai berikut :

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{6}\sum_{i=1}^{n} x_i = 12{,}65 \text{ detik}$$

BAB V
PENUTUP

Dari perancangan sistem, pengujian, dan analisis yang telah dilakukan, penelitian ini dapat dirangkum dalam sebuah kesimpulan. Hal-hal terkait kendala dan kesulitan yang dihadapi selama penelitian ini akan dibahas pada subbab saran untuk membantu pengembangan penelitian selanjutnya.

5.1 Kesimpulan

Kesimpulan yang dapat diambil setelah melakukan pengujian pada tugas akhir pada penilitian ini adalah sebagai berikut :

1. Hasil pengujian perbandingan antara *trajectory* pada sistem yang sebenarnya dan pada program perintah dengan skenario membentuk jalur persegi memiliki nilai *error* RMS sebesar 6.48 cm.
2. Hasil pengujian perbandingan antara *trajectory* pada sistem yang sebenarnya dan pada program perintah dengan skenario membentuk jalur melingkar memiliki nilai *error* RMS sebesar 10.63 cm.
3. Hasil pengujian perbandingan antara *trajectory* pada sistem yang sebenarnya dan pada program perintah dengan skenario menghindari *obstacle* memiliki nilai *error* RMS sebesar 17.78 cm.
4. Waktu rata-rata respon sistem terhadap algoritma dengan koordinat posisi awal dan akhir yang sama, tetapi dengan jumlah dan peletakan obstacle yang berbeda adalah 12,65 detik.

5.2 Saran

Dari proses perancangan sistem dan perancangan yang ada terdapat kendala dan kesulitan yang dapat menjadi saran sebagai berikut:

1. Pengembangan algoritma untuk penelitian berikutnya yang dapat diaplikasikan pada *obstacle* yang dinamis.
2. Penggunaan *depth camera* dengan tingkat akurasi yang lebih baik guna menghindari kesalahan pengambilan citra.
3. Penggunaan CPU yang lebih *powerful* agar dapat melakukan komputasi algoritma yang lebih kompleks dengan respon yang lebih cepat terutama pada ektraksi fitur dan deteksi objek target yang ingin dituju.

.

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

[1] Suyanto, "*Artificial Intelligence Searching, Reasoning, Planning, Learning*," Bandung, Jawa Barat: Informatika Bandung, 2011, p. 15.

[2] S. J. Russell and P. Norvig, "*Artificial Intelligence A Modern Approach Second Edition*," M. Pompili, Ed., New Jersey: Alan Apt, 1995, p. 99-100.

[3] M. Higgins, "*A Robot in Every Home: an Introduction to Personal Robots & Brand-Name buyer's guide"*, 1st ed. Oakland, Calif: Kensington Pub. Co, 1985, p. 4.

[4] F. Tatji, G. Szayer, B. Kovács and P. Korondi, "*Robot base with holonomic drive*," in *19th World Congress The International Federation of Automatic Control*, Cape Town, 2014.

[5] R. Siegwart and I. R. Nourbakhsh, Introduction to Autonomous Mobile Robots, London: MIT Press, 2004, pp. 75-78.

[6] L. Feng, L. Xiaoyu, dan C. Yi, "*An efficient detection method for rare colored capsule based on RGB and HSV color space*," dalam *2014 IEEE International Conference on Granular Computing (GrC)*, Noboribetsu, Japan, 2014, pp. 175–178.

[7] P. S. Tantra, "Penentuan Posisi Robot Sepak Bola Beroda Berdasarkan Pengindraan Visual," p. 88, 2018.

[8] L. Yu, X. Chen, dan S. Zhou, "Research of Image Main Objects Detection Algorithm Based on Deep Learning," dalam *2018 IEEE 3rd International Conference on Image, Vision and Computing (ICIVC)*, Chongqing, 2018, p. 70–75.

[9] STMicroelectronics, UM1472 User manual Discovery kit for STM32F407/417 lines, 2014.

[10] . “Which Intel RealSense device is right for you? (Updated June 2020),” *Intel® RealSenseTM Depth and Tracking Cameras*, Mar. 21, 2019.https://www.intelrealsense.com/which-device-is-right-for-you/ (accessed Jun. 1, 2020).

[11] “TensorFlow,” *TensorFlow*. [Daring]. Tersedia pada: https://www.tensorflow.org/. [Diakses: 28-Mei-2020].

[12] “Google Colaboratory.” [Daring]. Tersedia pada: https://colab.research.google.com/notebooks/welcome.ipynb. [Diakses: 28-Mei-2020].

[13] “About OpenCV.” [Daring]. Tersedia pada: https://opencv.org/about/. [Diakses: 28-Mei-2020].

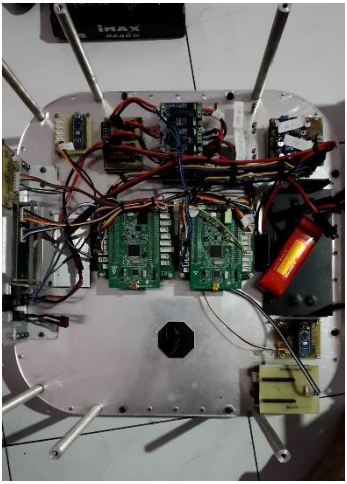
.

.

.

LAMPIRAN

1.Realisasi Sistem



2.Source Code Program

A.Program Odometri

```
#include "odometry.h"

float posisi_x_buffer = 0, posisi_y_buffer = 0;
float posisi_x_offset = 0, posisi_y_offset = 0;
float posisi_x = 0, posisi_y = 0;

char gyro_status = 0;
float gyro_buffer = 0, gyro_offset = 90;
float gyro_derajat = 90, gyro_radian = 1.5707963268;

void odometry_VectorKinematic(void)
{
short int kecepatan_odometry0 = odometry0;
odometry0 = 0;
short int kecepatan_odometry1 = odometry1;
odometry1 = 0;
```

```
float buffer_x[2];
float buffer_y[2];

buffer_x[0] = kecepatan_odometry0 * cosf(gyro_radian +
0.785398);
buffer_x[1] = kecepatan_odometry1 * cosf(gyro_radian +
2.356190);

buffer_y[0] = kecepatan_odometry0 * sinf(gyro_radian +
0.785398);
buffer_y[1] = kecepatan_odometry1 * sinf(gyro_radian +
2.356190);

posisi_x_buffer += (buffer_x[0] + buffer_x[1]) *
odometry_to_cm;
posisi_y_buffer += (buffer_y[0] + buffer_y[1]) *
odometry_to_cm;

posisi_x = posisi_x_buffer - posisi_x_offset;
posisi_y = posisi_y_buffer - posisi_y_offset;
}

void receive_gyro_serial()
{
gyro_derajat = (gyro_offset - gyro_buffer);
gyro_radian = (gyro_offset - gyro_buffer) * 0.01745329252;

while (gyro_derajat > 180)
       gyro_derajat -= 360;
while (gyro_derajat < -180)
       gyro_derajat += 360;

while (gyro_radian > 3.14159265359)
       gyro_radian -= 6.28318530718;
while (gyro_radian < -3.14159265359)
       gyro_radian += 6.28318530718;
}
```

B.Program Kinematik

```
#include "motor.h"

extern short int kecepatan_x;
extern short int kecepatan_y;
extern short int kecepatan_sudut;

void motor_VectorKinematic(short int vx, short int vy, short
int vsudut)
{
        motor_SetPoint[0] = (short int) ((vx * cosf(120 *
0.0174533)) + (vy * sinf(120 * 0.0174533)) + vsudut);
        motor_SetPoint[1] = (short int) ((vx * cosf(240 *
0.0174533)) + (vy * sinf(240 * 0.0174533)) + vsudut);
        motor_SetPoint[2] = (short int) ((vx * cosf(0 *
0.0174533)) + (vy * sinf(0 * 0.0174533)) + vsudut);
}

void motor_VeloControl(void)
{
        float proportional_motor[3], integral_motor[3],
derivative_motor[3];
        float error_velo_motor[3], sum_error_velo_motor[3],
previous_error_velo_motor[3];
        short int output_velo_motor[3];

        motor_velo[0] = Encoder0;
        motor_velo[1] = Encoder1;
        motor_velo[2] = Encoder2;

        Encoder0=Encoder1=Encoder2=0;

        for(int i = 0; i < 3; i++)
        {
                error_velo_motor[i] = motor_SetPoint[i] -
motor_velo[i];

                sum_error_velo_motor[i] += error_velo_motor[i];

                if(integral_motor[i] > integral_output_max)
integral_motor[i] = integral_output_max;
                else if(integral_motor[i] <
integral_output_min) integral_motor[i] = integral_output_min;

                if(derivative_motor[i] > derivative_output_max)
derivative_motor[i] = derivative_output_max;
```

```
                else if(derivative_motor[i] <
derivative_output_min) derivative_motor[i] =
derivative_output_min;

                proportional_motor[i] = KP_motor *
error_velo_motor[i];
                integral_motor[i] = KI_motor *
sum_error_velo_motor[i];
                derivative_motor[i] = KD_motor *
(error_velo_motor[i] - previous_error_velo_motor[i]);

                previous_error_velo_motor[i] =
error_velo_motor[i];

                output_velo_motor[i] = (short int)
proportional_motor[i] + integral_motor[i] +
derivative_motor[i];

                if(output_velo_motor[i] >
output_max_velo_motor) output_velo_motor[i] =
output_max_velo_motor;
                else if(output_velo_motor[i] <
output_min_velo_motor) output_velo_motor[i] =
output_min_velo_motor;
        }

        if (motor_status)
                for (int i = 0; i < 3; i++)
                {
                        proportional_motor[i] = 0;
                        integral_motor[i] = 0;
                        derivative_motor[i] = 0;
                        error_velo_motor[i] = 0;
                        sum_error_velo_motor[i] = 0;
                        previous_error_velo_motor[i] = 0;
                        output_velo_motor[i] = 0;
                }

        motor_SetPWM(output_velo_motor);
}

void motor_SetPWM(short int outputPWM[3])
{
        if (outputPWM[0] > 0)
        {
                HAL_GPIO_WritePin(DirAMotor0_GPIO_Port,
DirAMotor0_Pin, GPIO_PIN_SET);
```

```
                HAL_GPIO_WritePin(DirBMotor0_GPIO_Port,
DirBMotor0_Pin, GPIO_PIN_RESET);
        }
        else if (outputPWM[0] < 0)
        {
                HAL_GPIO_WritePin(DirAMotor0_GPIO_Port,
DirAMotor0_Pin, GPIO_PIN_RESET);
                HAL_GPIO_WritePin(DirBMotor0_GPIO_Port,
DirBMotor0_Pin, GPIO_PIN_SET);
        }

//==============================================================
================================

        if (outputPWM[1] > 0)
        {
                HAL_GPIO_WritePin(DirAMotor1_GPIO_Port,
DirAMotor1_Pin, GPIO_PIN_SET);
                HAL_GPIO_WritePin(DirBMotor1_GPIO_Port,
DirBMotor1_Pin, GPIO_PIN_RESET);
        }
        else if (outputPWM[1] < 0)
        {
                HAL_GPIO_WritePin(DirAMotor1_GPIO_Port,
DirAMotor1_Pin, GPIO_PIN_RESET);
                HAL_GPIO_WritePin(DirBMotor1_GPIO_Port,
DirBMotor1_Pin, GPIO_PIN_SET);
        }

//==============================================================
================================

        if (outputPWM[2] > 0)
        {
                HAL_GPIO_WritePin(DirAMotor2_GPIO_Port,
DirAMotor2_Pin, GPIO_PIN_SET);
                HAL_GPIO_WritePin(DirBMotor2_GPIO_Port,
DirBMotor2_Pin, GPIO_PIN_RESET);
        }
        else if (outputPWM[2] < 0)
        {
                HAL_GPIO_WritePin(DirAMotor2_GPIO_Port,
DirAMotor2_Pin, GPIO_PIN_RESET);
                HAL_GPIO_WritePin(DirBMotor2_GPIO_Port,
DirBMotor2_Pin, GPIO_PIN_SET);
        }

        PWM0 = abs(outputPWM[0]);
```

```
        PWM1 = abs(outputPWM[1]);
        PWM2 = abs(outputPWM[2]);
}
```

C.Program A-Star

```
#include <algorithm>
#include "PathPlanning.h"
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

using namespace std::placeholders;
bool AStar::matriks2d::operator==(const matriks2d
&koordinat_)
{
    return(x == koordinat_.x && y == koordinat_.y);
}

AStar::matriks2d operator + (const AStar::matriks2d
&kiri_,const AStar::matriks2d &kanan_)
{
    return{ kiri_.x + kanan_.x, kiri_.y + kanan_.y };
}

AStar::titik::titik(matriks2d koordinat_, titik
*parent_)
{
    parent = parent_;
    koordinat = koordinat_;
    G = H = 0;
}

AStar::uint AStar::titik::hasilF()
{
    return G + H;
}

AStar::aktifasi::aktifasi()
{
    pergerakanDiagonal(false);
    setHeuristik(&Heuristik::euclidean);
```

```
    arah = { { 0, 1 },{ 1, 0 },{ 0, -1 },{ -1, 0 },{ -1,
-1 },{ 1, 1 },{ -1, 1 },{ 1, -1 }};
}

void AStar::aktifasi::LuasArea(matriks2d ukuranMap_)
{
    ukuranMap = ukuranMap_;
}

void AStar::aktifasi::pergerakanDiagonal(bool enable_)
{
    direksi = (enable_ ? 8 : 4);
}

void AStar::aktifasi::setHeuristik(fungsiHeuristik
heuristik_)
{
    heuristik = std::bind(heuristik_, _1, _2);
}

void AStar::aktifasi::addObstacle(matriks2d koordinat_)
{
    obstacle.push_back(koordinat_);
}

void AStar::aktifasi::removeObstacle(matriks2d
koordinat_)
{
auto that = std::find(obstacle.begin(),
obstacle.end(),koordinat_);
    if (that != obstacle.end()) {
        obstacle.erase(that);
    }
}

void AStar::aktifasi::clearObstacle()
{
    obstacle.clear();
}
```

```
AStar::listKoordinat
AStar::aktifasi::cariJalur(matriks2d awal_, matriks2d
akhir_)
{
    titik *koorSekarang = nullptr;
    nodeset openSet, closedSet;
    openSet.insert(new titik(awal_));
    while (!openSet.empty()) {
        koorSekarang = *openSet.begin();
        for (auto titik : openSet) {
            if (titik->hasilF() <= koorSekarang-
>hasilF()) {
                koorSekarang = titik;
            }
        }
        if (koorSekarang->koordinat == akhir_) {
        break;
        }
        closedSet.insert(koorSekarang);
        openSet.erase(std::find(openSet.begin(),
openSet.end(), koorSekarang));
        for (uint i = 0; i < direksi; ++i) {
            matriks2d koorBaru(koorSekarang->koordinat +
arah[i]);
            if (deteksiObstacle(koorBaru) ||
cariTitik(closedSet, koorBaru)) {
                continue;
            }
            uint harga = koorSekarang->G + ((i < 4) ?
10:14);
            titik *successor = cariTitik(openSet,
koorBaru);
            if (successor == nullptr) {
                successor = new
titik(koorBaru,koorSekarang);
                successor->G = harga;
                successor->H = heuristik(successor-
>koordinat, akhir_);
                openSet.insert(successor);
            }
            else if (harga < successor->G) {
                successor->parent = koorSekarang;
```

```
                successor->G = harga;
            }
        }
    }
    listKoordinat jalur;
    while (koorSekarang != nullptr) {
        jalur.push_back(koorSekarang->koordinat);
        koorSekarang = koorSekarang->parent;
    }
    bukanTitik(openSet);
    bukanTitik(closedSet);
    return jalur;
}

AStar::titik* AStar::aktifasi::cariTitik(nodeset&
titik_,matriks2d koordinat_)
{
    for (auto titik : titik_) {
        if (titik->koordinat == koordinat_) {
        return titik;
        }
    }
    return nullptr;
}

void AStar::aktifasi::bukanTitik(nodeset& titik_)
{
    for (auto that = titik_.begin(); that !=
titik_.end();) {
        delete *that;
        that = titik_.erase(that);
    }
}

bool AStar::aktifasi::deteksiObstacle(matriks2d
koordinat_)
{
    if (koordinat_.x < 0 || koordinat_.x >= ukuranMap.x
|| koordinat_.y < 0 || koordinat_.y >= ukuranMap.y
    ||std::find(obstacle.begin(), obstacle.end(),
koordinat_) != obstacle.end()) {
        return true;
```

```
    }
    return false;
}

AStar::matriks2d AStar::Heuristik::nilaiDelta(matriks2d
awal_, matriks2d akhir_)
{
    return{ static_cast<int>(fabs(awal_.x - akhir_.x)),
    static_cast<int>(fabs(awal_.y - akhir_.y)) };
}

AStar::uint AStar::Heuristik::euclidean(matriks2d awal_,
matriks2d akhir_)
{
    auto delta = std::move(nilaiDelta(awal_, akhir_));
    return static_cast<uint>(100 * sqrt(pow(delta.x, 2)
+ pow(delta.y, 2)));
}
```

D.Program Motion

```
#include "motion.h"
#include "math.h"
#include "stdlib.h"

extern int motor_vector_speed[3];

extern float globalPositionFusion[3],
globalSpeedFusion[3], localSpeedFusion[3];

extern char  uart3_data,uart3_status,gyro_terima[6];

extern int motor_vector_speed[3];
extern char lcd[21];

void xysudut(float targetX, float targetY, float
targetSudut, int V_x, int V_y, int V_sudut, int iterasi,
int tombol_x)
{
 int globalSpeedOut[3];
 float kp[3] =
 { 4, 4, 8 };
 float ki[3] =
```

```
 { 0, 0, 0 };
 float kd[3] =
 { 0.5, 0.5, 1 };
 static float error[3], sumError[3], previousError[3];
 float proportional[3], integral[3], derivative[3];

 float globalPositionSetPoint[3];
 globalPositionSetPoint[0] = targetX;
 globalPositionSetPoint[1] = targetY;
 globalPositionSetPoint[2] = targetSudut;

 for (int i = 0; i < 3; i++)
 {

        error[i] = globalPositionSetPoint[i] -
globalPositionFusion[i];
        if (i == 2)
        {
                error[2] = atan2f(sinf(error[2] *
0.0174533), cosf(error[2] * 0.0174533)) * 57.2958;
        }
        sumError[i] += error[i];

        proportional[i] = kp[i] * error[i];
        integral[i] = ki[i] * sumError[i];
        derivative[i] = kd[i] * (error[i] -
previousError[i]);

        previousError[i] = error[i];

        globalSpeedOut[i] = (int) proportional[i] + (int)
integral[i] + (int) derivative[i];
 }

 if (globalSpeedOut[0] > V_x)
        globalSpeedOut[0] = V_x;
 if (globalSpeedOut[0] < -V_x)
        globalSpeedOut[0] = -V_x;

 if (globalSpeedOut[1] > V_y)
        globalSpeedOut[1] = V_y;
 if (globalSpeedOut[1] < -V_y)
```

```
        globalSpeedOut[1] = -V_y;

 if (globalSpeedOut[2] > V_sudut)
        globalSpeedOut[2] = V_sudut;
 if (globalSpeedOut[2] < -V_sudut)
        globalSpeedOut[2] = -V_sudut;

 MotorLocalSpeed(motor_vector_speed, globalSpeedOut,
globalPositionFusion[2] * 0.0174533);
 }
}

void jalan_x(float target_x, float target_y, float
target_sudut, int V_x, int V_y, int V_sudut)
{
 lcd_clear();


 int globalSpeedOut[3];
 float kp[3] =
 { 1.2, 1.5, 7.2 };
 float ki[3] =
 { 0, 0, 0 };
 float kd[3] =
 { 0.8, 1.5, 3.8 };
 static float error[3], sumError[3], previousError[3];
 float proportional[3], integral[3], derivative[3];
 float posisiAwal;

 float globalPositionSetPoint[3];
 globalPositionSetPoint[0] = target_x;
 globalPositionSetPoint[1] = target_y;
 globalPositionSetPoint[2] = target_sudut;


 posisiAwal=globalPositionFusion[0];

 for (int i = 0; i < 3; i++)
 {
        error[i] = globalPositionSetPoint[i] -
globalPositionFusion[i];
 }
```

```
 while ((globalPositionFusion[0] <
globalPositionSetPoint[0] && globalPositionSetPoint[0] >
posisiAwal)
                || (globalPositionFusion[0] >
globalPositionSetPoint[0] && globalPositionSetPoint[0] <
posisiAwal))
 {
        for (int i = 0; i < 3; i++)
        {
                error[i] = globalPositionSetPoint[i] -
globalPositionFusion[i];
                if (i == 2)
                {
                        error[2] = atan2f(sinf(error[2] *
0.0174533), cosf(error[2] * 0.0174533)) * 57.2958;
                }
                sumError[i] += error[i];

                proportional[i] = kp[i] * error[i];
                integral[i] = ki[i] * sumError[i];
                derivative[i] = kd[i] * (error[i] -
previousError[i]);

                previousError[i] = error[i];

                globalSpeedOut[i] = (int) proportional[i]
+ (int) integral[i] + (int) derivative[i];
        }

        if (globalSpeedOut[0] > V_x)
                globalSpeedOut[0] = V_x;
        if (globalSpeedOut[0] < -V_x)
                globalSpeedOut[0] = -V_x;

        if (globalSpeedOut[1] > V_y)
                globalSpeedOut[1] = V_y;
        if (globalSpeedOut[1] < -V_y)
                globalSpeedOut[1] = -V_y;

        if (globalSpeedOut[2] > V_sudut)
                globalSpeedOut[2] = V_sudut;
```

```
        if (globalSpeedOut[2] < -V_sudut)
                globalSpeedOut[2] = -V_sudut;

        MotorLocalSpeed(motor_vector_speed,
globalSpeedOut, globalPositionFusion[2] * 0.0174533);
 }
}

void jalan_x_offset_min(float target_x, float target_y,
float target_sudut, int V_x, int V_y, int V_sudut)
{
 int globalSpeedOut[3];
 float kp[3] =
 { 1.2, 1.5, 7.2 };
 float ki[3] =
 { 0, 0, 0 };
 float kd[3] =
 { 0.8, 1.5, 3.8 };
 static float error[3], sumError[3], previousError[3];
 float proportional[3], integral[3], derivative[3];
 float posisiAwal;

 float globalPositionSetPoint[3];
 globalPositionSetPoint[0] = target_x;
 globalPositionSetPoint[1] = target_y;
 globalPositionSetPoint[2] = target_sudut;

 posisiAwal=globalPositionFusion[0];

 for (int i = 0; i < 3; i++)
 {
        error[i] = globalPositionSetPoint[i] -
globalPositionFusion[i];
 }

 while ((globalPositionFusion[0] <
globalPositionSetPoint[0]-500 &&
globalPositionSetPoint[0]-500 > posisiAwal)
                || (globalPositionFusion[0] >
globalPositionSetPoint[0]+500 &&
globalPositionSetPoint[0]+500 < posisiAwal))
 {
```

```
        for (int i = 0; i < 3; i++)
        {
                error[i] = globalPositionSetPoint[i] -
globalPositionFusion[i];
                if (i == 2)
                {
                        error[2] = atan2f(sinf(error[2] *
0.0174533), cosf(error[2] * 0.0174533)) * 57.2958;
                }
                sumError[i] += error[i];

                proportional[i] = kp[i] * error[i];
                integral[i] = ki[i] * sumError[i];
                derivative[i] = kd[i] * (error[i] -
previousError[i]);

                previousError[i] = error[i];

                globalSpeedOut[i] = (int) proportional[i]
+ (int) integral[i] + (int) derivative[i];
        }

        if (globalSpeedOut[0] > V_x)
                globalSpeedOut[0] = V_x;
        if (globalSpeedOut[0] < -V_x)
                globalSpeedOut[0] = -V_x;

        if (globalSpeedOut[1] > V_y)
                globalSpeedOut[1] = V_y;
        if (globalSpeedOut[1] < -V_y)
                globalSpeedOut[1] = -V_y;

        if (globalSpeedOut[2] > V_sudut)
                globalSpeedOut[2] = V_sudut;
        if (globalSpeedOut[2] < -V_sudut)
                globalSpeedOut[2] = -V_sudut;

        MotorLocalSpeed(motor_vector_speed,
globalSpeedOut, globalPositionFusion[2] * 0.0174533);
 }
}
```

```
void jalan_y(float target_x, float target_y, float
target_sudut, int V_x, int V_y, int V_sudut)
{
 lcd_clear();

 int globalSpeedOut[3];
 float kp[3] =
 { 1.2, 1.2, 7.2 };
 float ki[3] =
 { 0, 0, 0 };
 float kd[3] =
 { 0.5, 0.5, 3.8 };
 static float error[3], sumError[3], previousError[3];
 float proportional[3], integral[3], derivative[3];
 float posisiAwal;

 float globalPositionSetPoint[3];
 globalPositionSetPoint[0] = target_x;
 globalPositionSetPoint[1] = target_y;
 globalPositionSetPoint[2] = target_sudut;

 posisiAwal=globalPositionFusion[1];

 while (((globalPositionFusion[1] <
globalPositionSetPoint[1] && globalPositionSetPoint[1] >
posisiAwal)
                || (globalPositionFusion[1] >
globalPositionSetPoint[1] && globalPositionSetPoint[1] <
posisiAwal)))
 {
        for (int i = 0; i < 3; i++)
        {
                error[i] = globalPositionSetPoint[i] -
globalPositionFusion[i];
                if (i == 2)
                {
                        error[2] = atan2f(sinf(error[2] *
0.0174533), cosf(error[2] * 0.0174533)) * 57.2958;
                }
                sumError[i] += error[i];

                proportional[i] = kp[i] * error[i];
```

```
                integral[i] = ki[i] * sumError[i];
                derivative[i] = kd[i] * (error[i] -
previousError[i]);

                previousError[i] = error[i];

                globalSpeedOut[i] = (int) proportional[i]
+ (int) integral[i] + (int) derivative[i];

        }
        if (globalSpeedOut[0] > V_x)
                globalSpeedOut[0] = V_x;
        if (globalSpeedOut[0] < -V_x)
                globalSpeedOut[0] = -V_x;

        if (globalSpeedOut[1] > V_y)
                globalSpeedOut[1] = V_y;
        if (globalSpeedOut[1] < -V_y)
                globalSpeedOut[1] = -V_y;

        if (globalSpeedOut[2] > V_sudut)
                globalSpeedOut[2] = V_sudut;
        if (globalSpeedOut[2] < -V_sudut)
                globalSpeedOut[2] = -V_sudut;

        MotorLocalSpeed(motor_vector_speed,
globalSpeedOut, globalPositionFusion[2] * 0.0174533);
 }
}

void jalan_y_offset_min(float target_x, float target_y,
float target_sudut, int V_x, int V_y, int V_sudut)
{
 int globalSpeedOut[3];
 float kp[3] =
 { 1.2, 1.5, 7.2 };
 float ki[3] =
 { 0, 0, 0 };
 float kd[3] =
 { 0.8, 1.5, 3.8 };
 static float error[3], sumError[3], previousError[3];
 float proportional[3], integral[3], derivative[3];
```

```
 float posisiAwal;

 float globalPositionSetPoint[3];
 globalPositionSetPoint[0] = target_x;
 globalPositionSetPoint[1] = target_y;
 globalPositionSetPoint[2] = target_sudut;

 posisiAwal=globalPositionFusion[0];

 for (int i = 0; i < 3; i++)
 {
        error[i] = globalPositionSetPoint[i] -
globalPositionFusion[i];
 }

 while ((globalPositionFusion[1] <
globalPositionSetPoint[1]-500 &&
globalPositionSetPoint[1]-500 > posisiAwal)
                || (globalPositionFusion[1] >
globalPositionSetPoint[1]+500 &&
globalPositionSetPoint[1]+500 < posisiAwal))
 {
        for (int i = 0; i < 3; i++)
        {
                error[i] = globalPositionSetPoint[i] -
globalPositionFusion[i];
                if (i == 2)
                {
                        error[2] = atan2f(sinf(error[2] *
0.0174533), cosf(error[2] * 0.0174533)) * 57.2958;
                }
                sumError[i] += error[i];

                proportional[i] = kp[i] * error[i];
                integral[i] = ki[i] * sumError[i];
                derivative[i] = kd[i] * (error[i] -
previousError[i]);

                previousError[i] = error[i];

                globalSpeedOut[i] = (int) proportional[i]
+ (int) integral[i] + (int) derivative[i];
```

```
        }

        if (globalSpeedOut[0] > V_x)
                globalSpeedOut[0] = V_x;
        if (globalSpeedOut[0] < -V_x)
                globalSpeedOut[0] = -V_x;

        if (globalSpeedOut[1] > V_y)
                globalSpeedOut[1] = V_y;
        if (globalSpeedOut[1] < -V_y)
                globalSpeedOut[1] = -V_y;

        if (globalSpeedOut[2] > V_sudut)
                globalSpeedOut[2] = V_sudut;
        if (globalSpeedOut[2] < -V_sudut)
                globalSpeedOut[2] = -V_sudut;

        MotorLocalSpeed(motor_vector_speed,
globalSpeedOut, globalPositionFusion[2] * 0.0174533);
 }
}
```