TUGAS AKHIR – TE 141599

SISTEM PERENCANAAN RUTE GERAK PADA ROBOT SEPAKBOLA BERODA

Aulia Aditya Rachman
NRP. 2215105024

Dosen Pembimbing
Dr. Ir. Hendra Kusuma, M.Eng.Sc.
Ronny Mardiyanto, S.T., M.T., Ph.D.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2017

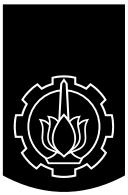
TUGAS AKHIR – TE 141599

SISTEM PERENCANAAN RUTE GERAK PADA ROBOT SEPAKBOLA BERODA

Aulia Aditya Rachman
NRP. 2215105024

Dosen Pembimbing
Dr. Ir. Hendra Kusuma, M.Eng.Sc.
Ronny Mardiyanto, S.T., M.T., Ph.D.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2017



ITS
Institut
Teknologi
Sepuluh Nopember

FINAL PROJECT – TE 141599

PATH PLANNING SYSTEM IN WHEELED SOCCER ROBOT

Aulia Aditya Rachman
NRP 2215105024

Advisor
Dr. Ir. Hendra Kusuma, M.Eng.Sc.
Ronny Mardiyanto, S.T., M.T., Ph.D.

ELECTRICAL ENGINEERING DEPARTEMENT
Faculty Of Electrical Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2017

PERNYATAAN KEASLIAN
TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "SISTEM PERENCANAAN RUTE GERAK PADA ROBOT SEPAK BOLA BERODA" adalah benar-benar hasil karya intelektual sendiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya orang 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 dengan peraturan yang berlaku.

Surabaya, Juli 2017

Aulia Aditya Rachman
NRP. 2215105024

SISTEM PERENCANAAN RUTE GERAK PADA ROBOT SEPAK BOLA BERODA

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik
Pada
Bidang Studi Elektronika
Departemen Teknik Elektro
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui:

Dosen Pembimbing I	Dosen Pembimbing II
Dr. Ir. Hendra Kusuma, M.Eng.Sc.	**Ronny Mardiyanto, S.T., M.T., Ph.D.**
NIP. 196409021989031003	**NIP. 198101182003121003**

KEMENTERIAN RISET, TEKNOLOGI, DAN PENDIDIKAN TINGGI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
FAKULTAS TEKNOLOGI ELEKTRO
DEPARTEMEN TEKNIK ELEKTRO

SURABAYA
JULI, 2017

SISTEM PERENCANAAN RUTE GERAK PADA ROBOT SEPAK BOLA BERODA

Nama : Aulia Aditya Rachman
Pembimbing I : Dr. Ir Hendra Kusuma, M.Eng.Sc.
Pembimbing II : Ronny Mardiyanto, S.T., M.T., Ph.D.

ABSTRAK

Robot Sepak Bola adalah salah satu bidang robotika yang mengkombinasikan kecerdasan buatan pada robot dengan permainan sepak bola. Secara ringkas, bidang ini mempelajari bagaimana robot dapat dibuat dan dilatih untuk memainkan permainan sepak bola. Bagian utama dari bidang ini adalah persepsi mesin (*Machine Perception),* perencanaan rute gerak (*path planning*), kinematika dan kontrol.

Pada tugas akhir ini telah dibahas tentang perancangan platform simulator yang dapat menjalankan fungsi-fungsi dari sistem perencanaan rute (*path planning*) dan diimplementasikan pada robot sepak bola beroda. Perencanaan rute gerak yang dilakukan adalah perencanaan rute gerak robot dari titik awal dimana robot berada menuju titik akhir yaitu titik dimana bola berada.

Hasil dari pengujian yang dilakukan pada tugas akhir ini merupakan perbandingan *trajectory* pada simulator dan pada sistem yang sebenarnya dengan jalur yang menghubungkan dari titik awal dimana robot berada dan titik target dimana bola berada dengan mempertimbangkan kemungkinan posisi halangan yang ada pada lapangan. Perbandingan *trajectory* tersebut memiliki *error RMS* sebesar 18.43 cm. Pengaruh kecepatan terhadap *trajectory* memiliki nilai yang berbanding terbalik , nilai *error* RMS terkecil didapatkan pada kecepatan 30% yaitu sebesar 4.22 cm pada sistem yang sebenarnya dan 3.89 cm pada *simulator*. Sedangkan *error* RMS terbesar didapatkan pada kecepatan 80% sebesar 15.58 cm pada sistem yang sebenarnya dan 13.52 cm pada *simulator*.

Kata kunci : simulator, robot, robot sepak bola, perencanaan rute, jalur, *platform*.

Halaman ini sengaja dikosongkan

PATH PLANNING SYSTEM IN WHEELED SOCCER ROBOT

Name **: Aulia Aditya Rachman**
Supervisor **: Dr. Ir Hendra Kusuma, M.Eng.Sc.**
Co-Supervisor **: Ronny Mardiyanto, S.T., M.T., Ph.D.**

ABSTRACT

Soccer robot is one of robotics field that combine artificial intelligence with football game. Briefly, this field is studying how the robot can be created and trained to play a football game. The main part of this field consist of Machine Perception, path planning, kinematic and dynamic control.

In this final project has been discussed about the design of simulator platform that can run the functions of the path planning system and implemented on the wheeled soccer robot. The path planning is a planning in the robot system from the starting point where the robot located towards the target point where the ball located.

The results of the tests performed on this final project is the comparison of the trajectory between the simulator and the real system from the starting point where the robot is located and the target point where the ball is located with considering the possibility of obstacles positions in the field. Trajectory comparison has an RMS error of 18.43 cm. The effect of speed on trajectory has inversed value, the smallest RMS error value obtained at 30% speed is 4.22 cm in actual system and 3.89 cm in the simulator. While the largest RMS error obtained at 80% speed of 15.58 cm on the actual system and 13.52 cm on the simulator.

Keywords : simulator, robot, soccer robot, path planning, path, platform.

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji syukur kepada Tuhan Yang Maha Esa atas nikmat rahmat dan karunia-Nya penulis dapat menyelesaikan Tugas Akhir ini dengan judul :

Sistem Perencanaan Rute Gerak pada Robot Sepak Bola Beroda

Tugas Akhir ini merupakan persyaratan dalam menyelesaikan pendidikan program Strata-Satu di Jurusan Teknik Elektro, Fakultas Teknologi Industri, Institut Teknologi Sepuluh Nopember Surabaya.

Tugas Akhir ini dibuat berdasarkan teori-teori yang didapat selama mengikuti perkuliahan, berbagai literatur penunjang dan pengarahan dosen pembimbing dari awal hingga akhir pengerjaan Tugas Akhir ini.

Pada kesempatan ini, penulis ingin berterima kasih kepada pihak-pihak yang membantu pembuatan tugas akhir ini, khususnya kepada:

1. Bapak, Ibu, kakak serta seluruh keluarga yang memberikan dukungan baik moril maupun materiil.
2. Dr. Ir. Hendra Kusuma, M.Eng.Sc selaku dosen pembimbing 1 atas bimbingan dan arahan selama penulis mengerjakan tugas akhir ini.
3. Ronny Mardiyanto, S.T., M.T., Ph.D. selaku dosen pembimbing 2 atas bimbingan dan arahan selama penulis mengerjakan tugas akhir ini.
4. Ir. Tasripan, M.T. selaku Koordinator Bidang Studi Elektronika.
5. Dr.Eng. Ardyono Priyadi, S.T., M.Eng. selaku Ketua Jurusan Teknik Elektro ITS Surabaya.
6. Seluruh dosen bidang studi elektronika.
7. Teman-teman laboratorium Elektronika yang tidak dapat disebutkan satu-persatu, telah membantu proses pengerjaan tugas akhir ini.

Penulis sadar bahwa Tugas Akhir ini belum sempurna dan masih banyak hal yang dapat diperbaiki. Saran, kritik dan masukan baik dari semua pihak sangat membantu penulis untuk pengembangan lebih lanjut.

Akhirnya penulis berharap agar Tugas Akhir ini dapat memberikan manfaat yang sebesar-besarnya bagi semua pihak serta pengembangan dalam bidang robotika. Penulis juga berharap supaya Tugas Akhir ini dapat menjadi aplikasi yang lebih bermanfaat.

Surabaya, Juli 2017

Penulis

Halaman ini sengaja dikosongkan

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Halaman ini sengaja dikosongkan

BAB I
PENDAHULUAN

1.1. Latar Belakang

Robot merupakan alat yang diciptakan sebagai alat bantu untuk membantu menyelesaikan kebutuhan manusia secara otomatis dengan program yang telah ditanamkan pada sistem robot yang telah dibuat atau dengan menggunakan kontrol yang dilakukan oleh manusia. Untuk memacu perkembangan robot di Indonesia, setiap tahunnya diadakan kompetisi robot dengan tema yang bervariasi. Salah satu kompetisi yang diadakan adalah Kontes Robot Sepak Bola Indonesia kategori beroda.

Robot Sepak Bola adalah salah satu bidang robotika yang mengkombinasikan kecerdasan buatan pada robot dengan sepak bola. Secara ringkas, bidang ini mempelajari bagaimana robot dapat dibuat dan dilatih untuk memainkan permainan sepak bola. Bagian utama dari bidang ini adalah persepsi mesin (*Machine Perception),* perencanaan rute gerak (*Path Planning*), kinematika dan kontrol. Robot mengumpulkan informasi posisi robot lain dan bola menggunakan kamera dimana kemudian gerakan yang efisien ditentukan oleh bagian *path planning*. Gerakan nyata dari robot akan diatur pada bagian kinematik dan kontrol. Pada tugas akhir ini akan dilakukan perancangan sistem perencanaan gerak robot dari titik awal menuju titik akhir yaitu titik dimana bola berada.

Pada pertandingan robot sepak bola umumnya, terdapat beberapa permasalahan yang sangat penting, Salah satu contoh masalah yang umum terjadi pada saat pertandingan antara lain adalah seringnya terjadi tabrakan antar robot baik tabrakan dengan robot lawan maupun dengan robot kawan dalam pertandingan saat melakukan pengejaran bola.

Otomatisasi dan kecerdasan buatan pada robot sepak bola dapat digunakan untuk menyelesaikan beberapa masalah tersebut, yaitu dengan cara menemukan bola dengan jalur tercepat, menghindari robot lawan, strategi yang digunakan dalam pertandingan dan koordinasi antar robot yang optimal dengan membuat sebuah perencanaan rute gerak.

Perencanaan rute gerak robot merupakan salah satu unsur yang paling penting dalam kecerdasan buatan. 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.

Umumnya, perencanaan rute gerak robot digunakan untuk memecahkan permasalahan ketika robot berada dalam area labirin. Perencanaan rute gerak juga dapat digunakan untuk menentukan strategi yang digunakan pada pertandingan.

Dalam merancang Perencanaan rute gerak robot, unit pemrosesan hanya membutuhkan informasi posisi robot, posisi bola, posisi lawan, posisi gawang terhadap lapangan. Jika posisi dari parameter-parameter tersebut dapat diketahui dalam koordinat kartesian pada lapangan, unit pemrosesan dapat mengendalikan dan mengarahkan robot menuju titik target yang dalam hal ini merupakan letak bola sekaligus dapat menghindari robot lawan ataupun halangan dan mengarahkan robot ke gawang lawan setelah mendapatkan bola.

Salah satu metode yang cukup efisien untuk menguji dan mengevaluasi algoritma perencanaan rute gerak yang telah dibuat adalah dengan menyimulasikan sistem seperti keadaan yang sebenarnya pada sebuah *simulator*. *Simulator* digunakan untuk menyimulasikan keadaan sebenarnya sebelum suatu algoritma ditanamkan kepada *platform* yang sebenarnya. Dengan cara menyimulasikan suatu algoritma, maka dapat diamati apakah algoritma tersebut sudah berjalan dengan baik ataukaH masih ada beberapa *bug* yang harus diperbaiki kembali sebelum algoritma tersebut ditanamkan pada sistem yang sebenarnya.

1.2. Perumusan Masalah

Permasalahan yang dibahas dalam tugas akhir ini adalah:

1. Perancangan *platform simulator* yang dapat menjalankan fungsi-fungsi yang ada pada robot sepakbola beroda.
2. Perancangan *path planning* dengan informasi data yang diperoleh dari sensor kamera.
3. Perancangan *path planning* yang menghasilkan keputusan berupa rute terbaik dari titik awal menuju titik akhir yang diinginkan dengan mempertimbangkan posisi robot lawan.
4. Cara memvisualisasikan perencanaan gerak yang telah dipindai.

1.3. Batasan Masalah

Batasan masalah dalam tugas akhir ini adalah:

1. *Simulator* yang dibuat hanya menjalankan fungsi-fungsi yang tertanam pada robot sepak bola beroda.

2. *Simulator* menyimulasikan satu robot sepak bola beroda.
3. *Path planning* yang dibuat adalah pada robot sepak bola beroda.
4. Sistem sudah mendapatkan informasi posisi robot, target dan *obstacle.*
5. *Path planning* yang dibuat hanya pada area dua dimensi dengan halangan statis.
6. Luasan area yang digunakan adalah berbentuk lapangan sepak bola dengan ukuran 9 m x 6 m.

1.4. Tujuan

Tujuan dari tugas akhir ini adalah sebagai berikut:

1. Simulator dapat menyimulasikan sistem seperti pada keadaan yang sesungguhnya.
2. Dapat membuat sebuah *path planning* pada robot sepak bola beroda.
3. Sistem dapat membuat *path planning* yang dapat menghindari *obstacle* atau halangan.
4. Sistem dapat menemukan jalur tercepat dari titik awal menuju titik yang diinginkan.

1.5. Metodologi

Langkah-langkah yang dikerjakan pada tugas akhir ini adalah sebagai berikut:

1. **Studi Literatur**

   Pada tahap ini dilakukan pengumpulan dasar teori yang menunjang dalam pembuatan Tugas Akhir ini. Dasar teori ini dapat diambil dari buku-buku literatur, jurnal, artikel-artikel di internet dan forum-forum diskusi internet.

2. **Perancangan *Software***

   Perancangan *software* dilakukan dengan pembuatan *source code* yang meliputi pembuatan *platform* GUI dan *simulator* yang dapat menjalankan fungsi-fungsi pada sistem yang sebenarnya, penentuan titik awal dimana robot berada, titik target dimana bola berada, arah orientasi robot menghadap ke bola dan perintah untuk membuat perencanaan rute gerak robot secara otomatis. Titik target ditentukan dengan menggunakan

algoritma perhitungan untuk mendapatkan titik koordinat robot terhadap lapangan dengan memanfaatkan parameter titik koordinat dimana robot berada, orientasi robot dan jarak bola terhadap robot. Kemudian, *source code* yang dibuat akan mengirimkan perintah untuk membuat perencanaan jalur 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-bidirectional* dengan kamera sebagai sensor yang akan mendeteksi objek tertentu, *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 algoritma yang telah dibuat dengan pembuatan simulasi pertandingan. Kedua adalah pengujian implementasi algoritma yang telah dibuat kedalam 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 disimulasikan 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

Dalam buku tugas akhir ini, pembahasan mengenai sistem yang dibuat terbagi menjadi lima bab dengan sistematika sebagai berikut:

- BAB I : PENDAHULUAN

  Pada bagian ini menjelaskan beberapa sub bagian yang antara lain berisi Latar Belakang, Permasalahan, Tujuan, Penelitian pada tugas akhir ini bertujuan untuk merancang perencanaan gerak pada robot sepak bola beroda untuk mencapai titik target yang diinginkan. Serta Metodologi, Sistematika Penulisan, dan Relevansi penulisan Tugas Akhir ini.

- BAB II : TINJAUAN PUSTAKA DAN DASAR TEORI

  Pada bagian ini berisi mengenai landasan teori yang digunakan dalam pelaksanaan tugas akhir yang meliputi robot holonomic, komputer *mini* dan STM32F4. Bagian ini memaparkan mengenai beberapa teori penunjang dan beberapa literatur yang berguna bagi pembuatan Tugas Akhir ini.

- BAB III : PERANCANGAN SISTEM

  Bab ini menjelasakan tentang perencanaan sistem baik perangkat keras (*hardware*) maupun perangkat lunak (*software*) untuk sistem perencanaan rute gerak robot sepakbola beroda.

- BAB IV : PENGUJIAN

  Pada bagian ini akan menjelaskan hasil uji coba sistem beserta analisanya.

- BAB V : PENUTUP

  Bagian ini merupakan bagian akhir yang berisikan kesimpulan yang diperoleh dari pembuatan Tugas Akhir ini, serta saran-saran untuk pengembangannya.

1.7. Relevansi

Hasil yang diharapkan dari pembuatan tugas akhir ini diharapkan dapat membuat *platform simulator* yang dapat menyimulasikan sistem seperti keadaan yang sebenarnya dengan menjalankan semua fungsi-fungsi dari robot sebelum ditanamkan pada *platform* yang sebenarnya dan juga dapat diimplementasikan pada *platform* yang sebenarnya agar dapat diaplikasikan dalam kehidupan sehari-hari untuk membantu meringankan pekerjaan manusia. Pengembangan lebih lanjut dari sistem ini adalah penambahan algoritma strategi dalam pertandingan yang melibatkan koordinasi antar robot sehingga sistem dapat menjadi lebih efisien.

BAB II
TINJAUAN PUSTAKA DAN TEORI PENUNJANG

Tinjauan pustaka dalam bab ini menjelaskan tentang sistem-sistem yang berhubungan dengan tugas akhir ini dan pernah diimplementasikan oleh penulis-penulis sebelumnya. Sedangkan bagian dasar teori menjelaskan tentang teori penunjang yang berhubungan dengan keseluruhan sistem pada tugas akhir ini.

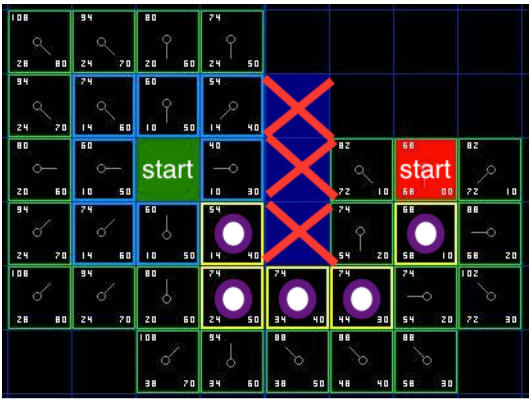
2.1. Metode Pencarian

Untuk merancang sebuar perencanaan rute gerak, tersapat banyak metode pencarian yang telah dikembangkan. Metode yang telah dikembangkan sebelumnya dibagi menjadi dua jenis, yaitu, pencarian buta/tanpa informasi (blind atau un-informed search) dan pencarian heuristik/dengan informasi (*heuristic* atau *informed search*) [1]. Setiap jenis metode pencarian memiliki karakteistik yang berbeda.

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*). Kata *Heuristic* berasal dari sebuah kata kerja Yunani, yaitu *heuriskein*, yang berarti 'mencari' atau 'menemukan'. Maka, kata heuristik dalam penggunaan metode ini dapat diartikan sebagai fungsi yang menghasilkan suatu nilai yang berupa estimasi atau biaya perkiraan dari suatu solusi permasalahan.

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* *[2]*

$$f(n) = g(n) + h(n) \tag{2.1}$$

Dimana:
f(n) : Estimasi harga terkecil untuk solusi jalur sepanjang n.
g(n) : Harga dari titik awal menuju titik ke n.
h(n) : Estimasi harga terkecil dari titik ke n menuju titik tujuan.

Algoritma A* merupakan algortima yang *optimal*. Hal ini dapat dibuktikan dengan penjelasan berikut. Dimisalkan titik *G* merupakan titik target dengan harga jalur *f** dan G_2 merupakan suboptimal *goal state*, yang berarti goal state dengan harga $g(G_2) > f^*$. Keadaan ini dapat diimajinasikan bahwa algortima A* memilih G_2 dari antrian. Karena G_2 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 G_2, maka $f(n) >= f(G_2)$. Sehingga $r > f(G_2)$. Tetapi karena G_2 merupakan *goal state*, maka $h(G_2) = 0$. Sehingga $f(G_2) = g(G_2)$. Oleh karena itu, dapat

diasumsikan bahwa $f^* >= g(G_2)$. Hal ini bertentangan dengan asumsi bahwa G_2 adalah suboptimal, sehingga dapat diketahui bawha 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 faktor 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 faktor percabangan terbatas [3].

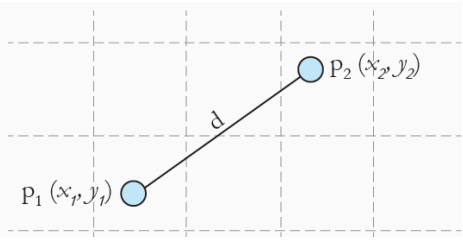
Prinsip algoritma A* adalah memperluas setiap simpul yang mungkin dari awal ke tujuan dan membandingkan biaya masing-masing jalur. Begitu ada hambatan yang membuat jalur saat ini lebih mahal daripada jalur lain yang tersedia, A* akan kembali ke jalur terendah yang baru dan mengembangnya lagi. Setelah 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 *Euclidean*

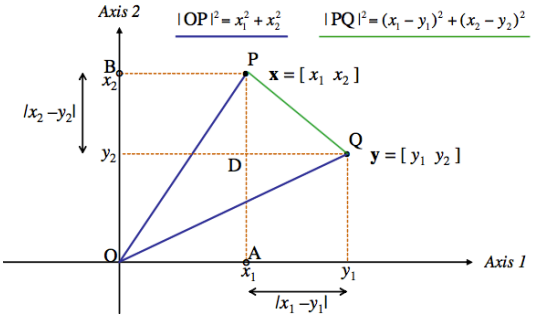
Fungsi heuristik memiliki peranan penting salam metode pencarian yang termasuk kedalam pencarian heuristik. Fungsi heuristik tersebut hanya dapat diterima apabila fungsi tersebut menghasilkan estimasi harga yang tidak melebihi harga sebenarnya. Karena ketika suatu fungsi heuristik menghasilkan estimasi 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 [3]. 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 *euclidean*



Gambar 2. 3 Teorema *pythagoras* dalam area dua dimensi

Pada gambar 2.3, dapat dijelaskan bahwa panjang kuadrat dari vektor $x = [x_1 \ \ x_2]$ merupakan penjumlahan kuadrat dari koordinat itu sendiri seperti pada segitiga OPA atau OPB pada gambar 2.3. $|OP|^2$ adalah panjang kuadrat dari koordinat x yang merupakan jarak antara titik O dan titik P. Kemudian, jarak kuadrat antara vektor $x = [x_1 \ \ x_2]$ dan $y = [y_1 \ \ y_2]$ adalah penjumlah dari selisih kuadrat dari koordinat tersebut, hal ini dapat dilihat pada segitiga PQD pada gambar 2.3. $|PQ|^2$ adalah kuadrat jarak antara titik *P* dan titik Q. Untuk mendapatkan jarak antara vektor *x* dan vektor *y* yang dapat ditulis dengan notasi $d_{x,y}$, maka dapat dirumuskan sebagai berikut:

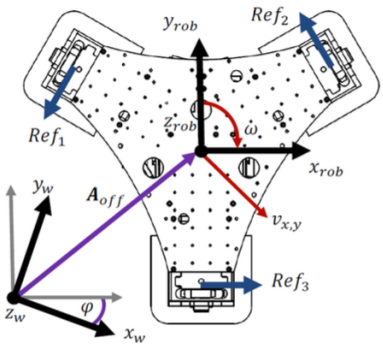
$$d_{x,y} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{2.2}$$

2.2. Robot

Robot merupakan alat yang diciptakan sebagai alat bantu untuk membantu menyelesaikan kebutuhan manusia secara otomatis dengan program yang telah ditanamkan pada sistem robot yang telah dibuat atau dengan menggunakan kontrol yang dilakukan oleh manusia. Saat ini, robot dengan akselerasi dan kecepatan yang tinggi telah banyak digunakan dalam bidang industri manufaktur untuk melakukan tugas yang berulang-ulang.

2.2.1. Robot *Holonomic*

Dalam bidang robotika, metode *holonomic* merupakan metode yang sering digunakan untuk menggambarkan ruang gerak pada robot mobile. Istilah *holonomic* memiliki penerapan yang luas terhadap beberapa bidang matematis, termasuk persamaan diferensial, fungsi dan ekspresi dari kendala. Dalam bidang robot *mobile*, istilah ini mengacu secara khusus pada batasan kinematis dari *chassis* robot. Robot *holonomic* adalah robot yang tidak memiliki batasan kinematis *nonholonomic*. Sebaliknya, robot *nonholonomic* adalah robot dengan satu atau lebih batasan kinematis *nonholonomic*. Robot *holonomic* memiliki 3 DOF terhadap bidang gerak robot menggunakan roda *omni directional* yang menghadap jauh pada pusat robot seperti pada gambar 2.4.



Gambar 2. 4 Robot *holonomic [4]*

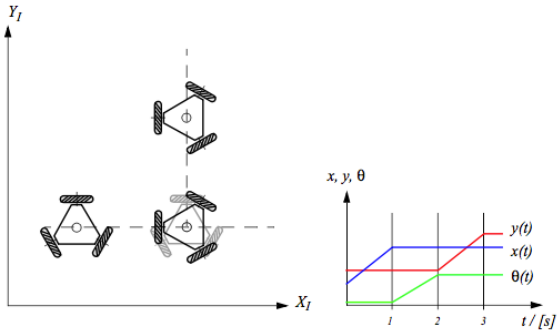
Kinematika robot *holonomic* dapat dihitung berdasarkan susunan struktur mekanis dengan menggunakan pengembangan metode kinematika *invers* dan transformasi antara koordinat bidang dan koordinat robot [4]. Implementasi dari kinematika *invers* pergerakan robot dapat dituliskan dengan persamaan berikut:

$$Ref_1 = K\left(C\omega - v_x sin30° - v_y cos30°\right) \quad (2.3)$$
$$Ref_2 = K\left(C\omega - v_x sin30° + v_y cos30°\right) \quad (2.4)$$
$$Ref_3 = K(C\omega + 2v_x) \quad (2.5)$$

2.2.2. *Trajectory* dan Penentuan Jalur

Dalam bidang robotika, kemampuan robot untuk mengikuti jalur yang telah ditentukan untuk mencapai titik target perlu dipertimbangkan. Robot harus bisa melacak jalur yang harus dilalui dalam ruang geraknya [5].

Pada robot *holonomic* 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.



(a) (b)

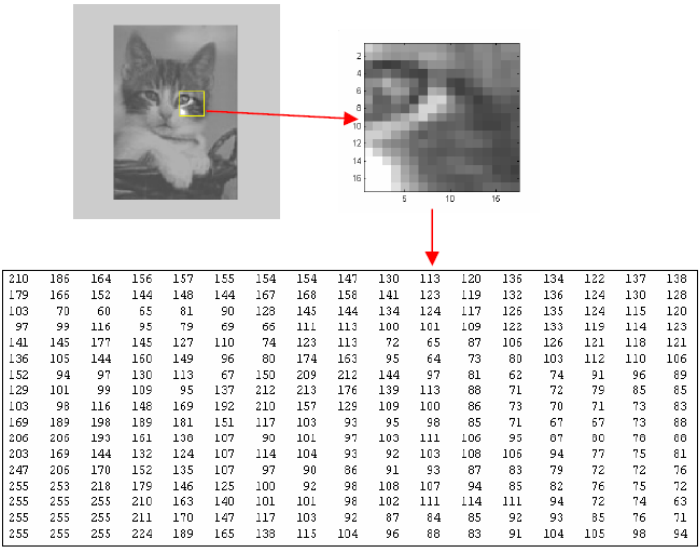
Gambar 2. 5 (a)Contoh lintasan pergerakan robot holonomic (b)Grafik kecepatan x, y, θ [5]

Contoh lintasan robot omnidirectional dapat dilihat pada gambar 2.5. 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.

2.3. Citra Digital

Citra digital merupakan hasil dari sebuah proses digitalisasi citra agar sebuah gambar dapat ditampilkan pada layar digital. Proses digitalisasi citra tersebut terdiri dari proses *sampling* dan proses *quantization*. Proses sampling merupakan proses pembagian gambar menjadi beberapa kotak kecil. Sedangkan proses *quantization* merupakan proses untuk memberikan kecerahan dari setiap kotak kecil tersebut. Proses *quantization* menghasilkan nilai kedalaman.

Citra digital merupakan sebuah matriks atau *array* yang terdiri dari nilai-nilai riil. Bagian dari matriks tersebut disebut *pixel* atau dapat juga disebut sebagai *picture element.* Sebuah *pixel* memiliki dua *property* yaitu koordinat posisi dari *pixel* dan nilai dari *pixel* itu sendiri [6]. Sehingga *pixel* tersebut dapat dinyatakan sebagai fungsi dua dimensi $f(x, y)$ yang merupakan fungsi dua dimensi dengan titik x dan y berasal dari titik sudut kiri atas gambar.

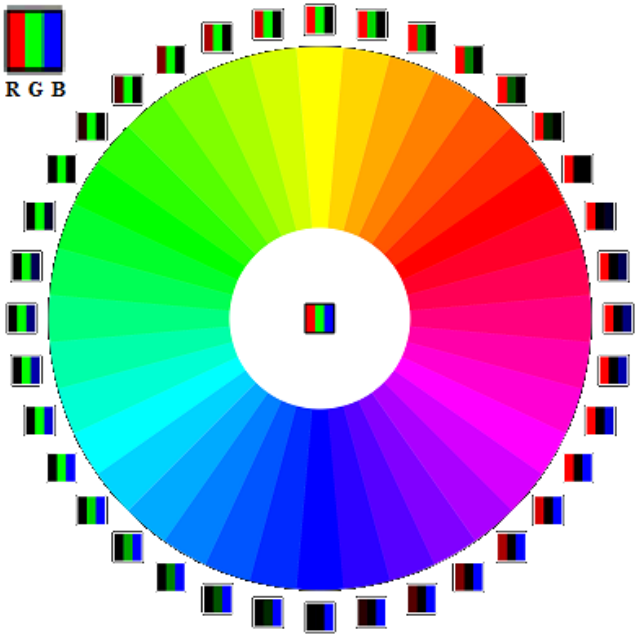


Gambar 2. 6 Konsep *pixel* dalam citra digital

Konsep dari *pixel* dalam citra digital dapat dilihat pada gambar 2.6. Jenis-jenis citra dapat dikelompokan berdasarkan kedalaman citra. Beberapa jenis citra berdasarkan kedalaman citra antara lain adalah citra warna atau sering disebut sebagai citra RGB (*Red Green Blue*), citra *grayscale*, dan citra biner.

2.3.1. Citra Warna

Citra warna merupakan citra dari setiap pixel yang nilainya ditentukan dari komponen nilai warna dasar yaitu merah, biru dan hijau [7]. Kombinasi dari ketiga warna dasar tersebut akan menghasilkan suatu warna tertentu. Nilai dari masing-masing komponen warna dasar merah, biru dan hijau memiliki rentang tertentu berdasarkan tipe data yang dipakai. Banyaknya jumlah bit akan memberikan variasi warna yang lebih banyak pula. Akan tetapi dalam pengolahan citra, semakin banyak jumlah bit, maka semakin lama proses komputasi dari pengolahan citra. Gambar 2.7 menujukan gambar dari kombinasi warna dasar merah, hijau dan biru.



Gambar 2. 7 Kombinasi warna RGB

2.3.2. Citra Grayscale

Citra graysclae merupakan citra yang nilai dari pixelnya hanya tergantung pada satu komponen yang disebut *gray level*. Citra *grayscale* disebut juga citra intensitas atau citra *gray level* [7]. *Gray level* akan menujukan tingkat kecerahan dari suatu citra. Jika *gray level* merupakan data 8 bit, akan ada 256 *gray level*, sehingga setiap *pixel* akan memiliki nilai antara 0 sampai 255. Nilai 0 merupakan warna hitam dan 255 merupakan warna putih. Diantara nilai 0 hingga 255 akan mewakili tingkat kecerahan suatu citra, semakin tinggi nilainya maka akan semakin cerah. Gambar 2.8 menunjukkan rentang *gray level* dari hitam yang bernilai 0 hingga putih yang bernilai 255. Tipe citra ini sangat sering digunakan di dalam pengolahan citra. Hal ini dikarenakan jumlah datanya yang tidak terlalu besar yang akan mempercepat proses pengolahan data, tapi data *grayscale* tidak menghilangkan informasi penting dari citra.

Gambar 2. 8 Rentang *gray level*

2.3.3. Citra Biner

Citra biner merupakan *array* logika yang bernilai 0 dan 1, yang dapat diartikan hitam dan putih [7]. Citra biner berasal dari citra *grayscale* yang di-*threshold* dengan suatu nilai tertentu untuk memberikan batas nilai citra yang dianggap 0 dan nilai citra yang dianggap 1. Dalam citra biner, suatu objek biasanya ditandai dengan warna hitam. Sedangkan warna putih menunjukkan warna latar dari suatu citra. Gambar 2.9 menunjukkan contoh citra yang telah dikonversi kedalam citra biner.

Gambar 2. 9 Citra Biner

2.4. *User Datagram Protocol*

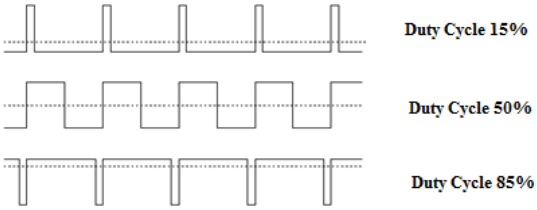
UDP (*User Datagram Protocol*) merupakan *protocol* yang sangat sederhana dengan *overhead* yang minimum [8]. Jika suatu proses perlu untuk mengirim pesan yang realatif kecil dan tidak terlalu mementingkan kehandalan, tepat jika menggunakan UDP. Pengiriman pesan kecil menggunakan UDP membutuhkan interaksi antara pengirim dan penerima lebih sedikit dibandingkan bila menggunakan TCP atau SCTP.

Paket UDP disebut *user datagram*, dengan ukuran *header* 8 *byte*, yang terdiri :

1. *Port* sumber, dengan panjang 16 bit, jika host sumber sebagai *client*, nomor *port* biasanya ditentukan oleh *software* UDP yang berjalan di host sumber, namun jika host sumber sebagai *server*, nomor *port* menggunakan *port* yang umum digunakan.
2. *Port* tujuan, panjang 16 bit, jika *host* tujuan adalah server, biasanya nomor port adalah yang biasa digunakan, jika host tujuan adalah *client*, nomor *port* adalah nomor yang disalin dari nomor *port* sementara yang diterima pada paket.
3. Panjang data, sepanjang 16 bit menyatakan panjang total *user datagram* dan *header*.
4. *Checksum*, bagian ini digunakan untuk mengetahui adanya *error* pada *user datagram*.

2.5. *Pulse Width Modulation*

Pulse Width Modulation (PWM) merupakan sinyal dengan frekuensi tetap yang memiliki panjang pulsa yang berbeda-beda setiap periodenya. Perbedaan panjang pulsa ini disebut dengan Duty Cycle atau perbandingan perubahan pulsa dengan keseluruhan periodenya. Secara sederhana, PWM merupakan sinyal dengan lebar pulsa "*HIGH*" dalam satu periode yang mewakili suatu nilai tegangan DC yang bergantung dari persenrtase *duty cycle* dari PWM tersebut. Nilai *duty cycle* dapat dihitung dari perbandingan panjangnya pulsa "*HIGH*" dengan keseluruhan periode seperti pada gambar 2.10.



Gambar 2. 10 *Duty Cycle* pada PWM

Beberapa contoh aplikasi penggunaan pulse width moduliation selain berfungsi sebagai pengkonversi daya juga dapat diaplikasikan sebagai pengatur kecerahan lampu LED, pengendali motor DC, pengendali sudut dan kecepatan servo dan lain sebagainya.

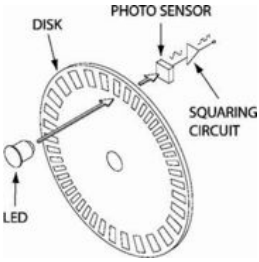
2.6. *Rotary Encoder*

Rotary Encoder merupakan suatu piranti sensor untuk mendeteksi posisi dan pergerakan. *Rotary encoder* terdiri dari sensor optikal yang berupa LED ataupun IR LED sebagai pemancar dan *phototransistor* sebagai penerima. Diantara pemancar dan penerima terdapat piringan yang berlubang dengan pola tertentu yang dapat berputar bersamaan dengan poros roda. Sehingga pada saat roda berputar, sensor akan mengahsilkan sinyal pulsa akibat dari pancaran dari LED yang terbuka dan tertutup oleh lubang piringan sehingga *phototransistor* akan hidup dan mati secara bergantian. Bentuk umum dari *rotary encoder* dapat dilihat pada gambar 2.11.

Untuk mengetahui kecepatan robot digunakan perhitungan frekuensi sebagai berikut:

$$V = \frac{S_1 - S_0}{|T_1 - T_0|} \tag{2.6}$$

Dimana :
V = Kecepatan
S_1 = Jarak tempuh sekarang.
S_0= Jarak tempuh sebelumnya.
T_1= Waktu sekarang.
T_0= Waktu sebelumnya.

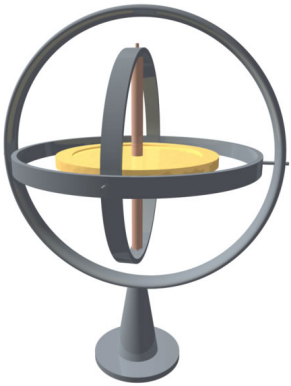


Gambar 2. 11 *Rotary encoder*

2.7. Giroskop

Giroskop adalah perangkat yang bisa mengukur kecepatan sudut. Pada awal 1700, perangkat pemintalan digunakan untuk navigasi laut dalam kondisi berkabut. Pada giroskop tahun 1800-an dan awal 1900-an telah dipatenkan untuk digunakan di kapal. Sekitar tahun 1916, giroskop ditemukan digunakan di pesawat terbang yang masih umum digunakan saat ini. Sepanjang perbaikan abad ke-20 dilakukan pada giroskop berputar [9].

Fungsi giroskop tergantung tipe dari masing-masing giroskop. Giroskop pemintalan tradisional bekerja dengan dasar bahwa benda berputar yang dimiringkan tegak lurus terhadap arah putaran akan memiliki hasil yang presisi. Hasil pembacaan yang presisi membuat perangkat tetap berorientasi pada arah vertikal sehingga sudut relatif terhadap permukaan referensi dapat diukur. Bentuk giroskop dapat dilihat pada gambar 2.12.



Gambar 2. 12 Giroskop

2.8. Komputer *mini*

Komputer *mini* atau *single board computer* adalah kelas komputer *multi-user* yang memiliki bentuk yang lebih kecil daripada komputer personal pada umumnya. Komputer mini memiliki level komputasi yang berada di posisi menengah di bawah kelas komputer mainframe dan sistem komputer *single-user* seperti komputer pribadi.

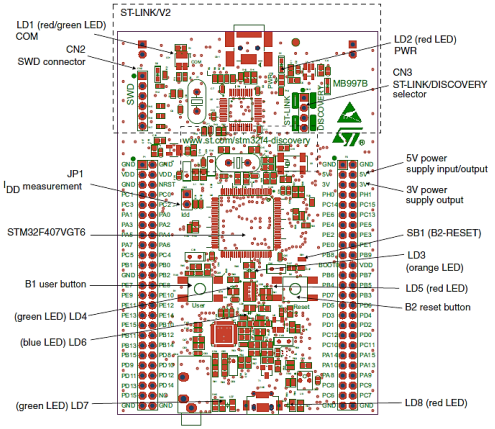
Spesifikasi atau kemampuan yang dimiliki oleh komputer mini dapat beberapa kali lebih besar jika dibanding dengan personal komputer pada umumnya. Hal ini disebabkan karena *micropocessor* yang digunakan dalam pemrosesan data memiliki kemampuan jauh lebih unggul jika dibanding dengan *micropocessor* yang digunakan pada 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, perhitungan kontrol PID dan mengendalikan pergerakan robot. Komputer mini yang dipakai adalah Intel NUC 5i7-KYK seperti pada gambar 2.13 dengan spesifikasi Prosesor Intel® Core™ i7-6770HQ (6M Cache, hingga 3.50 GHz), RAM 8 GB DDR4, M2 SSD 102GB, TDP 45 Watt, suplai sistem 19 Volt, memiliki port HDMI, 4 buah port USB dan sistem 64-bit.

Gambar 2. 13 Komputer *mini* Intel NUC 5i7-KYK

2.9. Mikrokontroler STM32F4

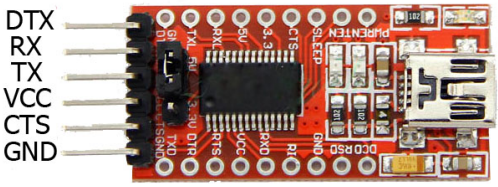
STM32F4 merupakan mikrokontroler berbasis ARM 32 bit. Mikrokontroler jenis STM32F4 menggunakan ARM core-M4F yang dapat melakukan pengolahan sinyal digital. STM32F4 memiliki fitur penambahan kecepatan clock yang lebih tinggi dari STM32F2, 64K CCM static RAM, full duplex I^2S, dan memiliki kecepatan konversi ADC. Gambar 2.14 merupakan gambar dari *Board Stm32f4-Discovery* yang sudah dilengkapi dengan *downloader* tipe ST-Link untuk memasukkan program dari komputer ke mikrokontroler STM32F4.



Gambar 2. 14 *Board* STM32F4-*Discovery [10]*

2.10. USB *to* TTL

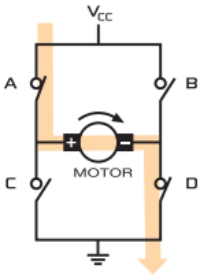
Perangkat USB *to* TTL digunakan sebagai alat komunikasi untuk mengkomunikasikan antara komputer mini dengan mikrokontroler STM32F4. Gambar 2.15 menunjukkan bentuk dari USB *to* TTL.



Gambar 2. 15 USB *to* TTL

2.11. *Driver* Motor

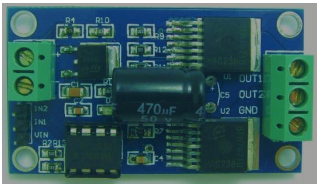
Driver motor merupakan rangkaian untuk mengendalikan kecepatan dan arah putar motor. Rangkaian ini akan memperkuat arus yang akan masuk ke motor dari unit pengontrol yang dalam hal ini merupakan mikrokontroler sehingga arus yang masuk ke motor cukup besar untuk menggerakkan motor. Rangkaian *diver* motor yang sering dipakai adalah rangkaian konfigurasi jembatan H atau biasa disebut *H-Bridge*.



Gambar 2. 16 Rangkaian *H-Bridge*

Seperti pada gambar 2.16, rangkaian dengan konfigurasi *H-Bridge* berasal dari rangkaian *full-bridge* [11]. Rangkaian ini dapat mengatur arah putaran motor. Jika saklar A dan saklar D disambungkan dan saklar B dan saklar C tidak disambungkan, arus akan mengalir dari tegangan sumber ke sisi kiri motor. Hal ini akan membuat gerakan motor searah jarum jam. Sebaliknya, jika saklar yang tersambung adalah saklar B dan saklar C, arus akan mengalir dari tegangan sumber ke sisi kanan motor. Hal ini akan membuat motor bergerak berlawanan arah jarum jam.

Dalam tugas akhir ini, *driver* motor yang digunakan adalah *driver* motor dengan modul BTN7970 sebanyak 3 buah untuk menggerakan motor. Bentuk dari modul *driver* motor BTN7970 dapat dilihat pada gambar 2.17. Pada modul driver motor ini terdapat 3 input, yaitu IN1, IN2, dan VIN. IN1 dan IN2 merupakan pengatur arah gerak motor yang menentukan kutub positif dan negatif dari OUT1 dan OUT2. Sedangkan VIN merupakan input tegangan DC antara 0-5 Volt yang berfungsi untuk mengatur kecepatan motor. Untuk pin tegangan suplai, nilai tegangan disesuaikan dengan tegangan yang dibutuhkan motor, pada tugas akhir ini tegangan motor yang dibutuhkan adalah sebesar 24 Volt.



Gambar 2. 17 *Driver* Motor BTN 7970

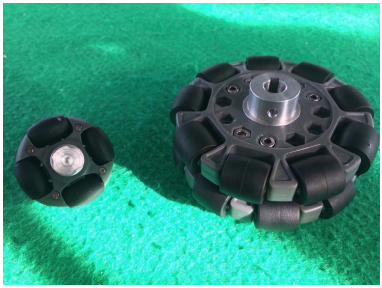
2.12. Motor DC

Motor DC merupakan perangkat elektromagnetik yang berfungsi untuk mengubah energi listrik menjadi energi mekanik. Prinsip kerja Motor DC memanfatkan hukum Lorentz. Ketika ada arus yang melewati rotor, interaksi antara magnet yang berada pada motor dan medan magnet yang ditimbulkan oleh arus di rotor menyebabkan rotor berputar. Salah satu bentuk fisik dari motor DC dapat dilihat pada gambar 2.18.

Gambar 2. 18 Motor DC

2.13. Roda *Omni directional*

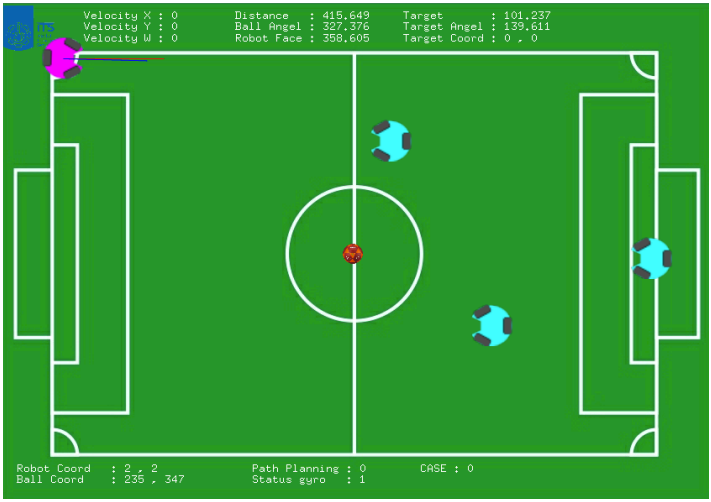
Roda *omni directional* merupakan roda yang dapat berputar ke segala arah. Hal ini disebabkan karena desain dari roda yang memiliki beberapa *roller* kecil di sekeliling roda tersebut dengan arah putar *roller* tegak lurus dengan arah putar roda. Gambar 2.19 menunjukan bentuk fisik dari roda *omni directional*. Pada pembuatan tugas akhir ini, spesifikasi roda *omni directional* yang digunakan adalah roda *omni directional* dengan diameter roda 10cm, ketebalan 3cm, marterial badan roda berbahan plastik, dan material dari *roller* berbahan karet.



Gambar 2. 19 Roda *Omnidirectional*

2.14. *Simulator*

Robot *Soccer Simulator* merupakan satu perangkat lunak yang digunakan dalam pengujian algoritma perancanaan rute gerakyang telah dibuat. *Simulator* akan mensimulasikan keadaan yang sebenarnya seperti pada gambar 2.20. Dengan berdasarkan pada *graphical user interface* (GUI), *simulator* pertandingan robot sepak bola memodelkan keadaan yang sebenarnya dari lapangan pertandingan dan posisi setiap robot. Dengan melihat simulasi yang telah ditampilkan pada *simulator*, maka akan dapat menganalisa kemungkinan-kemungkinan yang dapat terjadi dalam pertandingan. Keadaan yang terjadi pada simulasi merupakan keadaan ideal, sehingga tidak memperhatikan beberapa faktor, seperti pencahayaan, gaya gesek dan sebagainya.



Gambar 2. 20 *Simulator*

2.15. Xcode

Xcode merupakan *integrated development environment* (IDE) untuk macOS yang berisi seperangkat alat pengembangan perangkat lunak yang dikembangkan oleh Apple untuk mengembangkan perangkat lunak untuk macos, iOS, watchOS dan tvOS [12]. Logo Xcode dapat dilihat seperti pada gambar 2.21.

Gambar 2. 21 Xcode

Xcode mendukung kode sumber untuk bahasa pemrograman C, C ++, Objective-C, Objective-C ++, Java, AppleScript, Python, Ruby, ResEdit (Rez), dan Swift, dengan berbagai model pemrograman, termasuk namun tidak terbatas pada Cocoa, Karbon, dan Jawa. Pihak ketiga telah menambahkan dukungan untuk GNU Pascal, Free Pascal, Ada, C #, Perl dan D.

2.16. Visual Studio 2015

Microsoft Visual Studio merupakan *integrated development environtment* (IDE) dari Microsoft. IDE ini dapat digunakan untuk mengembangkan aplikasi antarmuka pengguna konsol dan grafis beserta aplikasi Windows Forms, situs web, aplikasi web, dan layanan web di kedua kode asli beserta kode terkelola untuk semua *platform* yang didukung oleh Microsoft Windows, Windows Phone, Windows CE,. NET Framework, .NET Compact Framework dan Microsoft Silverlight [13]. Logo visual studio dapat dilihat seperti pada gambar 2.22.

Gambar 2. 22 Visual Studio

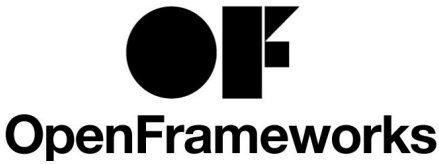
2.17. OpenFrameworks

OpenFrameworks adalah open source C ++ toolkit yang dirancang untuk membantu proses kreatif dengan menyediakan kerangka intuisi yang sederhana dan intuitif. Logo OpenFrameworks dapat dilihat seperti pada gambar 2.23.

OpenFrameworks dirancang untuk mempermudah pengguna dalam mengembangkan perangkat lunak. Openframework terdiri dari beberapa *library* yang umum digunakan, termasuk:

- OpenGL, GLEW, GLUT, libtess2 dan cairo untuk grafis
- RtAudio, PortAudio, OpenAL dan Kiss FFT atau FMOD untuk input, output dan analisis audio
- FreeType untuk font
- FreeImage untuk penghematan dan pemuatan gambar
- Quicktime, GStreamer dan videoInput untuk pemutaran video dan grabbing
- Poco untuk berbagai utilitas
- OpenCV untuk visi komputer
- Assimp untuk pemuatan model 3D

Kode ditulis secara masal *cross-compatible*. Saat ini OpenFrameworks mendukung lima sistem operasi (Windows, OSX, Linux, iOS, Android) dan empat IDE (XCode, Code :: Blocks, Visual Studio dan Eclipse). API dirancang untuk menjadi minimal dan mudah dipahami. OpenFrameworks didistribusikan di bawah Lisensi MIT [14].



Gambar 2. 23 OpenFrameworks

2.18. OpenCV

OpenCV merupakan library yang digunakan untuk mengolah data dari pengolahan citra digital atau *computer vision*. OpenCV merupakan *library* yang *open source*, artinya library tersebut dapat digunakan secara gratis baik digunakan untuk keperluan akademik ataupun komersil [15]. *Library* ini bisa digunakan dalam bahasa C, C++, Python, dan beberapa bahasa pemrograman lain. Dengan *library* OpenCV, program yang dibuat bisa digunakan untuk mengambil citra hingga mengolah citra. *Library* ini juga memungkinkan *programmer* untuk melakukan manipulasi video ataupun *real-time* video. Logo dari OpenCV dapat dilihat seperti pada Gambar 2.24.

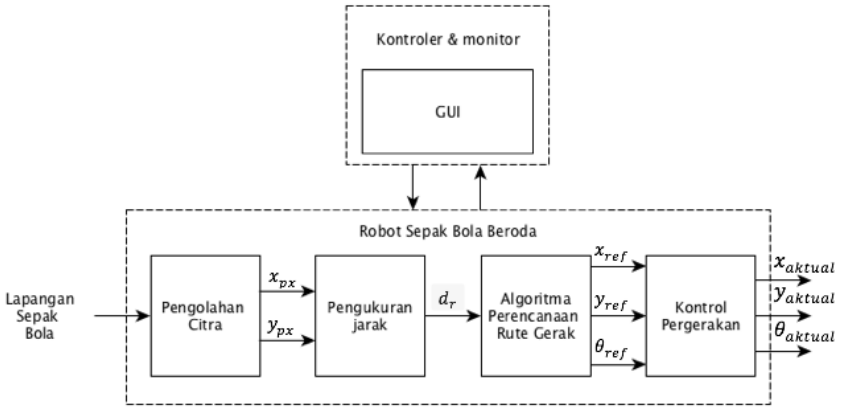
Gambar 2. 24 Logo OpenCV

Halaman ini sengaja dikosongkan

BAB III
PERANCANGAN SISTEM

Pada bab ini akan dijelaskan perancangan sistem mulai dari perancangan *software* yang meliputi perancangan algoritma, kendali pergerakan robot dan perancangan *graphical user interface* (GUI) dan *simulator* hingga perancangan *hardware* yang meliputi perancangan pembuatan bentuk fisik robot beserta pergerakan robot. Dalam bab ini juga dijelaskan tentang bagaimana proses komunikasi antara software yang telah dibuat di komputer mini pada robot yang akan mengolah data untuk perhitungan kendali robot dengan *software* yang telah dibuat di mikrokontroler yang akan mengontrol pergerakan motor pada robot.

Pada pembuatan tugas akhir ini, algoritma *path planning* digunakan untuk menentukan jalur yang harus dilalui robot dengan mempertimbangkan posisi *obstacle* dan titik target. Dengan informasi yang berupa posisi *obstacle* dan posisi target, robot yang yang telah dikendalikan oleh komputer mini dapat bergerak menuju titik target dengan menghindari *obstacle. Simulator* dibuat dengan skala 1:10 dari ukuran aslinya, yaitu 9 meter x 6 meter untuk lapangan bola, 6 centimeter untuk lebar garis, dan 60 centimeter x 60 centimeter untuk ukuran robot. Blok diagram dari sistem dapat digambarkan seperti pada gambar 3.1. Dari gambar tersebut, dapat dilihat cara kerja dari masing-masing blok.



Gambar 3. 1 Diagram blok sistem

Prinsip kerja dari sistem ini terdiri dari beberapa proses. Yang pertama adalah proses pengiriman data dari komputer user ke komputer mini dan sebaliknya untuk mengirimkan perintah ke sistem dan menampilkan posisi dari robot, bola dan *obstacle*. Komunikasi antara komputer user dan komputer mini merupakan komunikasi nirkabel menggunakan Wi-fi. Proses selanjutnya adalah pencarian posisi target dan *obstacle* yang kemudian dilakukan pengukuran jarak titik target dan *obstacle* dengan posisi robot oleh komputer *mini* pada robot. Komputer *mini* akan mengolah informasi tersebut menjadi parameter untuk menjalankan algoritma *path planning*, kemudian komputer *mini* mengirimkan data hasil pengolahan secara serial kepada mikrokontroler untuk menggerakan robot. Sehingga robot dapat bergerak pada jalur yang telah ditentukan menuju titik target dan menghindari *obstacle.*

Secara garis besar, prinsip kerja dari sistem terbagi menjadi lima sub sistem yaitu, pengolahan citra, pengukuran jarak, perancangan algoritma path planning, kontrol pergerakan dari sistem dan perancangan *Graphical User Interface*.

Sedangkan prinsip kerja dari *simulator* terdiri dari beberapa proses yaitu pehitungan algoritma path planning dan kontrol pergerakan gambar robot yang selanjutnya akan dibahas dalam sub bab perancangan *simulator*.

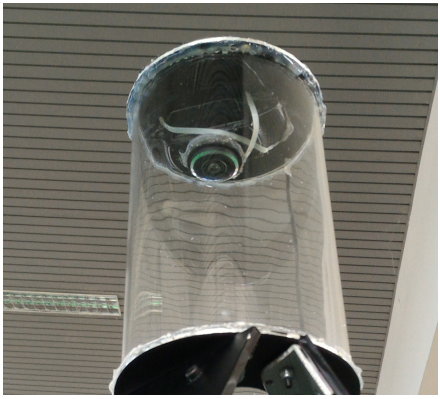
3.1. Pengolahan citra

Pengolahan citra berfungsi untuk mendapatkan informasi yang didapatkan dari citra lingkungan agar dapat diolah pada unit pemrosesan. Pada sub sistem ini, terdapat beberapa proses yaitu akuisisi citra, pendeteksian target 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 pixel target yang selanjutnya akan diolah pada sub sistem pengukuran jarak seperti pada diagram blok gambar 3.5.

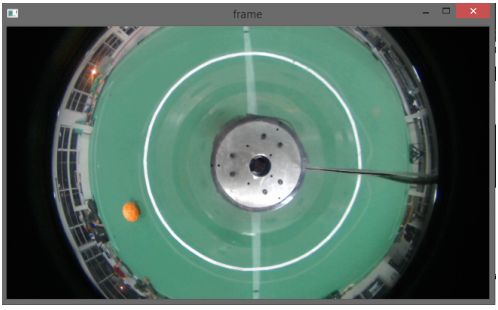
Gambar 3. 2 Diagram blok pengolahan citra

3.1.1. Pemasangan kamera pada sistem

Dalam tugas akhir ini, kamera berfungsi sebagai pendeteksi posisi objek. Kamera yang digunakan adalah kamera dengan merk Logitech C922. Kamera dipasang dengan lensa *fish-eye* 235 derajat, kemudian dipasang pada kepala robot menghadap kebawah dengan pelindung berupa tabung akrilik. Pemasangan kamera pada sistem dapat dilihat pada gambar 3.3. Citra yang dihasilkan dari pemasangan kamera dengan desain tersebut ditunjukan pada gambar 3.4.



Gambar 3. 3 Pemasangan kamera pada sistem



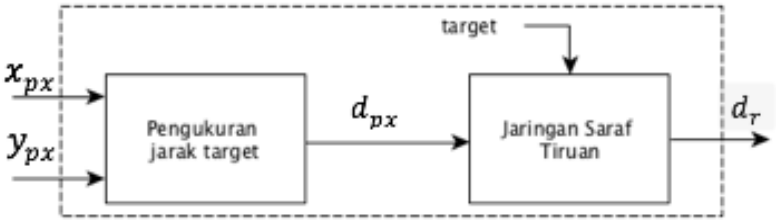
Gambar 3. 4 Citra yang dihasilkan oleh kamera

3.2. Pengukuran jarak

Pada proses sebelumnya, jarak target yang didapat belum merepresentasikan jarak yang sesungguhnya karena informasi yang diperoleh merupakan jarak pixel. Maka, pengukuran jarak dilakukan untuk mendapatkan nilai jarak yang sesungguhnya. Dalam hal ini, terdapat dua proses, yaitu mengukur jarak target yang didapatkan oleh jarak antar koordinat robot dengan koordinat target dalam pixel dengan menggunakan rumus sebagai berikut:

$$r_p = \sqrt{(x_o - x_c)^2 + (y_o - y_c)^2} \tag{3.1}$$

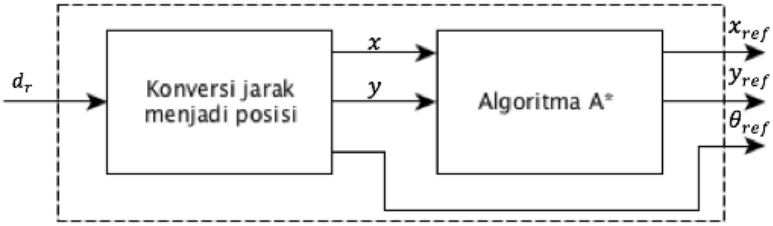
Kemudian, hasil dari jarak dalam pixel diolah untuk mendapatkan jarak yang sesungguhnya dalam satuan centimeter. Metode yang digunakan untuk mendapatkan jarak yang sesungguhnya adalah dengan menggunakan metode jaringan saraf tiruan. Proses-proses pada sub sistem pengukuran jarak digambarkan dalam bentuk diagram blok seperti pada gambar 3.5.



Gambar 3. 5 Diagram blok pengukuran jarak

3.3. Perancangan algoritma perencanaan rute (*path planning*)

Algoritma *path planning* dirancang untuk memberikan keputusan jalur 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 3.6. Dalam hal ini algoritma *path planning* yang digunakan adalah algoritma A* (A *Star*).



Gambar 3. 6 Diagram blok algoritma perencanaan rute

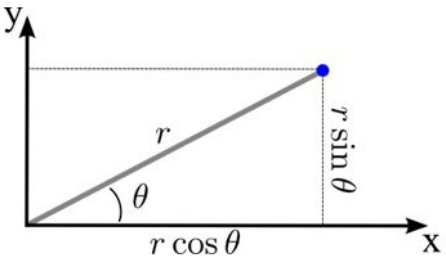
3.3.1. Perhitungan konversi jarak menjadi koordinat

Perhitungan konversi jarak menjadi titik koordinat digunakan untuk mengonversi jarak yang terukur dalam satuan centimeter kedalam 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 3.7.

Dari grafik gambar 3.7, dapat diperoleh perumusan untuk mencari koordinat x dan koordinat y sebagai berikut:

$$x = r\cos\theta \quad (3.2)$$

$$y = r\sin\theta \quad (3.3)$$



Gambar 3. 7 Grafik hubungan antara koordinat kartesian dengan koordinat kutub

3.3.2. Perancangan algoritma A*

Algoritma A* merupakan algoritma pencarian jalur dari satu titik awal yang menuju titik akhir yang telah ditentukan dalam sebuah area bebas beraturan. Persamaan untuk mendapatkan nilai estimasi harga terkecil untuk solusi jalur dapat dirumuskan sebagai berikut:

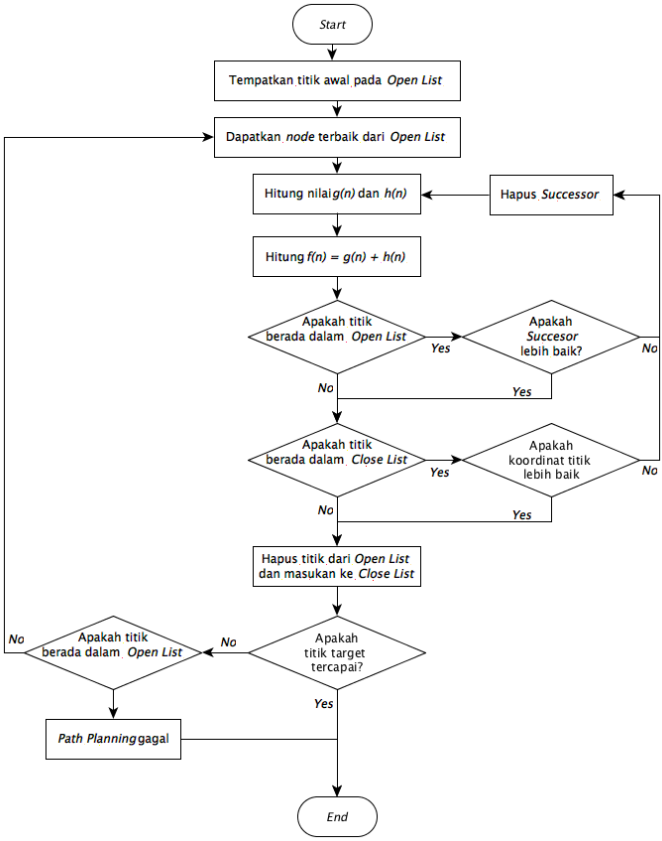
$$f(n) = g(n) + h(n) \tag{3.4}$$

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

$$d_{x,y} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3.5}$$

Gambar 3.8 menggambarkan diagram alir dari algoritma A*. Langkah-langkah dalam algoritma A* adalah sebagai berikut:

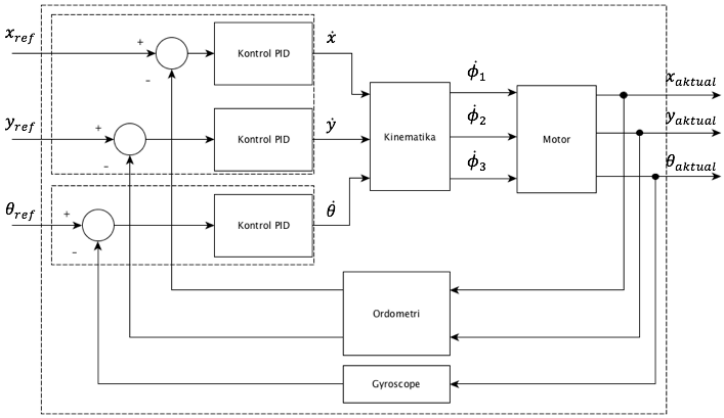
1. Tempatkan titik awal n_s pada *Open List*.
2. Jika *Open Lis*t kosong, maka proses gagal.
3. Kemudian keluarkan titik awal dari *Open List* dan tempatkan pada *Closed List* sebuah titik n_i yang memiliki fungsi *cost* $f(n_i)$ minimum.
4. Jika n_i merupakan titik akhir n_g, maka proses berhasil dengan 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 $f(n_{i+1})$ 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 jalur yang merupakan hasil terendah $g(n_{i+1})$.
6. Kembali ke langkah 2.



Gambar 3. 8 Diagram alir algoritma A*

3.4. Kontrol Pergerakan

Kontrol pergerakan digunakan untuk mengatur *duty cycle* PWM yang berfungsi untuk mengatur kecepatan masing-masing motor untuk menghasilkan nilai kecepatan $vx, vy,$ dan ω. *Rotary encoder* akan memberikan feedback posisi motor. Diagram blok dari kontrol pergerakan dapat dilihat dari gambar 3.9.



Gambar 3. 9 Blok diagram kontrol pergerakan

Berikut adalah *listing* program fungsi kontrol pergerkan dari robot:

```
bool ofApp::vxvy_control(float _x, float _y, float _x1,
float _y1, int &_vx, int &_vy, int _v, float akurasi)
{
    static float error;
    static float error_before;
    gyro_status = 1;
    error = pythagoras(_x, _y, _x1, _y1);

    p_vxvy_control = error * kp_vxvy_control;
    i_vxvy_control = i_vxvy_control + error *
ki_vxvy_control;
    d_vxvy_control = (error - error_before) *
kd_vxvy_control;

    pid_vxvy_control = p_vxvy_control + i_vxvy_control +
d_vxvy_control;

    if (pid_vxvy_control > _v) pid_vxvy_control = _v;
    else if (pid_vxvy_control < -_v) pid_vxvy_control = -_v;

    error_before = error;

    _vx = (int)((_x1 - _x) * pid_vxvy_control / error);
    _vy = (int)((_y1 - _y) * pid_vxvy_control / error);

    if (abs(error) < akurasi) return true;
    else return false;
}
```

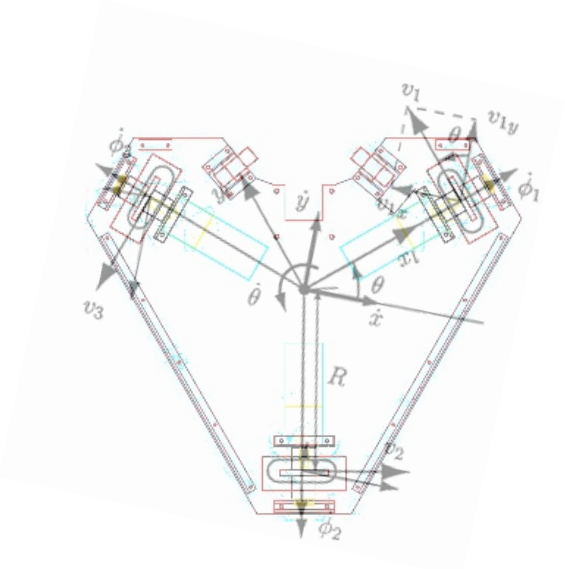
```
bool ofApp::w_control(float _w, float _w1, int &_vw, int _v,
int toleransi)
{
    static float error;
    static float error_before;
    error = _w1 - _w;

    if (error > 180) error = error - 360;
    else if (error < -180) error = error + 360;

    p_w_control = error * kp_w_control;
    i_w_control = i_w_control + error * ki_w_control;
    d_w_control = (error - error_before) * kd_w_control;

    pid_w_control = p_w_control + i_w_control + d_w_control;

    if (pid_w_control > _v) pid_w_control = _v;
    else if (pid_w_control < -_v) pid_w_control = -_v;

    error_before = error;

    _vw = (int)pid_w_control;

    if (abs(error) <= toleransi) return true;
    else return false;
}
```

3.4.1. Perancangan sistem penggerak robot

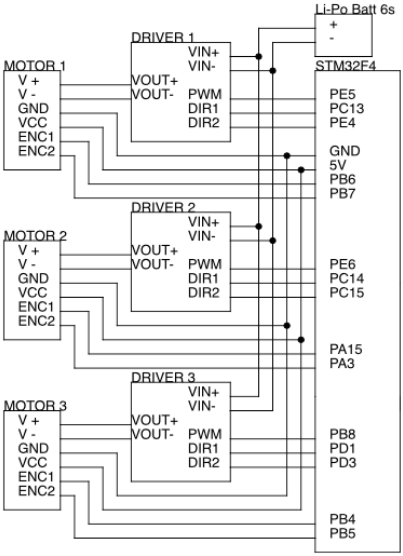
Motor yang digunakan sebagai penggerak sistem pada tugas akhir ini adalah motor DC PG-45 dengan spesifikasi suplai tegangan 24 Volt dengan kecepatan 400rpm. Motor ini dilengkapi dengan *gearbox planetary* dengan perbandingan 1:19. Arah pergerakan motor ditentukan dengan pemberan tegangan DC pada kedua input motor. Tiga buah motor DC dipasang membentuk segitiga dengan masing-masing sudut peletakan motor DC adalah 120 derajat seperti ditunjukan pada gambar 3.10. Gambar 3.11 menunjukan interkoneksi dari pin pada motor yang terhubung ke driver motor dan STM32F4.

Dengan desain pemasangan motor seperti pada gambar 3.10, maka kinematika dari sistem dapat dirumuskan sebagai berikut:

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



Gambar 3. 10 Desain pemasangan motor pada sistem



Gambar 3. 11 Interkoneksi pin motor dengan *driver* motor dan STM32F4

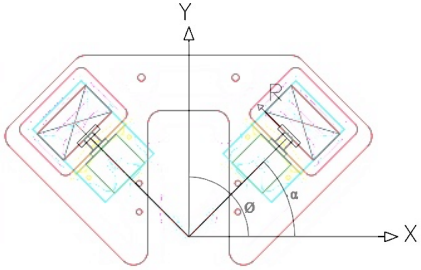
3.4.2. Perancangan sistem ordometri

Dalam tugas akhir ini, rotary encoder digunakan sebagai *feedback* posisi dari robot. *Rotary encoder* yang digunakan memiliki spesifikasi suplai 5-24VDC ± 5% dan 200 pulsa per rotasi.

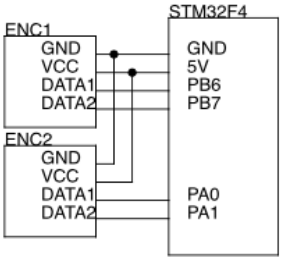
Gambar 3.12 menunjukan perancangan dua buah rotary encoder yang akan dipasang pada sistem. Sedangkan, Gambar 3.13 menunjukan interkoneksi dari pin pada rotary encoder yang terhubung ke pin STM32F4.

Dengan desain pemasangan *rotary encoder* seperti pada gambar 3.12, maka persamaan ordometri dari sistem dapat dirumuskan sebagai berikut:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = \begin{bmatrix}-\cos(225° + \theta) & -\cos(135° + \theta)\\ -\sin(225° + \theta) & -\sin(135° + \theta)\end{bmatrix}\begin{bmatrix}(\omega R_{enc1})\\ (\omega R_{enc2})\end{bmatrix} \quad (3.7)$$



Gambar 3. 12 Desain pemasangan *rotary encoder* pada sistem



Gambar 3. 13 Interkoneksi pin *rotary encoder* dengan STM32F4

3.4.3. Pembangkitan PWM

Pembangkitan PWM dalam mikrokontroler STM32F4 memanfaatkan *timer/counter*. Sehingga, timer pada pin yang akan dijadikan sebagai keluaran untuk membangkitkan PWM harus diaktifkan dan pin harus diset sebagai AF (*Alternate Function*). Frekuensi dari *timer* yang diaktifkan diperngaruhi oleh nilai TIM Period dan TIM Prescaler dengan perumusan sebagai berikut:

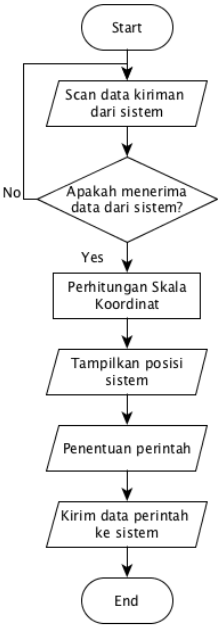
$$\frac{1}{f} = \frac{(TIM_{Period} + 1)(TIM_{Prescaler} + 1)}{\frac{Clock_{System}}{APBx\ Prescaler} x2} \tag{3.8}$$

Untuk mengatur *duty_cycle* dari PWM yang dihasilkan bisa dengan cara mengatur variabel TIMx -> CCRy dengan x merupakan indeks *timer* dan y merupakan *indeks* kanal keluaran PWM.

3.5. Perancangan GUI

Dalam tugas akhir ini, GUI berfungsi untuk memvisualisasikan pergerakan robot sehingga dapat dengan mudah dalam memonitor pergerakan robot Beberapa fitur yang dibuat pada GUI adalah dapat mengkomunikasikan robot dengan *referee box* atau *game controller,* GUI juga dilengkapi beberapa menu yang berfungsi untuk mengirimkan instruksi kepada robot. Menu-menu yang terdapat pada GUI berupa beberapa perintah yang mirip dengan perintah pada *referee box* atau *game controller*, perintah pergerakan robot dan beberapa perintah untuk pengkalibrasian posisi robot. GUI juga dapat merekam *trajectory* pergerakan yang dibuat oleh pergerkan robot. GUI dibuat pada komputer *user* di luar sistem dengan menggunakan komunikasi Wi-fi dengan protokol UDP sebagai pengiriman dan penerimaan data.

Prinsip kerja dari GUI yang dirancang digambarkan kedalam bentuk diagram alir seperti pada gambar 3.14. Prinsip kerja dari GUI yang dirancang adalah menampilkan koordinat posisi robot, bola dan *obstacle* dengan data yang diterima dari sistem dengan skala 1:133 cm atau sekitar 75% dari skala asli. GUI juga dapat menampilkan *trajectory* yang dibuat oleh pergerakan robot. Tombol-tombol yang tertampil pada GUI digunakan untuk mengirimkan data perintah kepada sistem. Gambar 3.15 menunjukan tampilan GUI dari sistem yang dibuat.

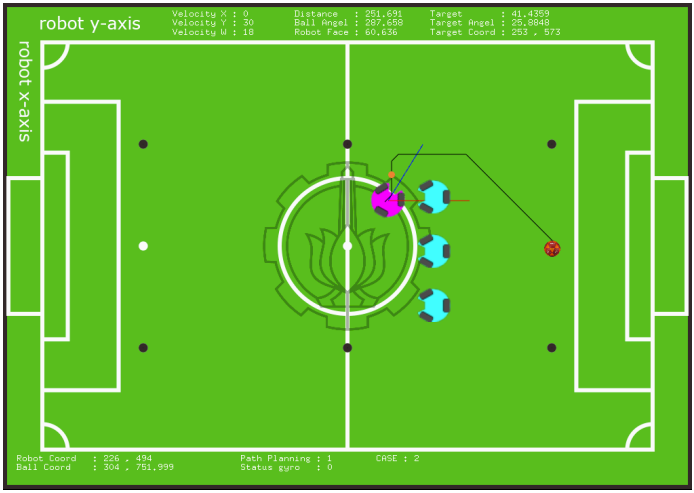


Gambar 3. 14 Diagram alir prinsip kerja GUI

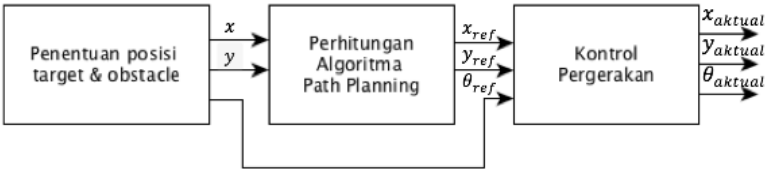
Gambar 3. 15 Tampilan *Graphical User Interface*

3.6. Perancangan *Simulator*

Simulator berfungsi untuk menyimulasikan pergerakan robot dalam keadaan yang ideal dengan harapan mendekati keadaan yang sebenarnya. Dalam *Simulator* seperti yang terlihat pada gambar 3.16, posisi robot ditunjukan pada gambar robot berwarna merah magenta. Gambar robot berwara biru menunjukan posisi obstacle. Bola digambarkan berwarna oranye. Garis berwarna merah menunjukan arah dari muka robot, sedangkan garis berwarna biru menunjukan arah depan robot terhadap koordinat lapangan. Garis berwarna hitam merupan jalur yang harus dilalui robot menuju titik target yang dalam hal ini berupa posisi bola. Proses-proses pada *simulator* digambarkan dalam blok diagram seperti pada gambar 3.17.



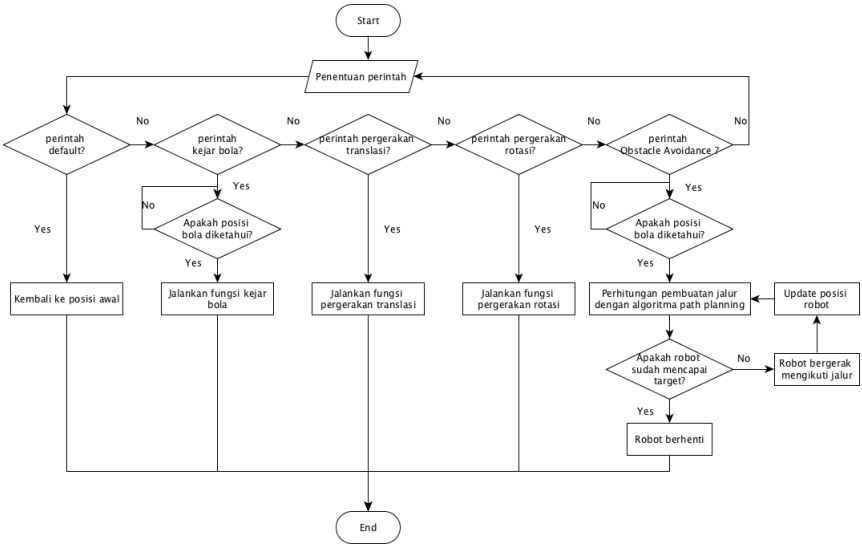
Gambar 3. 16 Tampilan *Simulator*



Gambar 3. 17 Diagram blok *Simulator*

Prinsip kerja dari *simulator* yang dirancang adalah dengan menampilkan pergerakan dari gambar sistem berdasarkan kepada fungsi kontrol pergerakan yang tertanam pada sistem yang sebenarnya. Fungsi-fungsi tersebut disesuaikan agar dapat dijalankan pada *simulator*. *Simulator* terdiri dari beberapa pilihan menu pergerakan, yaitu kejar bola, pererakan translasi, pergerakan rotasi dan *obstacle avoidance.* Prinsip kerja dari *simulator* digambarkan dalam bentuk diagram blok seperti pada gambar 3.26. Proses yang terjadi dalam *simulator* hampir sama dengan proses yang terjadi pada sistem, perbedaannya adalah jika pada *simulator*, peletakan posisi robot, target dan *obstacle* diposisikan secara *manual*. Diagram alir dari simulator digambarkan seperti pada gambar 3.18.

Dalam simulasi, penggambaran dan pergerakan sistem dibuat berdasarkan titik koordinat. Sedangkan pada sistem yang sebenarnya, informasi yang diterima oleh sistem dari sensor adalah berupa jarak riil antara posisi sistem dengan posisi target dan posisi *obstacle.* Maka, harus dibuat persamaan rumus untuk mengonversi dari data koordinat menjadi data jarak sehingga fungsi-fungsi yang tertanam pada sistem dapat diolah dalam simulasi.



Gambar 3. 18 Diagram alir *Simulator*

Gambar 3.19 menunjukan parameter-parameter antara koordinat posisi robot dengan posisi bola, dimana (x, y) merupakan koordinat pusat robot, γ adalah sudut rotasi dari robot, ϕ adalah sudut antara vektor depan robot, adalah v kecepatan pergerakan maju pada robot dan ω adalah kecepatan rotasi dari robot.

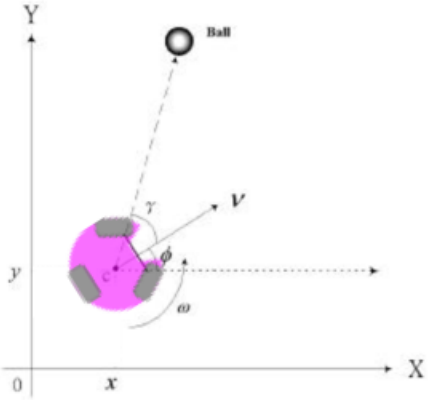
Dari data-data tersebut, maka persamaan dinamika gerak robot dapat dirumuskan sebagai berikut:

- Pergerakan robot dengan mode *gyro*:

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\phi}(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.9}$$

- Pergerakan robot tanpa mode *gyro*:

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\phi}(t) \end{bmatrix} = \begin{bmatrix} \sin\phi & 0 \\ \cos\phi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.10}$$



Gambar 3. 19 Koordinat posisi robot

Ada dua macam pergerakan pada robot yaitu pergerakan dengan mode *gyro* dan pergerakan tanpa mode *gyro*.

Berikut adalah *listing* program yang menunjukan implementasi dari persamaan rumus dari dinamika gerak robot:

```
speedY = (float)temp_vy * 0.093;
speedX = (float)temp_vx * 0.093;
rotationSpeed = (float)temp_vw *0.093;

if(gyro_status == TRUE){
   robotX+=speedX;
   robotY+=speedY;
}

else{
   robotX += - speedY * sin((rotation + temp_o) * PI / 180)
+ speedX * cos((rotation + temp_o) * PI / 180);
   robotY += speedY * cos((rotation + temp_o) * PI / 180) +
speedX * sin((rotation + temp_o) * PI / 180);
}

   rotation += rotationSpeed;

if (rotation >=360){
   rotation = 0;
}

if (rotation < 0 ){
   rotation = 360;
}
```

- Mencari jarak antar titik menggunakan rumus Pythagoras:

$$d_{x,y} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3.11}$$

- Mencari sudut dari jarak yang diketahui:

$$\gamma = \tan^{-1}\left(-\frac{x_2 - x_1}{y_2 - y_1}\right) \tag{3.12}$$

Berikut adalah *listing* program fungsi yang menunjukan implementasi dari persamaan rumus pythagoras untuk mencari jarak antara dua titik koordinat dan fungsi untuk mengubah data koordinat

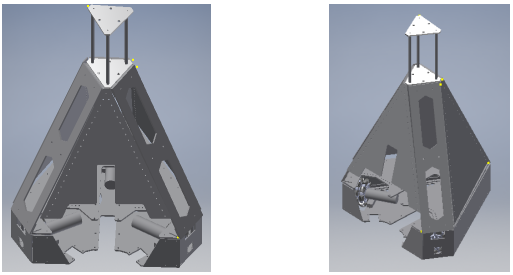
menjadi data sudut untuk menghasilkan nilai sudut dari arah hadap robot dengan suatu titik tertentu:

```
float ofApp::pythagoras(float _x, float _y, float _x1, float
_y1)
{
    return sqrt(pow((_x1 - _x), 2) + pow((_y1 - _y), 2));
}
```

```
float ofApp::point2angle(float _x, float _y, float _x1, float
_y1)
{
    float angle = atan2f(-(_x1 - _x), (_y1 - _y));
    angle *= 57.295779;

    if (angle < 0) angle = 360 + angle;

    return angle;
}
```

3.7. Desain sistem

Sistem didesain dengan bahan dasar alumunium padat dengan ketebalah 3mm dan 5mm untuk bagian kerangka badan robot dan akrilik untuk bagian penyangga sistem kelistrikan pada robot. Robot didesain dengan menggunakan penggerak 3WD dengan Motor DC yang dilengkapi dengan *encoder* dan roda *omni directional*. Desain dari sistem ditunjukan pada gambar 3.20.



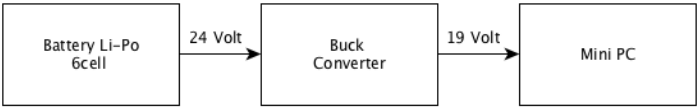
(a) (b)

Gambar 3. 20 Desain sistem (a) tampak depan (b) tampak samping

3.8. Perancangan sistem elektronik

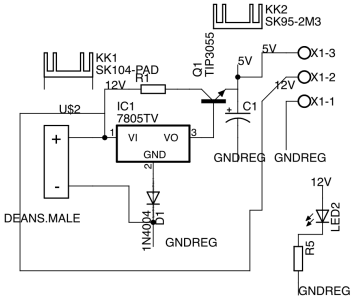
3.8.1. Manajemen *Power Supply*

Pemilihan spesifikasi dari *power supply* untuk sistem merupakan salah satu hal yang penting. Pada tugas akhir ini, spesifikasi dari *power supply* untuk sistem adalah baterai Li-Po 6cell 5000mAh untuk mensuplai bagian komputer mini dengan spesifikasi tegangan masukan 19 Volt. Oleh karena itu, diperlukan regulasi tegangan menjadi 19 Volt dengan menggunakan modul *Buck Converter* seperti pada diagram blok gambar 3.21. Kemudian, *power supply* untuk mensuplai tiga buah motor penggerak dan 2 buah motor untuk bagian penggiring dengan spesifikasi tegangan masukan sebesar 24 Volt adalah baterai Li-Po 6cell 5000 mAh. Sedangkan, *power supply* yang digunakan untuk sistem elektronik pada robot adalah baterai Li-Po 3cell 2000mAh. Pada bagian sistem elektronik, sumber yang dibutuhkan ada dua macam, yaitu 5 Volt untuk tegangan masukan mikrokontroler dan 12 Volt untuk tegangan masukan dari beberapa sensor yang digunakan. Sehingga diperlukan regulasi tegangan mejadi 5 Volt pada bagian mikrokontroler seperti pada diagram blok gambar 3.22. Sedangkan untuk bagian sensor yang memerlukan tegangan masukan sebesar 12 Volt didapatkan langsung dari baterai. Skematik rangkaian regulator yang digunakan dapat dilihat pada gambar 3.8 dan *board* PCB rangkaian regulator dapat dilihat pada gambar 3.9.

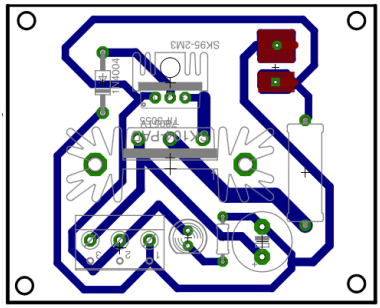


Gambar 3. 21 Diagram blok sistem suplai mini PC

Gambar 3. 22 Diagram blok sistem suplai sistem minimum



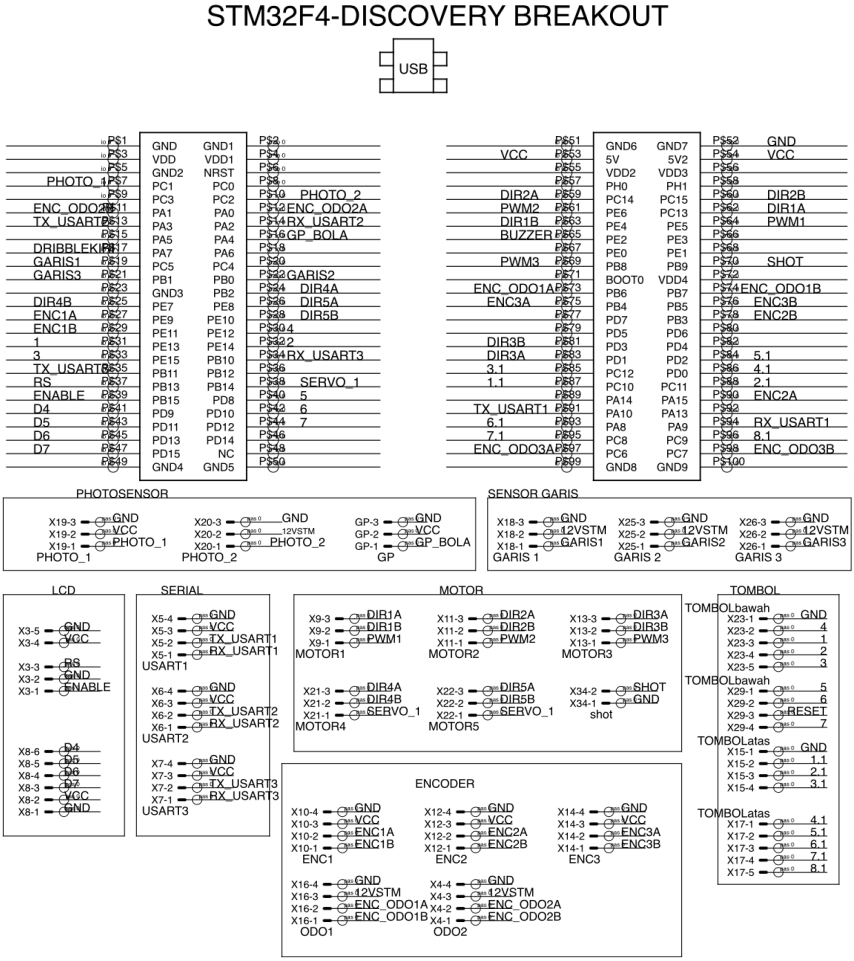
Gambar 3. 23 Skematik rangkaian regulator 5 Volt



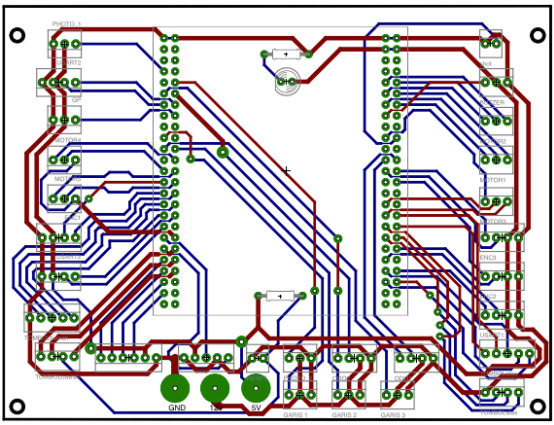
Gambar 3. 24 *Board* rangkaian regulator 5 volt

3.8.2. Sistem minimum *master* dengan STM32F4

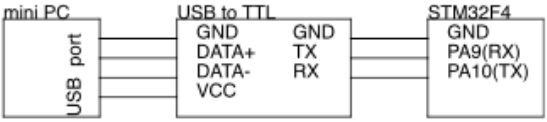
Rangkaian sistem minimum *master* dengan menggunakan mikrokontroler STM32F4 Discovery berfungsi untuk mengolah data dari sensor-sensor, menggerakan aktuator dan mengontrol sistem. Sistem minimum *master* ini juga menerima dan mengirimkan data hasil pengolahan kepada komputer mini secara *serial*. Skematik dari rangkaian *master* digambarkan seperti pada gambar 3.25. Untuk desain *board* PCB dari sistem minimum master digambarkan seperti pada gambar 3.26. Sedangkan, gambar 3.27 menunjukan interkoneksi antara komputer mini dengan USB to TTL untuk mengkomunikasikan PC dengan STM32F4.



Gambar 3. 25 Skematik sistem minimum *master*



Gambar 3. 26 *Board* sistem minimum *master*

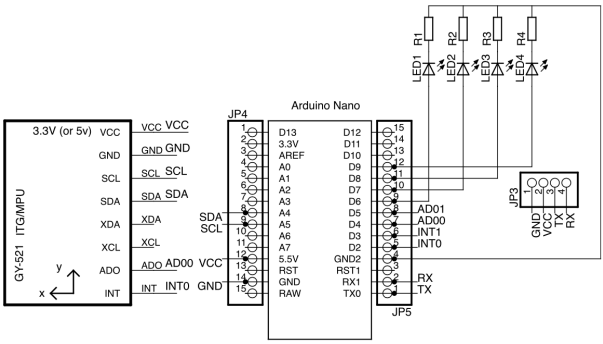


Gambar 3. 27 Interkoneksi Komputer *mini* dengan STM32F4 melalui USB *to* TTL

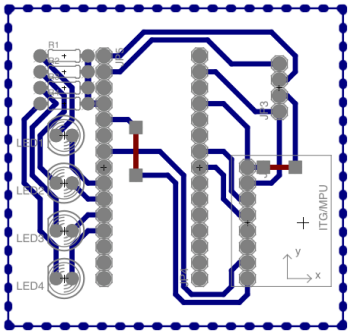
3.8.3. Sistem minimum *slave* dengan Arduino Nano

Rangkaian sistem minimum *slave* dengan Arduino nano yang berfungsi sebagai *processing unit* untuk mengakses sensor *gyro* dengan komunikasi I2C yang kemudian data hasil pembacaannya dikirimkan ke STM32F4. Arduino nano memiliki pin SDA dan SCL untuk mengakses perangkat dengan komunikasi I2C dan Pin TX dan RX untuk media komunikasi serial dengan *unit processing* lain. Rangkaian sistem minimum slave digambarkan seperti pada gambar 3.28. Untuk desain board pcb dari sistem minimum master digambarkan seperti pada gambar 3.29.

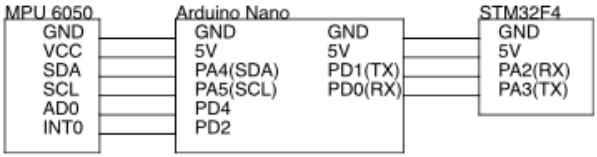
Gambar 3.30 menunjukan interkoneksi antara sensor MPU 6050 dengan Arduino nano kemudian dikomunikasikan dengan STM32F4.



Gambar 3. 28 Skematik sistem minimum slave



Gambar 3. 29 *Board* sistem minimum *slave*



Gambar 3. 30 Interkoneksi MPU-6050 dengan Arduino nano dan STM32F4

Halaman ini sengaja dikosongkan

BAB IV
PENGUJIAN

Pada bab ini akan dibahas mengenai pengujian dari sistem yang telah dirancang. Bab ini bertujuan untuk mengetahui apakah tujuan dalam perancangan sistem pada tugas akhir ini telah terlaksana atau tidak. Pengujian pada bab ini terdiri dari pengujian kecepatan motor, kontrol pergerakan robot, pengujian algoritma *path palnning* pada *simulator* dan pada *platfrorm* yang asli. Untuk pengujian algoritma *path planning,* pengujian dilakukan dengan memberikan beberapa skenario posisi robot, bola dan *obstacle* yang berbeda untuk mengetahui apakah algoritma dapat berjalan dengan baik dengan situasi tersebut.

4.1. Pengujian kecepatan motor

Pengujian kecepatan motor dilakukan untuk mengetahui perbandingan antara kecepatan putar motor terhadap masukan PWM dari mikrokontroler. Pengujian ini dilakukan dengan memberikan tegangan suplai 24 Volt DC, kemudian masukan PWM diberikan kepada driver motor dan mengubah nilai dari persentasi *duty cycle* PWM setiap 10% sehingga dapat diamati perubahan kecepatan putar motor terhadap *duty cycle* PWM yang diberikan. Tabel 4.1 merupakan data persentase *duty cycle* PWM dan kecepatan putar motor.

Tabel 4. 1 Hasil pengukuran kecepatan putar motor dengan masukan persentase *duty cycle* dari PWM.

No.	Persentase *duty cycle* (%)	Kecepatan putar motor (rpm)
1	10	36
2	20	76
3	30	112
4	40	157
5	50	193
6	60	229
7	70	270
8	80	310
9	90	351
10	100	387

4.2. Pengujian *rotary encoder*

Pengujian *rotary encoder* dilakukan untuk mengetahui koordinat posisi dari sistem. Pengujian ini dilakukan dengan menggerakan sistem kepada titik koordinat tertentu. Kemudian hasil pembacaan dari *rotary encoder* dibandingkan dengan titik koordinat yang sebenarnya. Pengukuran dilakukan setiap 100 cm dari mulai 0 cm hingga 900 cm. Pada pengujian ini, arah gerak sistem adalah maju, kanan dan kiri. Berikut adalah hasil dari pengujian yang disajikan dalam bentuk table 4.2, 4.3 dan 4.4.

Tabel 4. 2 Pengujian *rotary encoder* pergerakan arah maju

No.	Target		Terukur		RMSE (cm)
	x (cm)	y (cm)	x (cm)	y (cm)	
1	0	100	0	101	1
2	0	200	0	200	0
3	0	300	-1	300	1
4	0	400	-3	398	3.60
5	0	500	-4	498	4.47
6	0	600	-6	597	6.70
7	0	700	-8	697	8.54
8	0	800	-11	796	11.70
9	0	900	-13	896	13.60

Tabel 4. 3 Pengujian *rotary encoder* pergerakan arah kanan

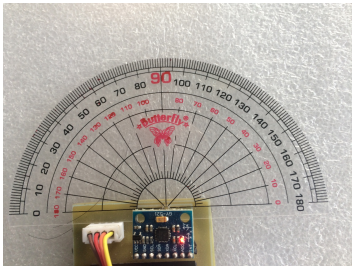
No.	Target		Terukur		RMSE (cm)
	x (cm)	y (cm)	x (cm)	y (cm)	
1	100	0	101	4	4.12
2	200	0	202	9	9.22
3	300	0	304	13	13.60
4	400	0	405	19	19.65
5	500	0	506	24	24.74
6	600	0	606	31	31.58
7	700	0	706	38	38.47
8	800	0	809	47	47.85
9	900	0	910	55	55.90

Tabel 4. 4 Pengujian *rotary encoder* pergerakan arah kiri

No.	Target		Terukur		RMSE (cm)
	x (cm)	y (cm)	x (cm)	y (cm)	
1	-100	0	-101	-4	4.12
2	-200	0	-202	-9	9.22
3	-300	0	-304	-13	13.60
4	-400	0	-405	-18	18.68
5	-500	0	-506	-23	23.77
6	-600	0	-607	-29	29.83
7	-700	0	-707	-35	35.69
8	-800	0	-809	-41	41.98
9	-900	0	-909	-48	48.84

4.3. Pengujian sensor *gyro*

Pengujian sensor gyro dilakukan untuk mengetahui arah dari hadap dari sistem. Pengujian ini dilakukan dengan memutar sensor gyro dengan sudut tertentu kemudian mengamati nilai hasil keluaran dari sensor gyro sehingga dapat diketahui perubahan nilai keluaran dari sensor gyro terhadap perubahan sudut yang sebenarnya. Sensor gyro dikalibrasi pada sudut 90 derajat, maka nilai keluaran dari sensor gyro bernilai mendekati 0 pada sudut 90 derajat seperti pada gambar 4.1. Tabel 4.5 merupakan data nilai hasil keluaran gyro dengan perubahan sudut yang sebenarnya yang digambarkan dalam bentuk grafik.



Gambar 4. 1 Pengujian sensor *gyro*

Tabel 4. 5 Data nilai hasil keluaran sensor *gyro*

No.	Sudut (^{o})	Nilai keluaran sensor *gyro*	Nilai *error* (%)
1	0	-89.2	0.89
2	10	-78.3	2.13
3	20	-69.5	0.71
4	30	-59.3	1.17
5	40	-47.8	4.40
6	50	-37.7	5.75
7	60	-28.5	5
8	70	-17.4	13
9	80	-9.3	7
10	90	0.4	0
11	100	9.4	6
12	110	19.8	1
13	120	29.7	1
14	130	38.4	4
15	140	47.8	4.40
16	150	58.9	1.83
17	160	69.5	0.71
18	170	78.9	1.37
19	180	89.8	0.22

4.4. Pengujian pergerakan robot

Pengujian pergerakan robot dilakukan untuk mengetahui nilai kecepatan putar masing-masing motor untuk menggerakan sistem. Pengujian dilakukan dengan mengamati masing-masing kecepatan motor ketika sistem diberi kecepatan $vx, vy,$ dan ω. Tabel 4.6 menunjukan kecepatan masing-masing motor untuk melakukan gerakan pada sistem.

Tabel 4. 6 Data kecepatan motor dengan pergerkan $vx, vy, dan\ \omega$

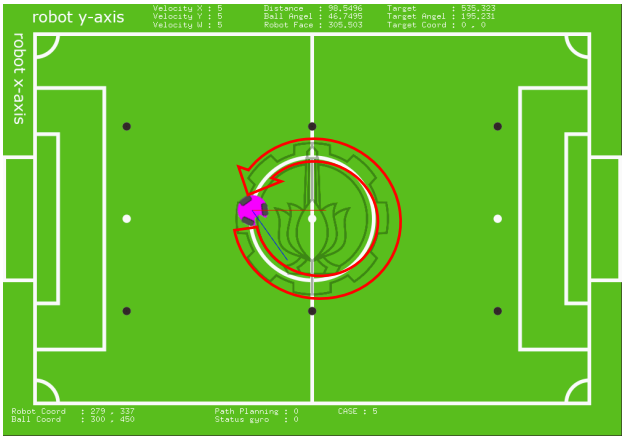
No.	Pergerakan	Motor 1 (rpm)	Motor 2 (rpm)	Motor 3 (rpm)
1	vx	112	225	112
2	vy	193	0	193
3	ω	54	54	54

Pengujian ini dilakukan dengan 2 tipe pengujian. Pengujian yang pertama berupa pengujian pergerakan robot tanpa mempertimbangkan sensor gyro sebagai parameternya. Sedangkan pengujian yang kedua adalah pengujian pergerakan robot dengan mempertimbangkan nilai dari sensor gyro sehingga robot akan bergerak sesuai dengan *offset* dari nilai keluaran sensor *gyro* yang telah ditentukan.

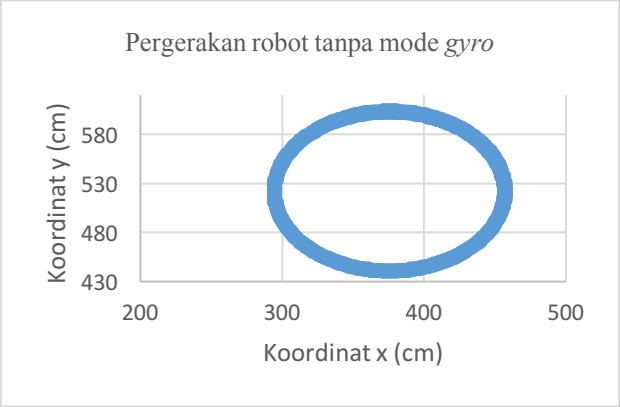
4.4.1. Pengujian pergerakan robot tanpa mode *gyro*

Pengujian pergerakan robot tanpa mode *gyro* dilakukan dengan memberikan kecepatan vx, vy, dan $v\omega$ tanpa mempertimbangkan nilai keluaran dari sensor gyro sehingga robot akan bergerak sesuai dengan arah muka robot.

Gambar 4.2 menunjukan gambar pergerakan robot pada *simulator* dengan memberikan nilai vx, vy, dan $v\omega$ yang sama. Dari gambar tersebut dapat diamati bahwa sistem akan bergerak melingkar seperti membentuk lingkaran ketika sistem diberikan nilai kecepatan vx, vy, dan ω yang sama. Gambar 4.3 menunjukan grafik dari trajectory pergerakan robot tanpa mode *gyro*.



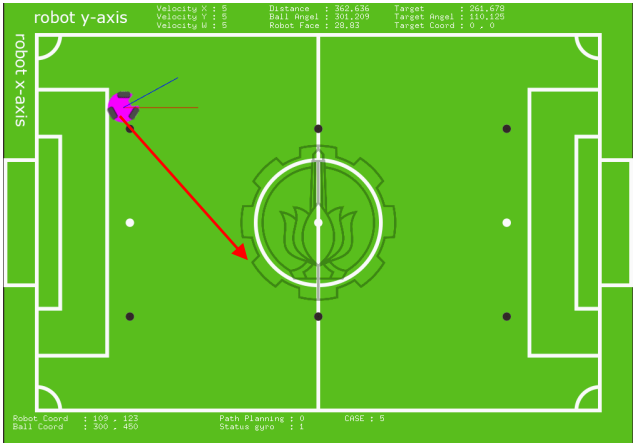
Gambar 4. 2 Pergerakan robot tanpa mode *gyro* pada *simulator*



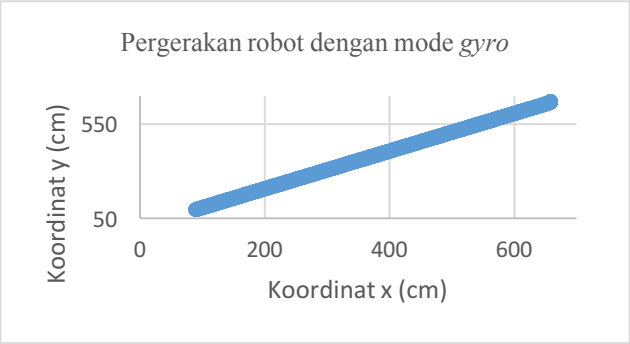
Gambar 4. 3 Grafik *trajectory* pergerakan robot tanpa mode *gyro*

4.4.2. Pengujian pergerakan robot dengan mode *gyro*

Pengujian pergerakan robot dengan mode *gyro* dilakukan dengan memberikan kecepatan $vx, vy,$ dan ω dan mengaktifkan mode gyro sehingga nilai keluaran dari sensor gyro menjadi salah satu parameter untuk menentukan arah pergerakan robot.



Gambar 4. 4 Pergerakan robot dengan mode *gyro* pada *simulator*



Gambar 4. 5 Grafik *trajectory* robot dengan mode *gyro*

Gambar 4.4 menunjukan gambar pergerakan robot pada *simulator* dengan memberikan nilai vx, vy, dan ω yang sama. Dari gambar tersebut dapat diamati bahwa sistem akan bergerak lurus serong kedepan membentuk seperti bentuk garis diagonal ketika sistem diberikan nilai kecepatan vx, vy, dan ω yang sama. Gambar 4.5 menunjukan grafik dari trajectory pergerakan robot tanpa mode gyro.

4.5. pengujian algoritma

Pengujian algoritma dilakukan dengan memberikan beberapa sknario yang berbeda. Algoritma yang diuji merupakan algoritma yang tertanam pada robot, seperti, algoritma pergerakan menuju ke sebuah titik dan algoritma *path planning*. Pada masing-masing skenario, peletakan posisi awal robot dan bola diletakan pada posisi yang berjauhan. Sedangkan untuk posisi obstacle diletakan secara acak. Pengujian ini dilakukan dengan tujuan untuk mengetahui apakah algoritma dapat bekerja dengan baik pada setiap kondisi tersebut.

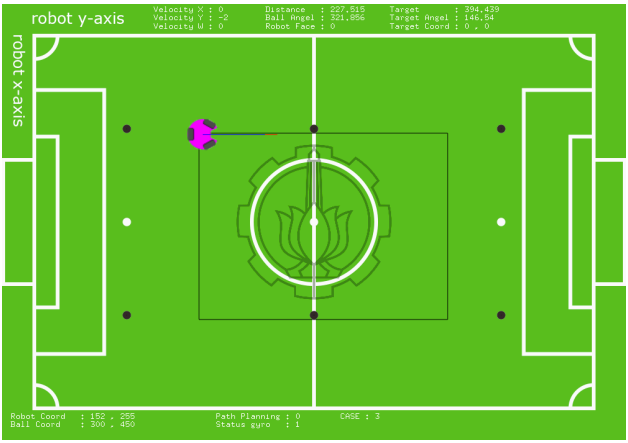
Pengujian ini dilakukan pada 2 bagian, yaitu pengujian algoritma pada *simulator* dan pada sistem yang sebenarnya. Skenario yang diujikan pada *simulator* dan sistem yang sebenarnya adalah skenario yang sama. Sehingga dapat dibandingkan hasil pengujian algoritma path planning pada *simulator* dan sistem yang sebenarnya.

4.5.1. Pengujian algortima pada *simulator*

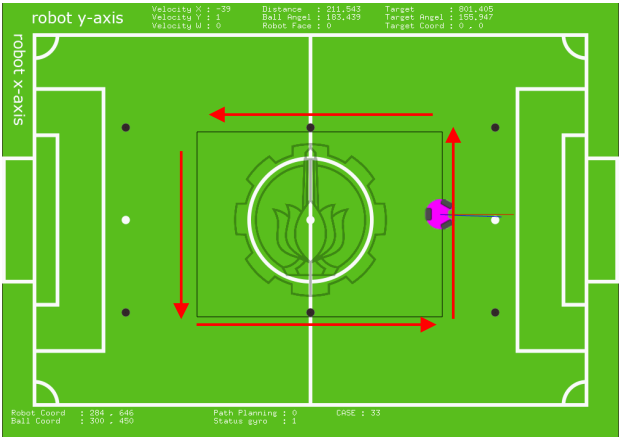
Pengujian algoritma pada *simulator* dilakukan dengan memberikan beberapa skenario berbeda dengan meletakan posisi robot dan bola berjauhan dan juga peletakan posisi obstacle secara acak. Kondisi pada *simulator* merupakan kondisi yang ideal sehingga robot diasumsikan sudah menerima informasi letak posisi bola dan *obstacle*, tidak mempertimbangkan tingkat kecerahan cahaya atau faktor eksternal lain dari lingkungan.

Skenario pertama adalah skenario tanpa adanya *obstacle*. Ada 3 titik tuju yang harus dicapai oleh sistem. Gambar 4.6 merupakan posisi awal dari skenario. Pengujian skenario ini dilakukan untuk mengetahui performa sistem dalam melakukan pergerakan translasi.

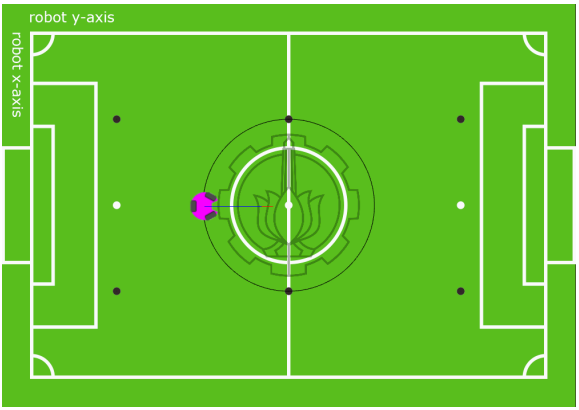
Pada gambar 4.6 dapat diamati garis berwarna hitam yang merupakan jalur yang harus dilalui menuju titik target. Robot akan selalu mengikuti jalur menuju titik target. Gambar 4.7 menunjukan bahwa robot akan selalu mengikuti jalur yang telah dibentuk.



Gambar 4. 6 Posisi awal robot untuk menuju titik yang telah ditentukan pada *simulator*



Gambar 4. 7 Pergerakan robot mengikuti jalur yang telah dibuat untuk menuju titik yang telah ditentukan

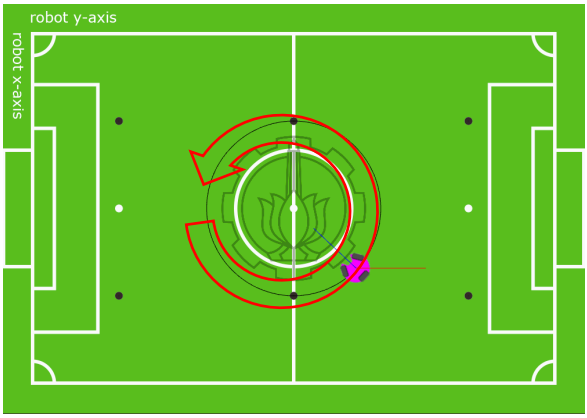


Gambar 4. 8 Posisi awal robot untuk gerak melingkar pada *simulator*

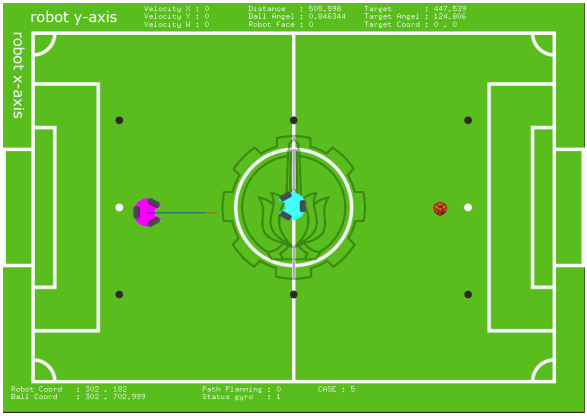
Skenario kedua adalah skenario gerak melingkar. Gambar 4.8 merupakan posisi awal dari skenario. Pengujian skenario ini dilakukan untuk mengetahui performa sistem dalam melakukan pergerakan rotasi.

Gambar 4.9 menunjukan bahwa robot akan selalu mengikuti jalur yang telah dibentuk.

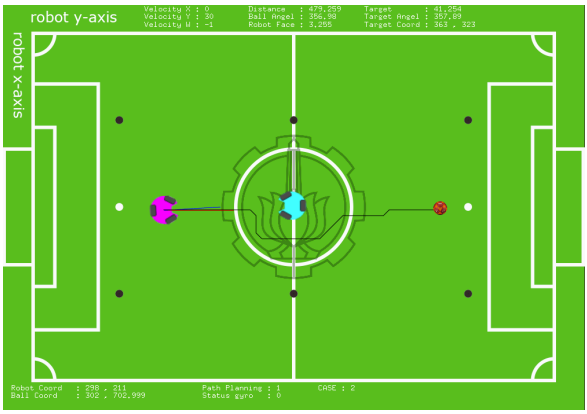
Skenario ketiga adalah skenario ketika robot dihadapkan dengan *obstacle* dan bola bergerak menuju belakang barisan *obstacle*. Posisi awal dapat dilihat pada gambar 4.10.



Gambar 4. 9 pergerakan robot melingkar pada *simulator*



Gambar 4. 10 Posisi awal robot dihadapkan dengan *obstacle* dengan titik target dibelakang *obstacle*



Gambar 4. 11 Jalur yang harus dilalui oleh robot untuk menuju titik yang ditentukan dengan menghindari *obstacle*

Gambar 4.11 menunjukan bahwa jalur menuju titik target tetap dapat dibentuk dengan mempertimbangkan letak *obstacle* yang ada sehingga dapat dihindari.

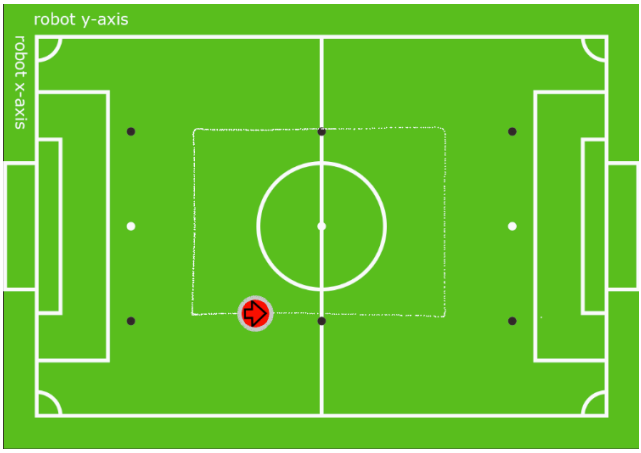
4.5.2. Pengujian algoritma pada sistem

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

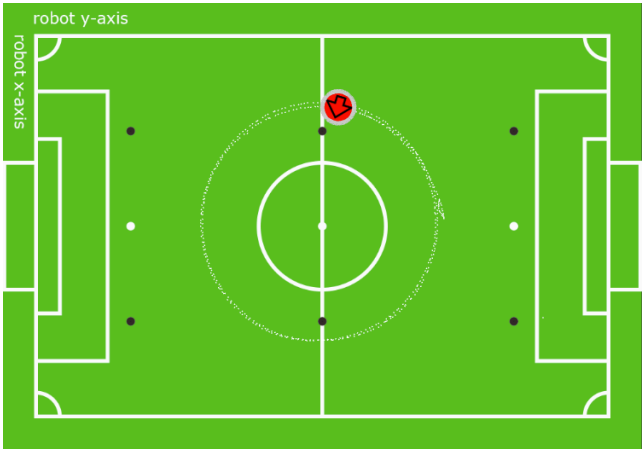
Skenario pertama adalah skenario tanpa adanya *obstacle.* Robot akan membuat jalur menuju ke koordinat yang telah ditentukan. Ada 3 titik tuju yang harus dicapai oleh sistem kemudian kembali ke titik awal.

Pada gambar 4.12 menunjukan gambar pergerakan robot yang dimonitor dan ditampilkan pada GUI, Robot akan selalu mengikuti jalur menuju titik target yang telah ditentukan.

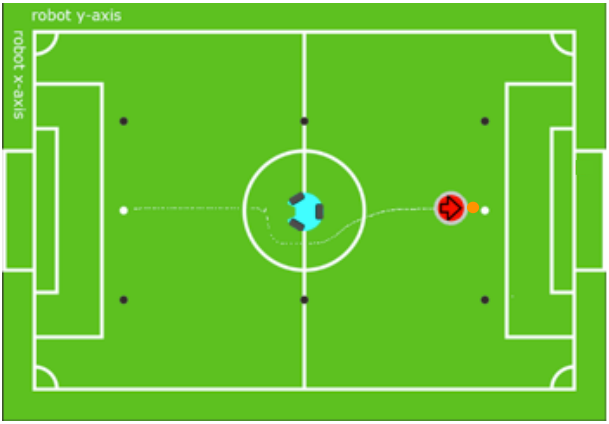
Skenario kedua adalah skenario pergerakan melingkar. Tampilan pergerakan robot yang dimonitor pada GUI dapat dilihat pada gambar 4.13.



Gambar 4. 12 Bentuk *trajectory* robot yang dimonitor pada GUI



Gambar 4. 13 Bentuk *trajectory* pergerakan robot melingkar dengan menghadap pada pusat lingkaran

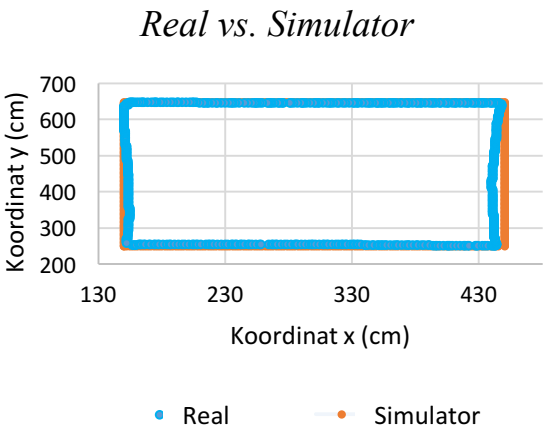


Gambar 4. 14 Bentuk *trajectory* robot pada skenario *obstacle avoidance*

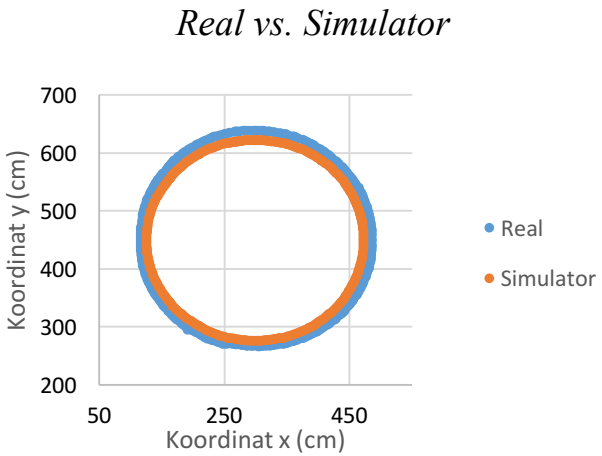
Skenario ketiga adalah skenario ketika robot dihadapkan dengan *obstacle* dan bola bergerak menuju belakang *obstacle.* Tampilan pergerakan robot yang dimonitor pada GUI dapat dilihat pada gambar 4.14 yang menunjukan bahwa robot bergerak menuju titik dimana bola berada dan ketika robot telah mendekati *obstacle*, jalur yang dibentuk berbelok menghindari titik dari posisi *obstacle*. Kemudian robot bergerak mengikuti jalur yang telah dibentuk sehingga robot dapat melewati *obstacle* dan menuju ke titik dimana bola berada tanpa menabrak *obstacle*.

Dari hasil pengujian algoritma pada simulator dan pada sistem yang sebeneranya, maka perbandingan antara pergerakan trajectory yang direkam pada simulator dan pada sistem yang sebenarnya dapat digambarkan dalam bentuk grafik.

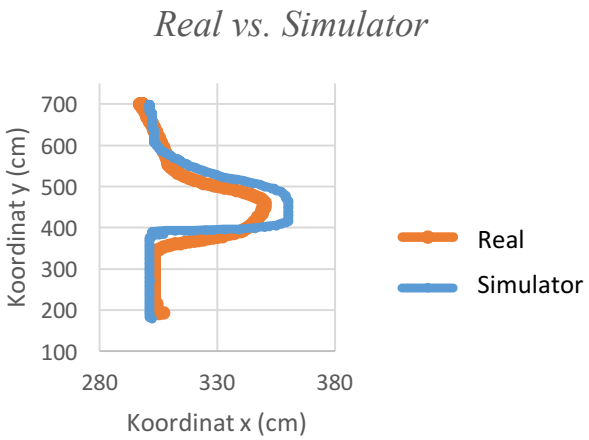
Gambar 4.15 menunjukan grafik perbandingan hasil yang diperoleh dari *simulator* dan pada sistem yang sesungguhnya. Grafik berwarna biru menunjukan *trajectory* yang dihasilkan oleh pergerakan robot, sedangkan grafik berwarna oranye merupakan yang dihasilkan oleh *simulator*. Dari data-data yang digambarkan pada grafik gambar 4.15 dapat dperoleh perhitungan nilai *error* RMS sebesar 6.52 cm.



Gambar 4. 15 Perbandingan bentuk *trajectory* pada sistem dan pada *simulator* pada skenario pergerakan translasi



Gambar 4. 16 Perbandingan bentuk *trajectory* pada sistem dan pada *simulator* pada skenario pergerakan rotasi



Gambar 4. 17 Perbandingan bentuk *trajectory* pada sistem dan pada *simulator* pada skenario pergerakan *obstacle avoidance*

Gambar 4.16 menunjukan grafik perbandingan hasil yang diperoleh dari *simulator* dan pada sistem yang sesungguhnya pada pergerakan rotasi. Grafik berwarna biru menunjukan *trajectory* yang dihasilkan oleh pergerakan robot, sedangkan grafik berwarna oranye merupakan yang dihasilkan oleh *simulator*. Dari data-data yang digambarkan pada grafik gambar 4.16 dapat diperoleh nilai *error* RMS sebesar 10.59 cm.

Gambar 4.17 menunjukan grafik perbandingan hasil yang diperoleh dari *simulator* dan pada sistem yang sesungguhnya. Grafik berwarna oranye menunjukan *trajectory* yang dihasilkan oleh pergerakan robot yang sebenarnya, sedangkan grafik berwarna biru merupakan yang jalur *trajectory* yang dihasilkan oleh pergerakan robot pada *simulator*. Dari data-data yang digambarkan pada grafik gambar 4.16 dapat diperoleh nilai *error* RMS sebesar 18.42 cm.

4.6. Pengujian respon sistem terhadap algoritma

Pengujian respon sistem terhadap algoritma ini dilakukan untuk mengetahui seberapa cepat komputer *mini* pada sistem dapat memproses algoritma yang telah dibuat. Tabel 4.7 menunjukan hasil pengujian respon sistem terhadap algoritma yang telah dibuat.

Tabel 4. 7 Pengujian respon sistem terhadap algoritma

No.	Titik awal (x,y)	Titik akhir (x,y)	Jumlah *obstacle*	Waktu (detik)
1	300,200	300,550	0	6.81
2	300,200	300,550	1	7.33
3	300,200	300,550	1	7.71
4	300,200	300,550	2	8.15

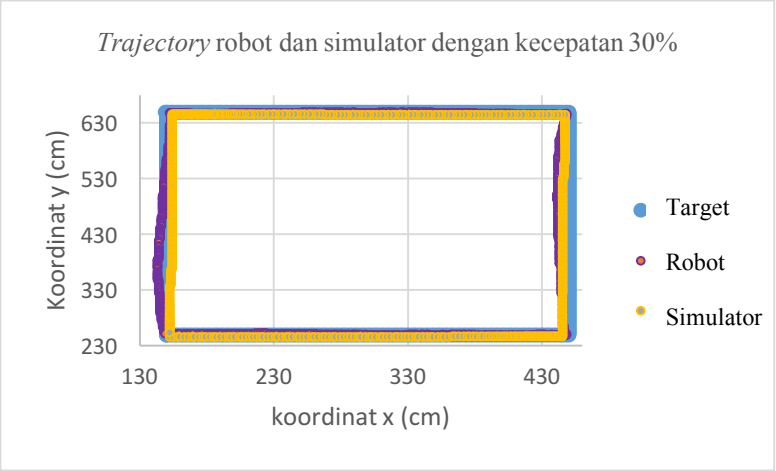
Pada pengujian ini, dilakukan 4 kali percobaan dengan kecepatan motor yang sama, posisi titik awal dan titik akhir pada koordinat yang sama, akan tetapi jumlah dan peletakan *obstacle* yang berbeda-beda. Dari data-data tersebut, maka dapat dihitung hasil rata-rata dari waktu yang ditempuh dari titik target menuju titik akhir adalah sebagai berikut:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{4}\sum_{i=1}^{n} x_i = 7.5\ detik$$

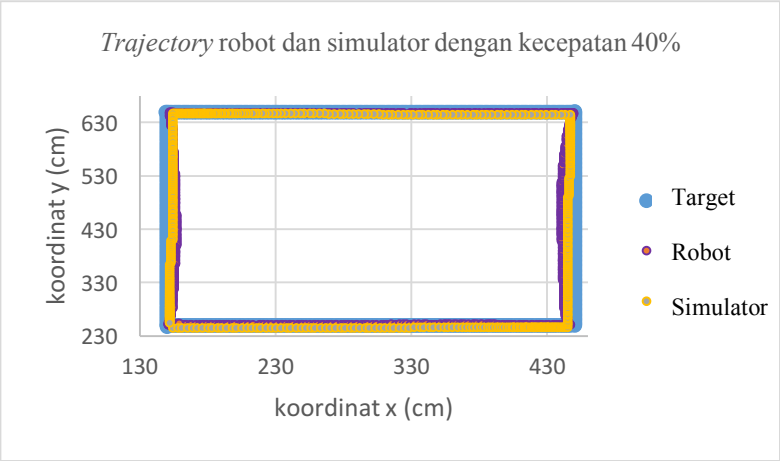
4.7. Pengujian pengaruh kecepatan terhadap *trajectory*

Pengujian ini dilakukan untuk mengetahui pengaruh kecepatan terhadap *trajectory* pergerakan dari sistem yang sebenarnya dan sistem pada simulator. Pengujian ini dilakukan dengan memberikan jalur yang menghubungkan tiga titik tujuan sehingga membentuk seperti persegi. Sistem akan diuji dengan kecepatan yang berbeda-beda. Data dari *trajectory* akan direkam dan dibandingkan dengan koordinat target yang dituju sehingga akan didapatkan nilai *error* RMS dari masing-masing *trajectory* pada sistem yang sebenarnya, maupun pada *simulator* dan waktu yang diperlukan untuk menempuh titik target yang dituju.

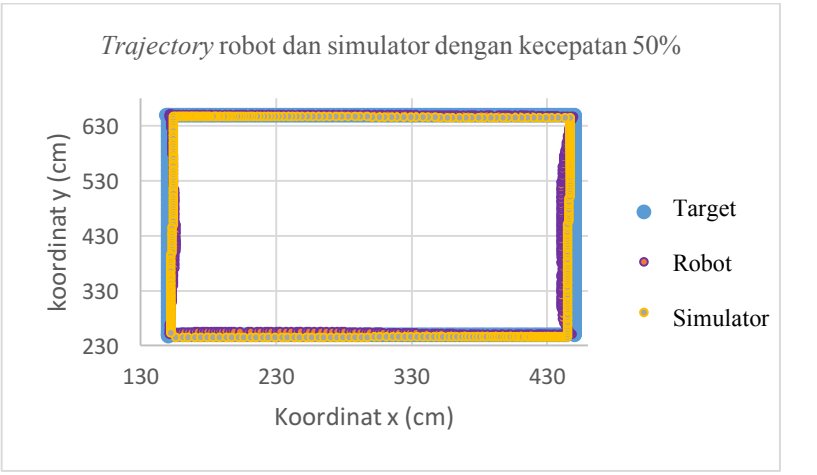
Hasil dari masing-masing pengujian digambarkan dalam bentuk grafik perbandingan antara koordinat target dengan trajectory yang dihasilkan pada simulator dan sistem yang sebeneranya seperti ditunjukan pada gambar sebagai berikut:



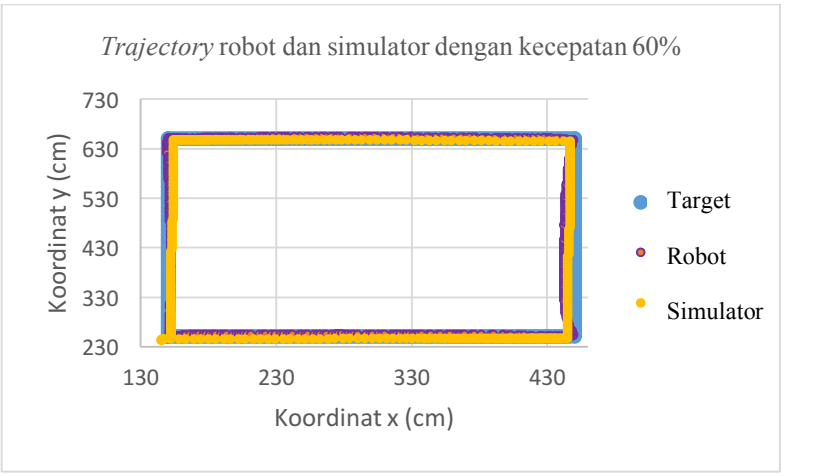
Gambar 4. 18 *Trajectory* pada sistem yang sebenarnya dan pada *simulator* dengan kecepatan 30%



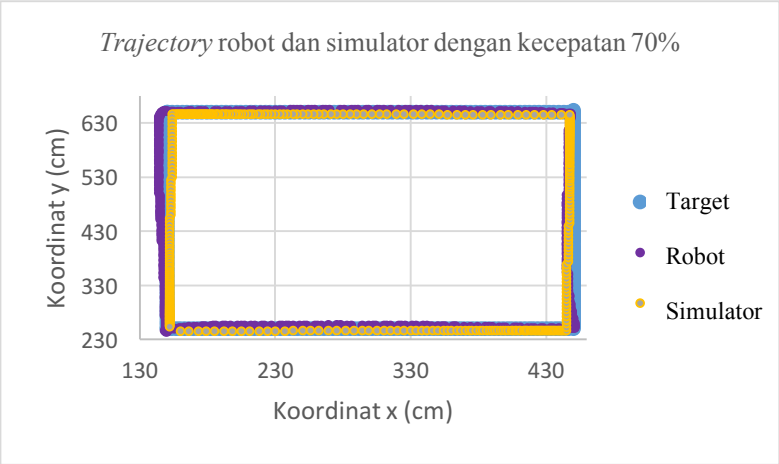
Gambar 4. 19 *Trajectory* pada sistem yang sebenarnya dan pada *simulator* dengan kecepatan 40%



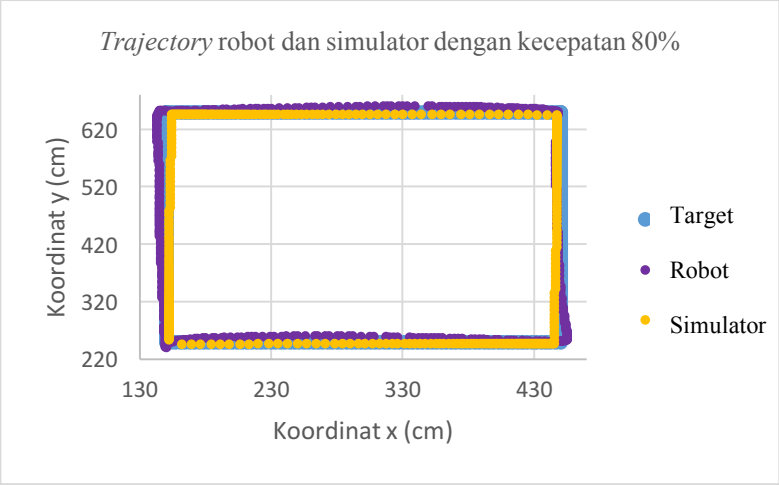
Gambar 4. 20 *Trajectory* pada sistem yang sebenarnya dan pada *simulator* dengan kecepatan 50%



Gambar 4. 21 *Trajectory* pada sistem yang sebenarnya dan pada *simulator* dengan kecepatan 60%



Gambar 4. 22 *Trajectory* pada sistem yang sebenarnya dan pada *simulator* dengan kecepatan 70%



Gambar 4. 23 *Trajectory* pada sistem yang sebenarnya dan pada *simulator* dengan kecepatan 90%

Dari hasil masing-masing pengujian tersebut, maka dapat diamati waktu tempuh dan perhitungan nilai error RMS yang disajikan pada tabel 4.8 sebagai berikut:

Tabel 4. 8 Tabel lama waktu dan nilai *error RMS* terhadap persentase kecepatan sistem yang sebenarnya dan *simulator*

No.	Kecepatan (%)	*Real*		*Simulator*	
		Waktu (detik)	RMSE (cm)	Waktu (detik)	RMSE (cm)
1	30	26.35	4.22	27.80	3.89
2	40	20.60	5.93	21.90	5.21
3	50	16.37	6.43	20.12	5.94
4	60	14.20	7.71	15.80	7.16
5	70	11.96	9.36	14.91	8.74
6	80	10.80	15.58	13.80	13.52

BAB V
KESIMPULAN DAN SARAN

Berdasarkan hasil dari pengujian yang dilakukan pada tugas akhir ini, maka dapat ditarik beberpa kesimpulan dan saran untuk pengembangan sistem kedepannya.

5.1. Kesimpulan

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

1. Hasil pengujian perbandingan antara *trajectory* pada sistem yang sebenarnya dan pada simulator dengan skenario membentuk jalur persegi memiliki nilai *error* RMS sebesar 6.52 cm.
2. Hasil pengujian perbandingan antara *trajectory* pada sistem yang sebenarnya dan pada simulator dengan skenario membentuk jalur melingkar memiliki nilai *error* RMS sebesar 10.59 cm.
3. Hasil pengujian perbandingan antara *trajectory* pada sistem yang sebenarnya dan pada simulator dengan skenario menghindari *obstacle* memiliki nilai *error* RMS sebesar 18.42 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 7.5 detik.
5. Pengaruh kecepatan terhadap nilai *error* RMS dari *trajectory* sistem adalah semakin tinggi kecepatan maka semakin besar nilai *error* RMS. Pada hasil pengujian dengan kecepatan 30% menghasilkan nilai *error* RMS sebesar 4.22 cm pada sistem yang sebenarnya dan 3.89 cm pada *simulator*. Sedangkan pengujian dengan kecepatan 80%, nilai *error* RMS yang dihasilkan adalah sebesar 15.58 cm pada sistem yang sebenarnya dan 13.52 cm pada *simulator*.

5.2. Saran

Saran yang dapat diberikan oleh penulis untuk pengembangan dari tugas akhir adalah sebagai berikut:

1. Pengembangan *Simulator* dan GUI menjadi satu aplikasi agar dapat dengan mudah memonitor dan membandingan hasil pergerakan robot pada *simulator* dan sistem yang sebenarnya.
2. Pengembangan algoritma sehingga dapat diaplikasikan pada keadaan dengan *obstacle* yang dinamis.
3. Pengembangan algoritma sehingga dapat diimplementasikan pada multi robot.

DAFTAR PUSTAKA

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

[2] P. Lester, "A* pathfinding for begineers," 2005. [Online]. Available: http://www. policyalmanac.org/games/aStarTutorial.htm. [Accessed 15 May 2017].

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

[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] F. A. Hermawati, Pengolahan Citra Digital Konsep & Teori, Yogyakarta: Andi, 2013.

[7] T. Kumar and K. Verna, "A Theory Based on Conversion of RGB Image to Gray Image," *International Journal of Computer Applications,* vol. 7, no. 22, pp. 7-10, September 2010.

[8] B. A. Forouzan, Data Comunications And Networking, Fourth edition ed., New York,: Alan R. Apt, 2007, pp. 709-711.

[9] A. Burg, A. Meruani, B. Sandheinrich and M. Wickmann, *Mems Gyroscopes and their Applications,* Evanston: Northwestern University, p. 3.

[10] *UM1472 User Manual,* Geneva: STMicroelectronics, 2016.

[11] *Controlling DC Brush Motors with H-Bridge Driver ICs,* San Diego, California: ROHM, 2009.

[12] "Apple Developer support," Apple .Inc, 2017. [Online]. Available: https://developer.apple.com/support/xcode/. [Accessed 22 May 2017].

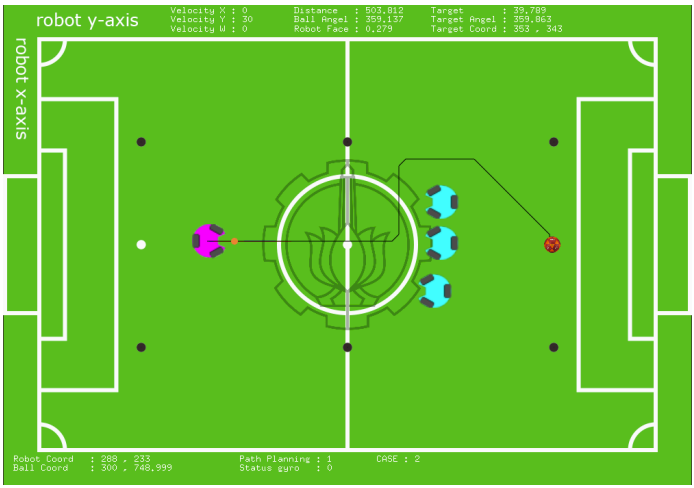
[13] H. P. Halvorsen, *Introduction to Visual Studio and C#,* Notodden: University College of Southeast Norway, 2016.

[14] Z. Lieberman, T. Watson and A. Castro , "OpenFrameWorks," 14 April 2017. [Online]. Available: http://openframeworks.cc/about/. [Accessed 22 May 2017].

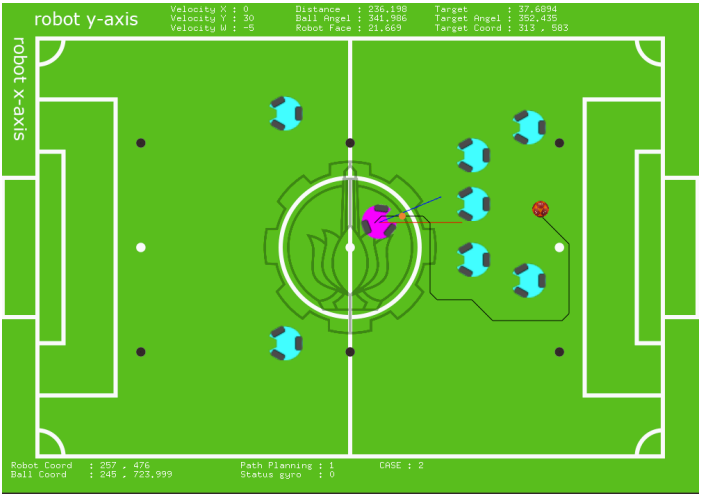
[15] "OpenCV," 2017. [Online]. Available: http://opencv.org. [Accessed 22 May 2017].

LAMPIRAN

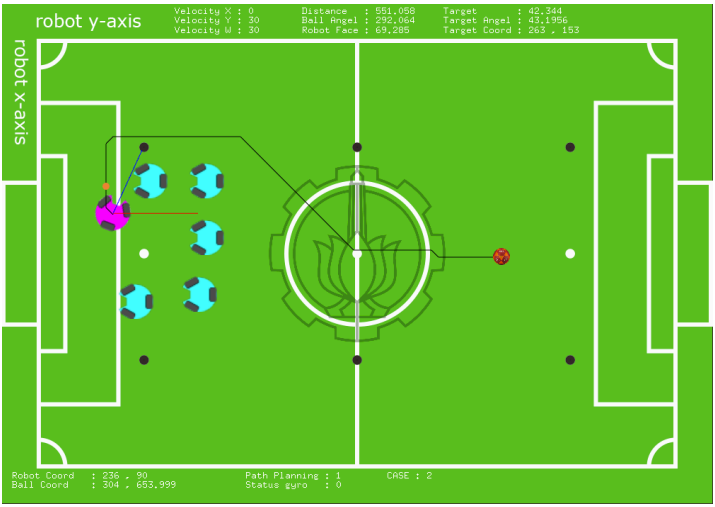
1. Skenario musuh berbaris



2. Skenario posisi musuh ramai dan acak



3. Skenario robot terkepung musuh



4. Foto Robot

Listing Program Path Planning

```
#include <algorithm>
#include "PathPlanning.h"
#include <math.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::LuasLapangan(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 itu = std::find(obstacle.begin(), obstacle.end(),
koordinat_);
      if (itu != obstacle.end()) {
        obstacle.erase(itu);
      }
}

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 itu = titik_.begin(); itu != titik_.end();) {
        delete *itu;
        itu = titik_.erase(itu);
      }
}

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)));
}
```

Listing Program *Simulator*

```
#include "ofApp.h"
#include "Serial.h"
#include "PathPlanning.h"

AStar::aktifasi aktif;

void ofApp::setup(){
    lapangan.load("images/Lapangan.png");
    robot.load("images/robotM.png");
    bola.load("images/ball2.png");
    musuh.load("images/robotC.png");
    logo2.load("images/itshitam.png");

    ofSetWindowShape(1016, 716);
    ofSetFrameRate(30);

    open_setting();

    posX = 300;  posY = 200;
    goalX = posX;  goalY = posY;
    velX = 0, velY = 0;

    robotY = ofGetWidth()/2;
    robotX = ofGetHeight()/2;
    ballY = ofGetWidth()/2;
    ballX = ofGetHeight()/2;
    speedX = 0;
    speedY = 0;
}

int sebelum_x,sebelum_y;

void ofApp::update(){
    simulasi();
    if(status_cou==1){
        if(robotX==sebelum_x && robotY==sebelum_y)
        {status_cou==1;}
        else {cout<<(int)robotX-lapX<< " " << (int)robotY-
lapY<<endl;}

        sebelum_x=robotX; sebelum_y=robotY;
    }
}
void ofApp::draw(){
    ofClear(0);
```

```
 lapangan.draw(0, 0, 1016, 716);

 ofPushStyle();
 ofEnableAlphaBlending();
 ofSetColor(0, 255, 0, 85);
 logo2.draw(383,233,250,250);
 ofDisableAlphaBlending();
 ofPopStyle();

  ofPushStyle();
  ofSetColor(0, 0, 0);
     ofNoFill();
     ofSetLineWidth(1);
     ofSetCircleResolution(360);
     ofDrawCircle(450+lapY, 300+lapX, 150);
  ofPopStyle();

ofPushMatrix();
 ofTranslate(robotY,robotX);
 ofRotate(-rotation);
 robot.draw(-25,-25,50,50);
 ofPopMatrix();

 ofPushStyle();
 ofSetColor(255, 0, 0);
 ofDrawLine(robotY, robotX,robotY+120, robotX);
 ofPopStyle();

 ofPushStyle();
 ofSetColor(0, 0, 255);
 ofDrawLine(robotY, robotX, yt, xt);
 ofPopStyle();

 if(i>0){
     if(status_path == TRUE){
         ofPushStyle();
         ofSetColor(0, 0, 0);
         line.draw();
         ofPopStyle();
     }
     // GAMBAR BOLA
     ofPushMatrix();
     ofTranslate(posY,posX);
     bola.draw(-12.5,-12.5,25,25);
     ofPopMatrix();
     //----------------------------//

 if (n != 0) {
```

```
            obsx[n] = ox;
            obsy[n] = oy;
            for (int z = 1; z <= n; z++) {

                ofPushMatrix();
                ofTranslate(obsy[z], obsx[z]);
                musuh.draw(-25,-25,50,50);
                ofPopMatrix();

                }
            }
        }

stringstream ss;
    ss << "Velocity X : " << temp_vx << endl;
    ss << "Velocity Y : " << temp_vy << endl;
    ss << "Velocity W : " << temp_vw << endl;

    ofDrawBitmapString(ss.str().c_str(), 250, 20);

    stringstream ss2;
    ss2 <<"Distance   : "<< jarak << endl;
    ss2 <<"Ball Angel : "<< sudut << endl;
    ss2 <<"Robot Face : "<< rotation <<endl;

    ofDrawBitmapString(ss2.str().c_str(), 430, 20);

    stringstream ss3;
    ss3 <<"Target       : "<< jarak_target << endl;
    ss3 <<"Target Angel : "<< sudut_target << endl;
    ss3 <<"Target Coord : "<< titikTuju_x <<" , "<<
titikTuju_y << endl;

    ofDrawBitmapString(ss3.str().c_str(), 630, 20);

    stringstream ss4;
    ss4 <<"Robot Coord   : "<< (int)robotX-65 <<" , "<<
(int)robotY-73 << endl;
    ss4 <<"Ball Coord    : "<< posX-58 <<" , "<< posY-58 <<
endl;

    ofDrawBitmapString(ss4.str().c_str(), 20, 675);

    stringstream ss5;
    ss5 <<"Path Planning : "<< status_path << endl;
    ss5 <<"Status gyro   : "<< gyro_status << endl;

    ofDrawBitmapString(ss5.str().c_str(), 350, 675);
```

```
    stringstream ss6;
    ss6 <<"CASE : "<< motion << endl;
    ss6 <<"CASE : "<< rotation << endl;
    ss6 <<"CASE : "<< temp_o << endl;
    ofDrawBitmapString(ss6.str().c_str(), 550, 675);
    }

void ofApp::mousePressed(int x, int y, int button){

    if(button == 0){

        ballX = y;  ballY = x;
        i++;
    }

    //ADD OBSTACLE
    if (button == 2) {
        n++;

        ox = (y / 10) * 10 + 5;
        oy = (x / 10) * 10 + 5;
        hitox = (ox - 58) / 10;
        hitoy = (oy - 58) / 10;

        if (hitoy < 0) {
            oy = 58;
            hitoy = 0;
        }

        if (hitox < 0) {
            ox = 58;
            hitox = 0;
        }

        if (hitoy > 89) {
            oy = 958;
            hitoy = 89;
        }

        if (hitox > 59) {
            ox = 658;
            hitox = 59;
        }
        cout << "OBS: "<< endl;

        for (int k = hitoy - 5; k < hitoy + 6; k++) {
```

```
            for (int l = hitox - 5; l < hitox + 6; l++) {
                aktif.addObstacle({ k,l });
                cout << l << "," << k <<endl;

            }
            cout<< endl;
        }
    }

}

bool ofApp::vxvy_control(float _x, float _y, float _x1,
float _y1, int &_vx, int &_vy, int _v, float akurasi)
{
    static float error;
    static float error_before;

    gyro_status = 1;

    error = pythagoras(_x, _y, _x1, _y1);

    p_vxvy_control = error * kp_vxvy_control;
    i_vxvy_control = i_vxvy_control + error *
ki_vxvy_control;
    d_vxvy_control = (error - error_before) *
kd_vxvy_control;

    pid_vxvy_control = p_vxvy_control + i_vxvy_control +
d_vxvy_control;

    if (pid_vxvy_control > _v) pid_vxvy_control = _v;
    else if (pid_vxvy_control < -_v) pid_vxvy_control = -_v;

    error_before = error;

    _vx = (int)((_x1 - _x) * pid_vxvy_control / error);
    _vy = (int)((_y1 - _y) * pid_vxvy_control / error);

    if (abs(error) < akurasi) return true;
    else return false;
}

bool ofApp::w_control(float _w, float _w1, int &_vw, int _v,
int toleransi)
{
    static float error;
    static float error_before;
    error = _w1 - _w;
```

```
    if (error > 180) error = error - 360;
    else if (error < -180) error = error + 360;

    p_w_control = error * kp_w_control;
    i_w_control = i_w_control + error * ki_w_control;
    d_w_control = (error - error_before) * kd_w_control;

    pid_w_control = p_w_control + i_w_control + d_w_control;

    if (pid_w_control > _v) pid_w_control = _v;
    else if (pid_w_control < -_v) pid_w_control = -_v;

    error_before = error;

    _vw = (int)pid_w_control;

    if (abs(error) <= toleransi) return true;
    else return false;
}

float ofApp::pythagoras(float _x, float _y, float _x1, float
_y1)
{
    return sqrt(pow((_x1 - _x), 2) + pow((_y1 - _y), 2));
}

void ofApp::chase_ball(float _distance, float _tetha, float
_distance1, float _tetha1, int _min_v, int _max_v, int
_max_w, int &_v, int &_w, int &_front)
{
    signed int temp_tetha;
    gyro_status = false;

    _v = v_control(_distance, _distance1, 0.5, 0, 0, _min_v,
_max_v);
    _w = o_control(_tetha, _tetha1, 0.7, 0, 0, -_max_w,
_max_w);
    _front = _tetha; //- (_w * 1.4);

    if (_distance <= 0 && _tetha <= 0)
    {
        _v = 0;
        if (_w >= 0) _w = 20;
        else _w = -20;

        _front = 0;
    }
```

```
    temp_tetha = (signed int)(_tetha - _tetha1);

    if (temp_tetha > 180)temp_tetha -= 360;
    if (temp_tetha < -180)temp_tetha += 360;

    if (abs(temp_tetha) > 40 && _distance<40) _v = 0;
    return;
}

float ofApp::v_control(float _in, float _sp, float _kp,
float _ki, float _kd, int _min_v, int _max_v)
{
    static float error;
    static float error_before;
    error = _in - _sp;

    p_v_control = error * _kp;
    i_v_control = i_v_control + error * _ki;
    d_v_control = (error - error_before) * _kd;

    pid_v_control = p_v_control + i_v_control + d_v_control;

    if (pid_v_control > _max_v) pid_v_control = _max_v;
    else if (pid_v_control < _min_v) pid_v_control = _min_v;

    error_before = error;

    return pid_v_control;
}

float ofApp::o_control(float _in, float _sp, float _kp,
float _ki, float _kd, int _min_w, int _max_w)
{
    static float error;
    static float error_before;
    error = _in - _sp;

    if (error > 180) error = error - 360;
    if (error < -180) error = error + 360;

    p_o_control = error * _kp;
    i_o_control = i_o_control + error * _ki;
    d_o_control = (error - error_before) * _kd;

    pid_o_control = p_o_control + i_o_control + d_o_control;

    if (pid_o_control > _max_w) pid_o_control = _max_w;
```

```
    else if (pid_o_control < _min_w) pid_o_control = _min_w;

    error_before = error;

    return pid_o_control;
}

bool ofApp::surrounds_the_ball(float _w, float _w1, float
_w_max, int &_vx, int & _vw)
{
    static float error;
    static float error_before;
    static int counter = 0;
    error = _w1 - _w;

    if (error > 180) error = error - 360;
    else if (error < -180) error = error + 360;

    p_stb = error * kp_stb;
    i_stb = i_stb + error * ki_stb;
    d_stb = (error - error_before) * kd_stb;

    pid_stb = p_stb + i_stb + d_stb;

    if (pid_stb > _w_max) pid_stb = _w_max;
    else if (pid_stb < -_w_max) pid_stb = -_w_max;

    error_before = error;

    if (error > -20 && error < 20) counter++;
    else counter = 0;


    _vx = (int) (0.2 * pid_stb);
    _vw = (int) (0.7 * pid_stb);

    if (counter > 2) return true;
    else return false;
}

float ofApp::point2angle(float _x, float _y, float _x1,
float _y1)
{
    float angle = atan2f(-(_x1 - _x), (_y1 - _y));
    angle *= 57.295779;

    if (angle < 0) angle = 360 + angle;
```

```
    return angle;
}


bool ofApp::posisi_terhadap_titik(float _x, float _y, float
xr, float yr, float _x1, float _y1, int &_vx, int &_vy, int
_v, float akurasi) {
    _x1 = _x1 + xr;
    _y1 = _y1 + yr;
    return vxvy_control(_x, _y, _x1, _y1, _vx, _vy, _v,
akurasi);
}

bool ofApp::kelilingi_titik(float r, float sudut_tujuan,
float XsetPoint, float YsetPoint, float resolusi, float
akurasi) {
    float sudut_sekarang = atan2(-(robotX - XsetPoint) ,
(robotY - YsetPoint))*57.295779513082320876798154814105;
    if (sudut_sekarang < 0) sudut_sekarang += 360;

    float error_sudut = sudut_tujuan - sudut_sekarang;
    float sudut_selanjutnya = 0;
    int x, y;

    if (error_sudut > 180) error_sudut = error_sudut - 360;
    else if (error_sudut < -180) error_sudut = error_sudut +
360;

    if (error_sudut > 0) {
        if (abs(error_sudut) < resolusi) sudut_selanjutnya =
sudut_tujuan;
        else sudut_selanjutnya = sudut_sekarang + resolusi;
    }

    else {
        if (abs(error_sudut) < resolusi) sudut_selanjutnya =
sudut_tujuan;
        else sudut_selanjutnya = sudut_sekarang - resolusi;
    }

    x = -r *
sin(sudut_selanjutnya*0.017453292519943295769236907684489);
    y = r *
cos(sudut_selanjutnya*0.017453292519943295769236907684489);

    if (posisi_terhadap_titik(robotX, robotY, x, y,
XsetPoint, YsetPoint, temp_vx, temp_vy, 30, 20) &&
abs(error_sudut) < akurasi) return true;
```

```
    else return false;
}


void ofApp::open_setting()
{
    kp_vxvy_control = 0.4;
    ki_vxvy_control = 0.0;
    kd_vxvy_control = 0.1;

    kp_w_control = 0.7;
    ki_w_control = 0.0;
    kd_w_control = 0.0;

}

void ofApp::set_kecepatan(signed char _vx, signed char _vy,
signed char _vw, signed int muka_robot, char mode_gyro)
{
    vx = _vx;
    vy = _vy;
    w = _vw;
    gyro_status = mode_gyro;
    front = muka_robot;
}

void ofApp::simulasi(){

    difX = ballX - posX,
    difY = ballY - posY;
    //  update ball position
    posX += difX * 0.05;
    posY += difY * 0.05;

    //sudut
    muka1 = point2angle(robotY,robotX,robotY+180, robotX);
    sudut1 = point2angle(robotY,robotX,yt,xt);
    sudut2 = point2angle(robotY,robotX,ballY,ballX);
    sudut3 =
point2angle(robotY,robotX,titikTuju_y,titikTuju_x);

    sudut = sudut2 - sudut1;
    sudut_target = sudut3 - sudut1;
    muka_robot = muka1 - sudut1;


    if (sudut < 0) sudut = 360 + sudut;
    if (sudut_target < 0) sudut_target = 360 + sudut_target;
```

```
    if (muka_robot < 0) muka_robot = 360 + muka_robot;

    sudut = 360-sudut;
    sudut_target = 360-sudut_target;
    jarak = pythagoras(robotY,robotX,ballY,ballX);
    jarak_target=
pythagoras(robotY,robotX,titikTuju_y,titikTuju_x);

    if(robotY < 58 ){
        robotY= 58;
        // speedX *= -1;
    } else if(robotY > 958){
        robotY = 958;
        //speedX *= -1;
    }

    if(robotX < 58 ){
        robotX = 58;
    } else if(robotX > 658){
        robotX = 658;
    }

    xt = robotX + 100 * sin(-rotation*PI/180);
    yt = robotY + 100 * cos(-rotation*PI/180);

    switch(motion){
        case 0:
            vxvy_control(robotX, robotY, lapX , lapY,
temp_vx, temp_vy, 40, 5);
            w_control(rotation, 0, temp_vw, 5);
            break;
        case 1 : //--- KEJAR BOLA
            if(i!=0){
                chase_ball(jarak, sudut, 100, 0, 20, 40, 20,
temp_vy, temp_vw, temp_o);}
            break;
        case 2 : //--- PATH PLANNING
            if(i!=0){
                if ( status_path == TRUE){
                    if(titikTuju_x == ballX && titikTuju_y
== ballY){
                        chase_ball(jarak_target,
sudut_target, 100, 0, 20, 30, 20, temp_vy, temp_vw, temp_o);
                    }

                    else {
                        if(jarak >150){
```

```
                            chase_ball(jarak_target,
sudut_target, 100, 0, 40, 40, 40, temp_vy, temp_vw, temp_o);
                            temp_vy=30;
                        }
                        else{
                            chase_ball(jarak_target, sudut,
100, 0, 20, 40, 20, temp_vy, temp_vw, temp_o);
                            temp_vy = jarak - 130;
                            if(temp_vy<20){
                                temp_vy = 20;
                            }
                        }
                    }
                    hitsy = (robotY - 58) / 10;
                    hitsx = (robotX - 58) / 10;
                    hitgy = (posY - 58) / 10;
                    hitgx = (posX - 58) / 10;

////////////////////////////////////////// CARI JALUR
                    aktif.LuasLapangan({ 90, 60 });

aktif.setHeuristik(AStar::Heuristik::euclidean);
                    aktif.pergerakanDiagonal(true);

                    auto jalur = aktif.cariJalur({ hitsy,
hitsx }, { hitgy, hitgx });
                    line.clear();
                    for (auto& koordinat : jalur) {

                        pt.set(((koordinat.x * 10 + 5) +
58), ((koordinat.y * 10 + 5) + 58));
                        line.addVertex(pt);

                        msy[cnt] = koordinat.x *10 + 63;
                        msx[cnt] = koordinat.y *10 + 63;
                        cnt++;
                    }

                    if (cnt<=10){
                        titikTuju_x = ballX;    //next step
                        titikTuju_y = ballY;
                    }
                    else{
                        titikTuju_x = msx[cnt - 5];
                        titikTuju_y = msy[cnt - 5];
                    }
                    if(hitsy==hitgy && hitsx==hitgx)break;
                    cnt=0;
```

```
                }
            }

            break;

        case 3 :
            if(vxvy_control(robotX, robotY, 150+lapX ,
250+lapY, temp_vx, temp_vy, 30, 5))motion = 31;
            break;
        case 4:
            if(vxvy_control(robotX,robotY,240+lapX ,
20+lapY,temp_vx,temp_vy,40,10 ))motion = 41;
            break;
        case 41:
            if(vxvy_control(robotX,robotY,240+lapX ,
700+lapY,temp_vx,temp_vy,40,10 ))motion = 4;
            break;
        case 31 :
            if(vxvy_control(robotX, robotY, 150+lapX ,
250+lapY, temp_vx, temp_vy, 30, 5))motion = 32;
            break;
        case 32:
            if(vxvy_control(robotX, robotY, 450+lapX ,
250+lapY, temp_vx, temp_vy, 30, 5))
                motion = 33;
            break;
        case 33:
            if(vxvy_control(robotX, robotY, 450+lapX
,650+lapY, temp_vx, temp_vy, 30, 5))
                motion = 34;
            break;
        case 34:
            if(vxvy_control(robotX, robotY, 150+lapX ,
650+lapY, temp_vx, temp_vy, 30, 5))
                motion = 31;
            break;

    }
    /////////////////////////// update posisi
speedY = (float)temp_vy * 0.093;
    speedX = (float)temp_vx * 0.093;
    rotationSpeed = (float)temp_vw *0.093;

    if(gyro_status == TRUE){
        robotX+=speedX;
        robotY+=speedY;
    }
else{
```

```
        robotX += - speedY * sin((rotation + temp_o) * PI /
180) + speedX * cos((rotation + temp_o) * PI / 180);
        robotY += speedY * cos((rotation + temp_o) * PI /
180) + speedX * sin((rotation + temp_o) * PI / 180);
    }
    rotation += rotationSpeed;
    if (rotation >=360){
        rotation = 0;
    }

    if (rotation < 0 ){
        rotation = 360;
    }
}
```

Foto Tim Al-Jazari KRSBI Beroda ITS

Halaman ini sengaja dikosongkan

BIODATA PENULIS

Aulia Aditya Rachman, lahir di Tangerang pada 16 Januari 1994, yang merupakan anak kedua dari dua bersaudara dari pasangan Giri Umbaran dan Endang Wijayanti. Penulis menyelesaikan pendidikan dasar di SD Negeri Taman Cibodas Tangerang (1999-2005) dan dilanjutkan dengan pendidikan menengah di SMP Negeri 2 Tangerang (2005-2008) dan SMA Negeri 2 Tangerang (2008-2011). Setelah lulus dari SMA, penulis melanjutkan pendidikan di program Diploma 3 Teknik Elektro Universitas Gadjah Mada Yogyakarta (2011-2014) dan mengambil bidang konsentrasi elektronika. Pada tahun 2015, penulis memulai pendidikan Sarjana di Departemen Teknik Elektro, Fakultas Teknologi Elektro, Institut Teknologi Sepuluh Nopember (ITS) Surabaya melalui program lintas jalur dan mengambil bidang konsentrasi elektronika. Selama kuliah, penulis aktif sebagai tim robotika ITS.

Email:
a.adityarachman@gmail.com

Halaman ini sengaja dikosongkan