TUGAS AKHIR – TE141599

DESAIN DAN IMPLEMENTASI PENGUKURAN POSISI BOLA MENGGUNAKAN KAMERA 360 DERAJAT PADA ROBOT SEPAK BOLA

Yohan Prakoso
NRP 2213100148

Dosen Pembimbing
Dr. Ir. Djoko Purwanto, M.Eng.
Dr. Ir. Hendra Kusuma, M.Eng.Sc.

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

FINAL PROJECT – TE141599

DESIGN AND IMPLEMENTATION OF BALL POSITION MEASUREMENT USING 360 DEGREE CAMERA ON SOCCER ROBOT

Yohan Prakoso
NRP 2213100148

Supervisor
Dr. Ir. Djoko Purwanto, M.Eng.
Dr. Ir. Hendra Kusuma, M.Eng.Sc.

ELECTRICAL ENGINEERING DEPARTMENT
Faculty of Electrical Technology
Sepuluh Nopember Institute of Technology
Surabaya 2017

PERNYATAAN KEASLIAN
TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "Desain Dan Implementasi Pengukuran Posisi Bola Menggunakan Kamera 360 Derajat Pada Robot Sepak Bola" 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, 4 Juli 2017

Yohan Prakoso
NRP. 2213100148

DESAIN DAN IMPLEMENTASI PENGUKURAN POSISI BOLA MENGGUNAKAN KAMERA 360 DERAJAT PADA ROBOT SEPAK BOLA

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. Djoko Purwanto, M.Eng.	Dr. Ir. Hendra Kusuma, M.Eng.Sc.
NIP. 196512111990021002	NIP. 196409021989031003

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

SURABAYA
JULI, 2017

DESAIN DAN IMPLEMENTASI PENGUKURAN POSISI BOLA MENGGUNAKAN KAMERA 360 DERAJAT PADA ROBOT SEPAK BOLA

Nama : Yohan Prakoso
Pembimbing I : Dr. Ir. Djoko Purwanto, M.Eng.
Pembimbing II : Dr. Ir. Hendra Kusuma, M.Eng.Sc.

ABSTRAK

Informasi posisi bola terhadap robot adalah salah satu hal penting dalam robot sepak bola untuk digunakan oleh robot sebagai referensi menentukan pergerakan baik secara individu ataupun tim. Informasi posisi bola dapat diperoleh dari pengolahan citra yang diperoleh dari sensor *imaging*.

Pada tugas akhir ini kamera 360 derajat/omnivision digunakan sebagai sensor *imaging*. Citra kamera omnivision diciptakan dengan menggunakan kamera webcam dengan tambahan lensa *fisheye* dengan sudut pandang lebar yang dipasang dibagian atas robot. Distorsi yang dihasilkan oleh lensa kamera menyebabkan pengukuran jarak terukur dengan satuan pixel dari citra tidak linear terhadap jarak sesungguhnya yang terukur terhadap robot. Metode Jaringan Saraf Tiruan digunakan untuk mendapatkan karakteristik dari jarak yang diperoleh dengan satuan pixel pada citra terhadap jarak sesungguhnya di lapangan. Setalah didapatkan karakteristik dari kamera, hal itu digunakan sebagai dasar pengukuran posisi bola terhadap robot.

Dari hasil pengujian didapatkan bahwa sistem kamera 360 derajat mampu mendeteksi bola pada jarak 40-400 cm. Namun pada posisi tertentu hanya mampu mendeteksi hingga jarak 160 cm dikarenakan resolusi citra yang memanjan dan terbatas pada sisi atas dan bawah. Selain itu sisitem ANN dapat diterapkan dan mampu mengikuti karakteristik dari hubungan jarak terukur dalam pixel dan jarak sesungguhnya dengan rata-rata kesalahan relatif sebesar 2.52 %.

Kata kunci: omnivision, jaringan saraf tiruan, pengukuran posisi

Halaman ini sengaja dikosongkan

DESIGN AND IMPLEMENTATION OF BALL POSITION MEASUREMENT USING 360 DEGREE CAMERA ON SOCCER ROBOT

Name : Yohan Prakoso
Supervisor : Dr. Ir. Djoko Purwanto, M.Eng.
Co-Supervisor : Dr. Ir. Hendra Kusuma, M.Eng.Sc.

ABSTARCT

Position information of the ball to the robot is one of the important things in soccer robot to be used by the robot as reference determine movement either individually or team. Position information of the ball can be obtained from image processing from image that taken by imaging sensor.

In this final project 360 degree/omnivision camera is used as imaging sensor. Image of omnivision camera is created by use webcam and additional fisheye lense with width angle placed on top of the robot. Artificial Neural Network method is used to get characteristic distance in pixel unit from image to real distance in field. After get the characteristic of the camera, that is used as basic of ball position measurement to the robot.

From experiment we can obtain that 360 degree system camera can detect ball in range 40-400 cm. But in some position the system just can detect ball up to 160 cm because wide resolution of image that have limitation at bottom and top of image. Furthermore ANN system can be used and can follow characteristic of distance measured in pixel and the real distance with average of error relative is 2.52%

Keyword : omnivision, artificial neural network, position measurement

Halaman ini sengaja dikosongkan

KATA PENGANTAR

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

Desain Dan Implementasi Pengukuran Posisi Bola Menggunakan Kamera 360 Derajat Pada Robot Sepak Bola

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

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

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

1. Bapak, Ibu, adik serta seluruh keluarga yang memberikan dukungan baik moril maupun materiil.
2. Dr. Ir. Djoko Purwanto, M.Eng. selaku dosen pembimbing 1 atas bimbingan dan arahan selama penulis mengerjakan tugas akhir ini.
3. Dr. Ir. Hendra Kusuma, M.Eng.Sc. 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 dari semua pihak sangat membantu penulis untuk pengembangan lebih lanjut.

Terakhir, penulis berharap Tugas Akhir ini dapat memberikan manfaat bagi banyak pihak. Penulis juga berharap Tugas Akhir ini dapat membantu pengembangan aplikasi *image processing* dan Robotika.

Surabaya, 12 Juli 2017

Penulis

Halaman ini sengaja dikosongkan

DAFTAR ISI

ABSTRAK ..... **i**
ABSTARCT ..... **iii**
KATA PENGANTAR ..... **v**
DAFTAR ISI ..... **vii**
CONTENS ..... **xi**
DAFTAR GAMBAR ..... **xv**
DAFTAR TABEL ..... **xix**
BAB 1 ..... **1**
1.1 Latar Belakang 1
1.2 Perumusan Masalah 2
1.3 Batasan Masalah 2
1.4 Tujuan 2
1.5 Metodologi 2
1.6 Sistematika Penulisan 3
1.7 Relevansi dan Manfaat 4
BAB 2 ..... **5**
2.1 Citra Digital 5
2.1.1 Citra RGB 7
2.1.2 Citra HSV 8
2.1.3 Citra Biner 9
2.2 OpenCV 10
2.2.1 Transformasi Morphology 11
2.2.1.1 Dilation 11
2.2.1.2 Erosion 11
2.3 Kamera Omnidireksional 12
2.4 Jaringan Saraf Tiruan 13
2.5 Back propagation 13
2.5.1 Prosedur inisialisasi 13
2.5.2 Prosedur Feedforward 14
2.5.3 Error Back Propagation 14
2.6 Regresi Polinomial 16
2.7 Open Framework 16
2.8 Baterai Lipo 17
2.9 STM32f4 Discovery 18
2.10 Rotary Encoder 18
2.11 Mini PC 19
2.12 Driver Motor 20

2.13 Motor DC 21
BAB 3 23
3.1 Vision 23
3.1.1 Pengambilan frame Citra (Image Acquisition) 25
3.1.2 Konversi citra RGB ke HSV(RGB to HSV) 27
3.1.3 Pendeteksian Bola(Ball detection) 28
3.1.3.1 Konvert ke citra biner 29
3.1.3.2 Proses morfologi 30
3.1.3.3 Pendeteksian objek bola 33
3.1.4 Penentuan Posisi bola pada citra 35
3.1.5 Pengukuran Posisi bola terhadap robot 36
3.1.6 Learning dengan menggunakan backpropagation .. 36
3.1.6.1 Inisialisasi nilai w dan b 37
3.1.6.2 Normalisasi nilai input dan output 38
3.1.6.3 Foward propagation 38
3.1.6.4 Backward propagation 42
3.1.6.5 Update nilai w dan b 45
3.2 Algoritma Perencanaan Rute 48
3.3 Kontrol Pergerakan 48
3.3.1 Perancangan Sistem Gerak 49
3.3.2 Perancangan Sistem Odometri 52
3.4 Desain mekanik robot 53
3.5 Desain elektronik robot 53
3.5.1 Supply 54
3.5.2 Mini PC Intel Nuc 55
3.5.3 STM32f4 56
3.5.4 USB to TTL 58
3.5.5 Arduino Nano 59
3.5.6 Sensor IMU 59
3.5.7 Sensor garis 60
3.5.8 Regulator Sistem Mikrokontroler dan sensor 61
3.5.9 Buck Converter untuk Mini PC 62
3.5.10 Driver Motor 63
3.5.11 Rotary encoder 64
3.5.12 Sensor IR proximity 65
BAB 4 67
4.1 Uji pendeteksian bola 67
4.2 Pengujian hubungan jarak pixel dan jarak sesungguhnya ... 68
4.3 Pengukuran Jarak Dengan Sistem ANN 70

BAB 5 ..... **75**
5.1 Kesimpulan 75
5.2 Saran 75
DAFTAR PUSTAKA **77**
LAMPIRAN **79**
BIODATA PENULIS **97**

Halaman ini sengaja dikosongakan

CONTENS

ABSTRACT (INDONESIA) ..... *Error! Bookmark not defined.*
ABSTRACT (ENGLISH) ..... *Error! Bookmark not defined.*
PREFACE ..... *Error! Bookmark not defined.*
CONTENS (INDONESIA) ..... ***vii***
CONTENS (ENGLISH) ..... ***xi***
LIST OF FIGURE ..... ***xv***
LIST OF TABLE ..... ***xix***
CHAPTER 1 ..... ***1***
1.1 Background *1*
1.2 Perumusan Masalah *2*
1.3 Scope of Problem *2*
1.4 Objective *2*
1.5 Methodology *2*
1.6 Writing System *3*
1.7 Relevance and Benefit *4*
CHAPTER 2 ..... ***5***
2.1 Digital Image *5*
2.1.1 RGB Image *7*
2.1.2 HSV Image *8*
2.1.3 Biner Image *9*
2.2 OpenCV *10*
2.2.1 Morphology Transformation *11*
2.2.1.1 Dilation *11*
2.2.1.2 Erotion *11*
2.3 Omnidirectional Camera *12*
2.4 Artificial Neural Network *13*
2.5 Back propagation *13*
2.5.1 Initialization Procedure *13*
2.5.2 Feedforward Procedure *14*
2.5.3 Back Propagation Error *14*
2.6 Polynomial Regression *16*
2.7 Open Framework *16*
2.8 Lipo Battery *17*
2.9 STM32f4 Discovery *18*
2.10 Rotary Encoder *18*
2.11 Mini PC *19*
2.12 Motor Driver *20*
2.13 Motor DC *21*

CHAPTER 3 ..... ***23***
3.1 Vision *23*
3.1.1 Image Acquisition *25*
3.1.2 RGB to HSV Convertion *27*
3.1.3 Ball detection *28*
3.1.3.1 Convert to Biner Image *29*
3.1.3.2 Morphology Process *30*
3.1.3.3 Ball Object Detection *33*
3.1.4 Determination of the ball position on the image *35*
3.1.5 Ball Position to Robot Measurement *36*
3.1.6 Learning using backpropagation *36*
3.1.6.1 w and b value initialization *37*
3.1.6.2 Input and Output Normalization *38*
3.1.6.3 Foward propagation *38*
3.1.6.4 Backward propagation *42*
3.1.6.5 w and b update value *45*
3.2 Route planning algorithm *48*
3.3 Motion Control *48*
3.3.1 Design of Motion System *49*
3.3.2 Design of Odometry Sistem *52*
3.4 Mechanical Design of Robot *53*
3.5 Electronic Design of Robot *53*
3.5.1 Supply *54*
3.5.2 Intel Nuc Mini PC *55*
3.5.3 STM32f4 ..... *56*
3.5.4 USB to TTL *58*
3.5.5 Arduino Nano *59*
3.5.6 IMU Sensor *59*
3.5.7 Line Sensor *60*
3.5.8 Regulator of Microcontroller and Sensor System *61*
3.5.9 Buck Converter for Mini PC *62*
3.5.10 Motor Driver *63*
3.5.11 Rotary encoder *64*
3.5.12 IR proximity Sensor *65*
CHAPTER 4 ..... ***67***
4.1 Detection Ball Testing *67*
4.2 Pixel Distance and Real Distance Testing *68*
4.3 ANN System Distance Measurement *70*
CHAPTER 5 ..... ***75***

5.1 Conclusion *75*
5.2 Suggestion *75*
REFERENCES ***77***
ATTACHMENT ***79***
AUTHOR BIODATA ***97***

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

Gambar 2.1 Koordinat 2D pada sebuah citra digital 5
Gambar 2.2 Konsep Pixel [2]..... 6
Gambar 2.3 Merah, Hijau dan biru adalah warna dasar yang dapat menghasilkan warna-warna lain..... 7
Gambar 2.4 Geometri Silinder Citra Warna HSV..... 8
Gambar 2.5 Citra warna biner..... 9
Gambar 2.6 Logo OpenCV [5]..... 10
Gambar 2.7 Proses *dilation* [2] 11
Gambar 2.8 Proses *Erosion* [2] 12
Gambar 2.9 Kamera Omnidireksional 12
Gambar 2.10 Logo openFrameworks..... 16
Gambar 2.11 Konfigurasi Baterai Lipo [9] 17
Gambar 2.12 STM32F4 - Discovery [10] 18
Gambar 2.13 Rotary Encoder [11] 19
Gambar 2.14 Mini PC [12]..... 20
Gambar 2.15 Rangkaian H-Bridge..... 20
Gambar 2.16 Motor DC 21
Gambar 3.1 Diagram Utama Robot 23
Gambar 3.2 Blok Diagram Vision 24
Gambar 3.3 Peletakan kamera Omnidireksional..... 25
Gambar 3.4 Citra yang dihasilkan oleh kamera omnidireksional dengan menggunakna webcam yang dipasang fish eye..... 25
Gambar 3.5 Penentuan indeks kamera pada device manager..... 26
Gambar 3.6 Hasil Pengambilan Citra..... 27
Gambar 3.7 Hasil konversi (a) RGB ke (b) HSV..... 28
Gambar 3.8 Blok Diagram Pendeteksian Bola..... 29
Gambar 3.9 Hasil *thresholding* bola 30
Gambar 3.10 Hasil *thresholding* lapangan..... 30
Gambar 3.11 Perbandingan (a) citra biner asli dan (b) hasil erotion dengan kernel elips 2 px 32
Gambar 3.12 Perbandingan (a) citra biner asli dan (b) hasil erotion dengan kernel elips 10 px 33
Gambar 3.13 Perbandingan (a) citra biner asli dan (b) hasil dilation dengan kernel elips 2 px..... 33
Gambar 3.14 Perbandingan (a) citra biner asli dan (b) hasil dilation dengan kernel elips 10 px..... 33
Gambar 3.15 Pendeteksian Bola 35

Gambar 3.16 Penentuan bagian depan kamera pada kamera omnidireksional 36
Gambar 3.17 Tampilan Software Matlab 2013 37
Gambar 3.18 Grafik *Transfer Function* Log Sigmoid 39
Gambar 3.19 Grafik *Transfer Function* Log Tangean Sigmoid 40
Gambar 3.20 Grafik *Transfer Function* Hard Limiter 40
Gambar 3.21 Grafik *Transfer Function* Linear 41
Gambar 3.22 Desain ANN yang akan diterapkan pada robot 46
Gambar 3.23 Blok Kontrol Robot 49
Gambar 3.24 Motor DC PG-45 49
Gambar 3.25 *Base* robot 50
Gambar 3.26 Roda Omni 51
Gambar 3.27 Interkoneksi antara Motor, Driver Motor dan STM32f4 . 51
Gambar 3.28 Desain base untuk rotary encoder 52
Gambar 3.29 Desain 3D Robot 53
Gambar 3.30 Blok diagram sistem elektronik robot 54
Gambar 3.31 Baterai Lipo Sebagai sumber energi utama robot 54
Gambar 3.32 Mini PC Intel NUC 5i7 KYK 55
Gambar 3.33 Diagram kontrol PID pada motor 57
Gambar 3.34 Tampilan IDE Coocox 57
Gambar 3.35 Tampilan Software ST-LINK 58
Gambar 3.36 USB to TTL 58
Gambar 3.37 Arduino nano [16] 59
Gambar 3.38 MPU-6050 dan pin outnya 60
Gambar 3.39 *Wire out* modul sensor garis 61
Gambar 3.40 Modul Sensor Garis 61
Gambar 3.41 Desain Regulator 62
Gambar 3.42 Modul Buck Converter 63
Gambar 3.43 Modul Driver Motor BTN 7970 64
Gambar 3.44 *Rotary Encoder* 65
Gambar 3.45 *Wire out* dari rotary encoder 65
Gambar 3.46 Sensor IR Proximity 66
Gambar 3.47 *Wire Out* Sensor Proximity 66
Gambar 4.1 Uji Pendeteksian Bola 67
Gambar 4.2 Citra terpotong pada bgaian atas dan bawah 68
Gambar 4.3 grafik hubungan jarak pixel dan jarak sesungguhnya 70
Gambar 4.4 Grafik perbandingan jarak terukur dan jarak sesungguhnya 72

Gambar 4.5 Grafik perbandingan jarak hasil pengukuran dan jarak sesungguhnya 73
Gambar 4.6 Grafik error relatif terhadap jarak sesungguhnya 73

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Tabel 4.1 Hubungan jarak pixel dan jarak sesungguhnya 69
Tabel 4.2 Hasil Uji ANN 70

Halaman ini sengaja dikosongkan

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Robot adalah alat yang dapat membantu banyak kebutuhan manusia secara fisik baik dengan kotrol oleh manusia ataupun secara otomatis dengan program yang telah ditanamkan terlebih dahulu pada robot. Untuk memacu pengembangan pengethuan tentang robot, ada banyak sekali kontes yang diadakan setiap tahunnya dengan tema-tema yang beraneka ragam. Salah satu kontes yang tersebut adalah Kontes Robot Sepak Bola Indonesia kategori beroda.

Salah satu tantangan dari robot ini adalah mentukan posisi bola dan objek lain terhadap robot. Kamera adalah alat yang sering digunakan untuk mendapatkan citra dari lingkuangan dan mengidentifikasi posisi dari bola. Penggunaan kamera tunggal membuat sudut pandang dari robot terhadap lingkungan sekitar terbatas sehingga diperlukan bantuan gerakan mekanik untuk membantu robot mendapatkan pandangan terhadap lingkungan yang lebih luas dan mendapatkan objek yang diinginkan baik berupa bola, gawang, kawan dan lawan untuk lebih lanjut diproses menjadi gerakan robot yang harus dilakukan.

Dengan menggunakan kamera dengan sudut pandang 360 derajat tentunya akan membuat sudut pandang robot terhadap lingkungan semakin luas sehingga tidak diperlukan lagi terlalu banyak gerakan mekanik untuk mencari bola sehingga robot dapat lebih cepat untuk menentukan keputusan selanjutnya.

Kamera akan menghasilkan citra lingkungan yang jika diproses lebih lanjut akan didapatkan posisi bola pada citra. Posisi bola pada citra dapat di representasikan dengan x, y dan jari-jari dari bola dengan satuan pixel. Namun posisi x, y, dan jari-jari bola pada citra tidak linear terhadap posisi bola sesungguhnya terhadap robot sehingga diperlukan proses pengolahan lebih lanjut untuk mendapatkan posisi bola terhadap robot. Beberapa metode yang dapat digunakan adalah Jaringan Saraf tiruan atau regresi polinomial.

Jaringan Syaraf Tiruan adalah sebuah model matematik yang berupa kumpulan unit yang terhubung secara paralel yang bentuknya menyerupai jaringan saraf pada otak manusia. Dengan menggunakan metode ini kita dapat menggunakan beberapa input data dan output yang seharusnya untuk dijadikan sebagai data pembelajaran. Dari data-data yang telah

diberikan, sistem ini dapat menentukan secara mandiri output untuk input-input lain. Dengan demikian, menggunakan metode ini dapat digunakan untuk mengatasi permasalahan ketidak linearan antara posisi objek pada citra dengan posisi objek pada lapangan sebenarnya terhadap robot.

Selain itu Penggunaan regresi polinomial juga dapat digunakan untuk mengidentifikasi jarak bola terhadap robot khususnya pada penggunaan sistem katadioptrik. Regresi Polinomial merupakan model regresi linear yang dibentuk dengan menjumlahkan pengaruh masing-masing peubah penjelas yang dipangkatkan meningkat sampai order ke-m.

1.2 Perumusan Masalah

Permasalahan pada penelitian ini adalah sebagai berikut:

1. Bagaimana pendeteksian bola dengan menggunakan kamera 360 derajat pada robot sepak bola?
2. Bagaimana karakteristik hubungan jarak pada pixel kamera terhadap jarak sesungguhnya di lapangan?
3. Apakah sistem ANN(*Artificial Neural Network)* dapat diterapkan untuk melakukan pengukuran posisi bola terhadap robot?

1.3 Batasan Masalah

Batasan masalah pada tugas akhir ini adalah sebagai berikut:

1. Objek yang di ukur posisinya hanyalah berupa bola dengan warna oranye.
2. Posisi bola dinyatakan dalam bidang cartesian 2 dimensi relatif terhadap robot sebagai titik origin.

1.4 Tujuan

Tujuan dari pembuatan tugas akhir ini adalah sebagai berikut:

1. Merancang dan menguji pendeteksian bola dengan menggunakan kamera 360 derajat.
2. Bagaimana karakteristik hubungan jarak pada pixel kamera terhadap jarak sesungguhnya di lapangan
3. Menguji sistem ANN(*Artificial Neural Network)* untuk digunakan sebagai metode pengukuran posisi bola terhadap robot sepak bola.

1.5 Metodologi

Metodologi yang digunakan pada penelitian tugas akhir ini adalah sebagai berikut:

1. **Studi Literatur**

   Pada tahap studi literatur dilakukan pengumpulan dasar teori yang menunjang dalam penulisan tugas akhir. Dasar teori tersebut diambil dari artikel-artikel di internet dan forum-forum diskusi.

2. **Perancangan Sistem**

   Robot di desain untuk dapat memahami lingkuangan sekitar terutama mengetahui jarak objek-objek baik berupa musuh ataupun bola dari robot tersebut. Pada bagian ini akan dibahas bagaimana perancangan dari sistem robot yang akna dibuat mulai dari blok robot secara keseluruhan hingga isi per blok diagram dari robot. Pembahasan perancangan akan lebih menitik beratkan pada bagian *vision* yaitu blok dimana tugas akhir ini diterapkan.

3. **Pengujian dan Penyempurnaan Sistem**

   Pada tahap ini, dilakukan pengujian terhadap sistem pendeteksian posisi objek terhadap robot dan membandingkan posisi objek tersebut terhadap robot sesungguhnya untuk selanjutnya dilakukan evaluasi dan perbaikan.

4. **Penulisan Laporan Tugas Akhir**

   Tahap penulisan laporan tugas akhir dilakukan beriringan pengerjaan.

1.6 Sistematika Penulisan

Dalam buku tugas akhir ini, pembahas mengenai sistem yang dibuat dibahas dalam 5 bab dengan sistematika penulisan sebagai berikut:

- BAB 1: PENDAHULUAN

  Pada Bab ini berisi hal-hal meliputi latar belakang, perumusan masalah, batasan masalah, tujuan, metodelogi, sistematika penulisan serta relevansi dan manfaat.
- BAB 2: TINJAUAN PUSTAKA DAN TEORI PENUNJANG

  Bab ini berisi tentang teori penunjang sera literatur dari berbagai sumber baik dari jurnal, buku, atau internet yang dapat menunjang pengerjaan tugas akhir ini.
- BAB 3: PERANCANGAN SISTEM

Bab ini menjelaskan tentang perencanaan sistem baik disisi *hardware* ataupun *software* untuk pengukuran posisi bola terhadap robot.

- BAB 4: PENGUJIAN

  Bab ini berisi tentang pengujian sistem yang telah dibuat beserta analisa dari hasil pengujian tersebut.

- BAB 5: PENUTUP

  Bab ini adalah bagian yang berisikan kesimpulan yang diperoleh dari pengerjaan tugas akhir serta saran-saran untuk pengembangan riset lebih lanjut.

1.7 Relevansi dan Manfaat

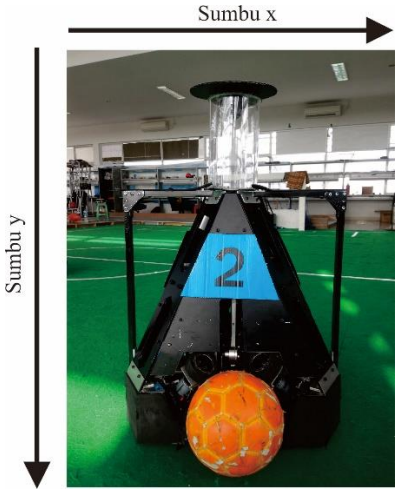
Pengerjaan tugas akhir ini diharapkan dapat digunakan pada robot untuk kompetisi Kontes Robot Sepak Bola Indonesia kategori beroda atau pun lomba sejenis baik tingkat nasional ataupun Internasonal seperti Robocup. Selain itu tidak menutup kemungkinan hasil dari tugas akhir ini dapat digunakan secara luas pada robot yang menggunakan kamera untuk mendeteksi objek dan memerlukan penentuan posisi objek tersebut.

BAB 2
TINJAUAN PUSTAKA DAN TEORI PENUNJANG

2.1 Citra Digital

Untuk menampilkan sebuah citra digital pada sebuah layar diperlukan proses digitalisasi yang terdiri dari proses sampling dan kuantisasi. Sampling membagi gambar menjadi kotak-kotak kecil sedangkan proses kuantisasi mengisi kotak-kotak kecil hasil sampling dengan nilai tertentu.

Sebuah citra digital disajikan oleh sekumpulan titik yang masing-masing di representasikan oleh posisi (x dan y) serta sebuah nilai [1]. Lebih lanjut titik-titik ini lebih dikenal dengan *pixel.* Oleh karena itu sebuah pixel dapat dinyatakan oleh fungsi dua dimensi $f(x, y)$ dengan titik origin berada pada pojok kiri atas citra. Sumbu x positif melintang ke arah kanan dan sumbu y positif membujur ke arah bawah. Koordinat pixel pada sebuah citra diilustrasikan pada Gambar 2.1. Dimisalkan sebuah pixel dinyatakan sebagai $f(10,5) = 127$ hal itu berarti pixel tersebut berada pada posisi 10 pixel dari kiri, 5 pixel dari atas dan memiliki nilai kecerahan sebesar 127.

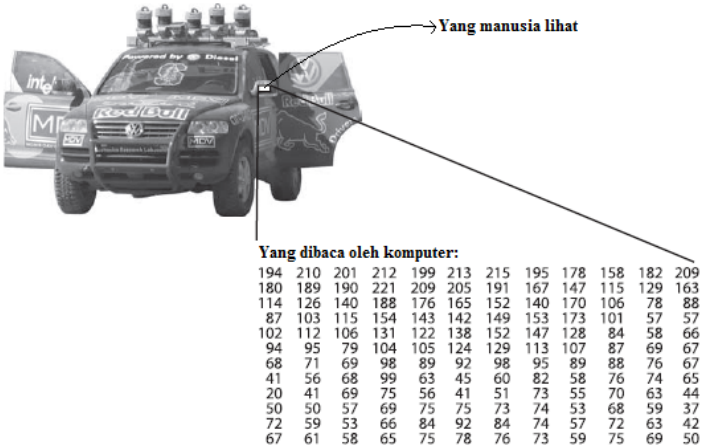


Gambar 2.1 Koordinat 2D pada sebuah citra digital

Sekumpulan pixel-pixel akan membentuk sebuah matrix dengan masing-masing memiliki sebuah nilai tertentu. Variasi nilai dari setiap matrix akan menciptakan sebuah citra digital yang dapat di lihat oleh manusia. Konsep ini dapat dilihat pada Gambar 2.2 dibawah.

Kedalaman warna warna sebuah citra menunjukkan variasi kecerahan yang mampu disajikan oleh sebuah citra digital untuk setiap pixel yang dinyatakan dalam n bit. Misalkan sebuah citra memiliki kedalaman sebesar 8 bit maka setiap pixel dari citra tersebut mampu menyajikan 2^8 atau setara dengan 255 variasi kecerahan. Demikian pula jika sebuah citra digital memiliki kedalaman sebesar 16 bit maka setiap pixel dari citra tersebut mampu menghasilkan 2^{16} atau setara dengan 65535 variasi kecerahan. Semakin tinggi nilai kedalaman warna dari sebuah citra maka semakin tajam dan detail citra tersebut dapat menyajikan warna kepada manusia.

Dalam pengolahan citra (*image processing*) terdapat beberapa jenis citra yang digunakan yaitu citra warna, citra *grayscale* (abu-abu), dan Citra biner. Ketiga jenis citra tersebut akan dijelaskan lebih lanjut berikut ini.

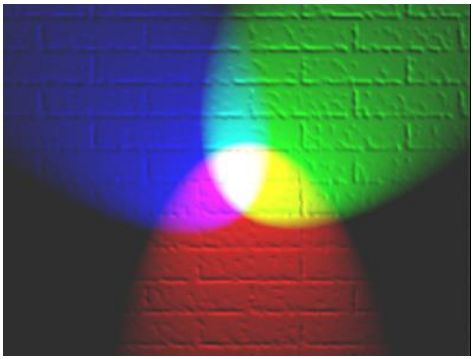


Gambar 2.2 Konsep Pixel **[2]**

2.1.1 Citra RGB

Citra RGB dihasilkan oleh 3 kanal warna utama yaitu merah(*Red*), hijau(*hijau*), dan biru(*blue*). Ketiga kanal tersebut biasa disingkat dengan RGB atau BGR yang masing-masing huruf merupakan singkatan dari warna dari masing-kanal tersebut yaitu *Red*, *Green*, dan *Blue*. Ketiga warna tersebut adalah warna dasar yang dapat merepresentasikan warna-warna lain [3]. Dapat dilihat pada gambar 3 bahwa kombinasi dari 2 atau lebih warna cahaya dapat menghasilkan warna lain. Oleh karena itu sebuah warna pada sebuah pixel dapat dinyatakan dengan kombinasi tingkat kecerahan dari masing-masing kanal.

Seperti penjelasan sebelumnya dijelaskan bahwa dalam merepresentasikan tingkat kecerahan suatu matrix kanal dapat dinyatakan dalam sebuah tingkat kedalaman warna yang dinyatakan dalam n – bit. 3 kanal warna dengan kedalaman warna n bit dapat menghasilkan variasi warna sebanyak $(2^n)^3$ sehingga untuk sebuah citra RGB dengan kedalaman warna 8 bit akan memiliki variasi warna sebanyak 16581375 warna dan citra RGB dengan kedalam warna 16 bit kan memiliki variasi warna sebesar 281474976710656 warna. Semakin tinggi tingkat kedalaman warna akan menghasilkan sebuah citra dengan warna yang tajam dan detail. Namun dalam pengolahan citra, Citra dengan tingkat kedalaman yang tinggi cenderung memerlukan waktu pemrosesan yang lebih lama sehingga untuk pemrosesan citra yang membutuhkan waktu yang lebih cepat, Citra dengan kedalaman warna yang lebih rendah



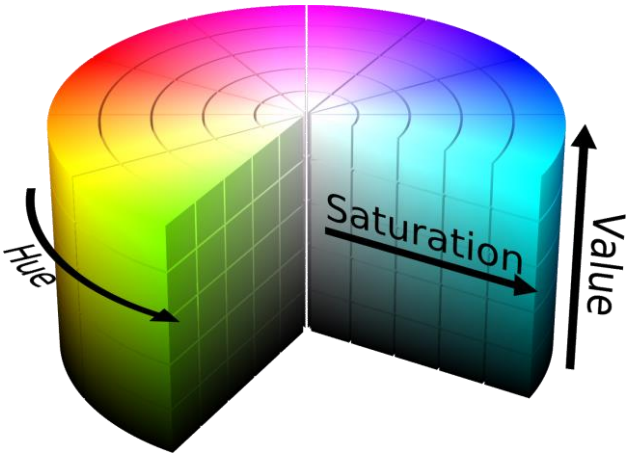
Gambar 2.3 Merah, Hijau dan biru adalah warna dasar yang dapat menghasilkan warna-warna lain

cenderung lebih baik selama informasi warna yang dibutuhkan sudah cukup terpenuhi.

2.1.2 Citra HSV

Citra HSV yang merupakan singkatan dari *Hue*, *Saturation*, dan *Value* adalah citra warna RGB yang diatur ulang dan direpresentasikan dalam geometri warna HSV. Hal ini diutujukan agar lebih intuitif dan lebih relevan dari pada representasi warna menggunakan koordinat cartesian kubus dari RGB. Citra warna HSV direpresentasikan dalam geometri silinder seperti ditunjukkan pada Gambar 2.4. *Hue* dinyatakan pada sudut silinder, *saturation* dinyatakan pada arah jari-jari tabung dan *value* dinyatakan oleh tinggi silinder.

Citra HSV banyak digunakan pre proses dikarenakan citra HSV memisahkan antara *luma* atau intensitas citra dari *chroma* atau informasi warna. Sangat banyak manfaat dari penggunaan citra warna ini. Misalnya saat kita ingin melakukan ekualisasi pada histogram warna tertentu semisal menaikkan intesitas warna tersebut. Selain itu penggunaan citra warna ini memungkinkan kita melakukan seleksi warna sebuah tertentu tanpa melihat intensitas cahaya yang jatuh pada objek tersebut. Hal ini sangat berguna terutama pada seleksi warna objek yang berada diluar ruangan dengan intensitas cahaya yang beraneka ragam.



Gambar 2.4 Geometri Silinder Citra Warna HSV

2.1.3 Citra Biner

Citra biner adalah citra yang hanya memiliki 2 nilai yaitu 1 dan 0 yang berarti hitam atau putih [4]. Pada penerapannya citra biner digunakan untuk memisahhkan antara objek yang ingin diolah lebih lanjut terhadapa background yang ingin diabaikan. pada umumnya objek ditandai dengan warna putih dan *background* ditandai dengan warna hitam. Contoh dari citra warna biner ditunjukkan pada Gambar 2.5

Citra warna biner diperoleh salah satunya dengan *thresholding*. *Thresholding* adalah sebuah metode untuk memisahkan sebuah nilai yang berada di atas dan dibawah sebuah batas tertentu dan dibedakan dengan nilai 1 dan 0. Semisal sebuah batas *theshold* bernilai 200, pixel yang memiliki nilai lebih dari 200 akan diberikan nilai 1 atau 255 (putih) sedangkan pixel yang memiliki nilai kurang dari atau sama dengan 200 akan diberi nilai 0 (hitam).

Citra biner banyak digunakan terutama dalam pemrosesan citra meskpun banyak orang lebih menyenangi warna biner ataupun grayscle. Hal ini dikarenakan pemrosesan citra biner membutuhkan waktu yang jauh lebih cepat dibandingkan citra warna yang lain.

Gambar 2.5 Citra warna biner

2.2 OpenCV

OpenCV adalah sebuah *library open source* untuk vision komputer yang disediakan pada link *http://SourceForge.net/projects/opencvlibrary*. Library ini ditulis dalam bahasa C dan C++ dan dapat dijalankan pada sistem operasi Linux,Windows dan Mac OS X. Selain itu terdapat pengembangan pada Python,Ruby,Matlab, dan bahasa pemrograman lain [2]. OpenCV adalah singkatan dari *Open Source Computer Vision*. OpenCV merupakan sebuah *software library* bebas (*open source*) yang digunakan untuk operasi *computer vision* dan *machine learning*. OpenCV telah dibangun untuk menyediakan sebuah infrastruktur umum untuk beberapa aplikasi *computer vision* dan untuk mempercepat penggunaan dari mesin persepsi dalam produk komersial. OpenCV mempermudah bisnis-bisnis untuk memanfaatkan dan memodifikasi kode. *Library* OpenCV mempunyai lebih dari 2500 algoritma yang telah dioptimalkan dimana meliputi sebuah himpunan menyeluruh dari keduanya yaitu klasik dan seni beberapa algoritma *computer vision* dan *machine learning.* Algoritma-algoritma tersebut dapat digunakan untuk mendeteksi dan mengenali wajah, mengidentifikasi obyek, mengklasifikasi tindakan manusia dalam video, mengikuti jejak perpindahan obyek, mengekstrak model-model 3D obyek, menghasilkan titik awan 3D dari kamera stereo, dan lain sebagainya.

OpenCV dapat diterapkan pada pemrograman C++, C. Python, Java dan MATLAB. OpenCV mendukung untuk sistem operasi Windows, Linux, Android dan Mac OS.

Gambar 2.6 Logo OpenCV **[5]**

2.2.1 Transformasi Morphology

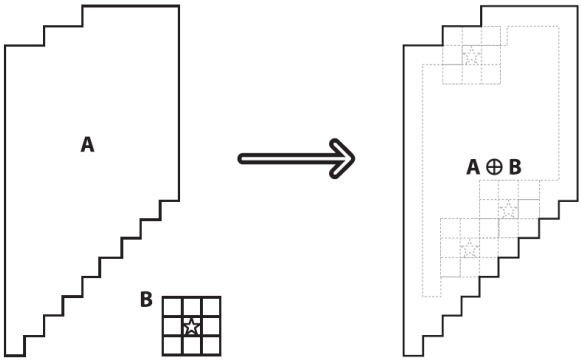
OpenCV menyediakan antar muka yang cepat dan mudah untuk melakukan transformasi morfologi. Transformasi morfologi paling dasar adalah *dilation* dan *erosion*. Transformasi geometri berguna dalam banyak hal seperti menghilangkan *noise*, mengisolasi sebuah elemen, menggabungkan beberapa elemen yang terpisah, serta mendeteksi lubang pada sebuah objek dan menjadikannya bagian dari objek tersebut [2].

2.2.1.1 Dilation

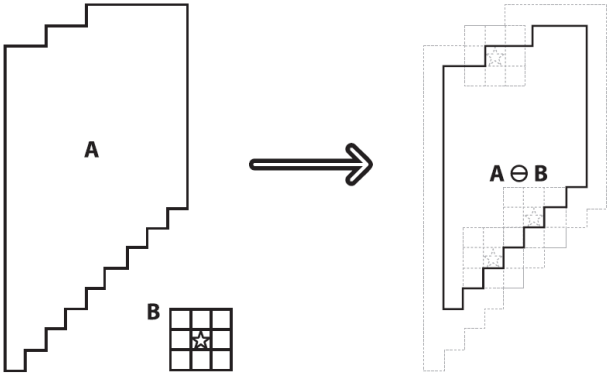
Dilation adalah konvolusi beberapa bagian dari citra yang kita asumsikan dengan A oleh sebuah *kernel* yang ita asumsikan dengan B. Pada umumnya kernel berbentuk persegi dengan ukuran tertentu. Kernel B memindai citra dan mencari nilai maksimum pada kernel dan mengganti nilai pada *anchor poin* (biasany ada di tengah kernel) dengan nilai maksimal tersebut. Proses ini menyebabkan kan daerah terah terang pada objek menjadi bertambah. Proses dari *dilation* diilustrasikan pada Gambar 2.7.

2.2.1.2 Erosion

Berlawanan dengan *Dilation, Erosion* merupakan sebuah proses konvolusi beberapa bagian dari citra yang kita asumsikan dengan A oleh sebuah kernel yang kita asumsikan dengan B. Proses konvolusinya adalah dengan mencari nilai terkecil dari area yang dilalui oleh kernel B pada citra A dan mengganti nilai pada *anchor point* dengan nilai tersebut. Proses ini menyebabkan daerah gelap pada citra akan semakin bertambah. Proses erosion diilustrasikan pada Gambar 2.8.



Gambar 2.7 Proses *dilation* [2]



Gambar 2.8 Proses *Erosion* [2]

2.3 Kamera Omnidireksional

Dalam fotografi omnidireksional kamera adalah sebuah kamera yang memiliki sudaut pandang 360 derajat pada area horizontal atau sudut pandang yang melingkupi hampir seluruh bola. Kamera omnidireksional

Gambar 2.9 Kamera Omnidireksional

digunakan pada fungsi yang memerlukan sudut pandang luas seperti pada fotografi panorama dan beberapa keperluan robotika.

Dalam robotika, kamera omnidireksional sering digunakan untuk odometri secara visual dan untuk menyelesaikan *simultaneous localization and mapping(SLAM).* Karena kemampuan kamera omnidireksional untuk mengambil gambar dengan sudut pandang 360 derajat, hasil yang lebih baik akan didapatkan untuk *optical flow* dan *feature selection and matching.*

2.4 Jaringan Saraf Tiruan

Jaringan saraf tiruan adalah sebuah bentuk proses yang menyerupai kerja dari otak manusia yang memiliki neuron dan interkoneksi. Sistem ini berperan layaknya otak manusia yaitu memiliki persepsi, mmemiliki kemampuan belajar dan pengenalan terhadapa suatu bentuk dan pola.

Elemen dasar dari sebuah sistem jjaringan saraf tiruan adalah neuron layaknya elemn dasar pada saraf manusia. Neuron merupakan model dari sebuah neuron natural. Setiap neuron memiliki x_1, x_2, …, x_m yang masuk pada neuron. Pada neoron manusia masukan-masukan ini merupakan tingkatan stimulus. Pada sistem jaringan saraf tiruan setiap masukan x_i,

memiliki sebuah bobot pengali yaitu w_i. Hasil perkalian dari $x_i w_i$ ini kemudian masuk kedalam neuron [6].

2.5 *Back propagation*

Backpropagation adalah metode pembelajaran untuk sistem jaringan saraf tiruan yang dirumuskan oleh Werbos dan dipopulerkan oleh McClelland dan Rumelhart. Algoritma ini mempuyai sitem kerja untuk mengkoreksi error yang dihasilkan oleh sebuah sistem jaringan saraf tiruan. Selanjutnya error yang diperoleh tersebut digunakan sebagai pertimbangan bagi algoritma tersebut untuk mengkoreksi bobot dengan tujuan memperkecil error yang dihasilkan. Langkah-langkah dari backpropagation adalah dijelasakan pada langkah-langkah dibawah ini

2.5.1 Prosedur inisialisasi

Langkah 0:

Inisialisasi bobot dan masukan setiap *layer* dengan menggunakan metode *random gaussian* dengan rata-rata 0 dan deviasi 0,1.

Langkah 1:

Jika kondisi berhenti tidak tercapai, maka ulangi langkah 2 – 9.

Langkah 2:

Untuk setiap pasangan pembelajaran, lakukan langkah 3 - 8.

2.5.2 Prosedur Feedforward

Langkah 3:

Setiap unit niput (X_i, i = 1,…, n) menerima sinyal input x_i dan menyebarkannya ke senua unit di *layer* di atasnya (*hidden layer*).

Langkah 4:

Setiap unit tersembunyi (Z_j, j = 1,…, p) menjumlahkan setiap sinyal input yang telah dikalikan dengan pembobotnya.

$$z\_in_j = v_{0j} + \sum_{i=1}^{n} x_i \, v_{ij} \tag{2.1}$$

Tetapkan fungsi aktivasi untuk menghitung sinyal outputnya.

$$z_j = f(z\_in_j) \tag{2.2}$$

Dan mengirimkan sinyal ini pada semua unit pada *layer* diatasnya (*layer* output).

Langkah 5:

Setiap unit output (Y_k, k = 1,…, m) menjumlahkan sinyal input berbobotnya,

$$y\_in_k = w_{0k} + \sum_{j=1}^{n} z_j w_{jk} \tag{2.3}$$

dan menetapkan fungsi aktivasi untuk menghitung sinyal outputnya.

$$y_k = f(y\_in_k) \tag{2.4}$$

2.5.3 Error Back Propagation

Langkah 6:

Setiap unit output (Y_k, k = 1,…, m) menerima pola target yang berhubungan dengan pola pelatihan input, menghitung error informasinya

$$\delta_k = (t_k - y_k) f'(y\_in_k) \tag{2.5}$$

Hitung bobot pengkoreksi (digunakan untuk memperbarui w_{jk} nanti),

$$\Delta w_{jk} = \mu \delta_k z_j \quad (2.6)$$

Menghitung pengoreksi biasnya (digunakan untuk memperbarui w_{0k} nanti)

$$\Delta w_{0k} = \mu \delta_k z_j \quad (2.7)$$

Langkah 7:

Setiap unit tersembunyi (Z_j, j = 1,…, p) menjumlahkan selisih inputnya (dari unit di layer atas)

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk} \quad (2.8)$$

Kalikan dengan derifatif dari fungsi aktivasinya untuk menghitung error informasi

$$\delta_j = \delta\_in_j f'(z\_in_j) \quad (2.9)$$

Hitung bobot pengkoreksinya (digunakan untuk meperbarui v_{ij} nanti),

$$\Delta v_{ij} = \alpha \delta_j x_i \quad (2.10)$$

Dan menghitung bias pengkoreksinya (digunakan untuk memperbarui v_{0j} nanti),

$$\Delta v_{0j} = \alpha \delta_j \quad (2.11)$$

Langkah 8:

Setiap unit output (Y_k, k = 1,…, m) memperbarui bias dan bobotnya (j = 0,. . . . , p)

$$w_{jk}(baru) = w_{jk}(lama) + \Delta w_{jk} \quad (2.12)$$

Setiap unit tersembunyi (Z_j, j = 1,…, p) memperbarui bias dan bobotnya (i = 0, ... , n):

$$v_{ij}(baru) = v_{ij}(lama) + \Delta v_{ij} \quad (2.13)$$

Langkah 9:
Test kondisi berhenti.

2.6 Regresi Polinomial

Metode regresi adalah suatu metode statistik untuk mnyelidiki dan memodelkan hubungan antara variabel respon *Y* dan variabel prediktor *X*. Misal diberikan himpunan data {(Xi,Yi)},i = 1, ...,n. Secara umum hubungan antara Y dan X dapat ditulis sebagai berikut:

$$Y_i = m(X_i) + \varepsilon_i \quad (2.14)$$

Dengan $m(X_i)$ adalah suatu fungsi regresi yang belum diketahui dan ingin ditaksir, dan ε_i adalah suatu variabel acak yang menggambarkan variasi Y di sekitar $m(X_i)$ [7].

2.7 Open Framework

OpenFramework adalah sebuah perangkat *open source C++* yang didesain untuk mendukung proses kreatif dengan menyediakan bingkai atau lembar kerja yang sederhana dan intuitif untuk bereksperimen [8]. openFramework dapat berjalan di lima sistem operasi yaitu Windows, OSX, Linux, iOS, dan Android. Selain itu openFramework dapat

Gambar 2.10 Logo openFrameworks

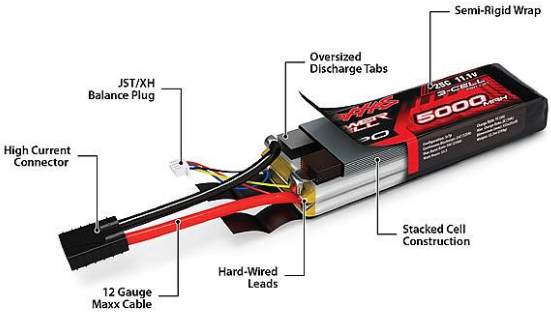
dijalankan di empat IDEs yaitu Xcode, Codeblock, Visual Studio dan Eclipse.

OpenFramework didistribusikan dibawah lisensi oleh MIT. Hal ini membebaskan setiap orang untuk menggunakan openFramework baik untuk tujuan komersial ataupun non-komersial, Umum ataupun privat, *open source* atupun *closed source.*

2.8 Baterai Lipo

Komponen sumber energi utama dari sebuah *quadcopter* adalah baterai LiPo.Seperti gambar 2.9, baterai lipo tidak menggunakan cairan sebagai elektrolit melainkan menggunakan elektrolit polimer kering yang berbentuk seperti lapisan plastik film tipis. Lapisan film ini disusun berlapis-lapis diantara anoda dan katoda yang mengakibatkan pertukaran ion. Dengan metode ini baterai LiPo dapat dibuat dalam berbagai bentuk dan ukuran. Diluar dari kelebihan arsitektur baterai LiPo, terdapat juga kekurangan yaitu lemahnya aliran pertukaran ion yang terjadi melalui elektrolit polimer kering. Hal ini menyebabkan penurunan pada *charging* dan *discharging rate.*

Cara pemilihan baterai adalah berdasarkan jumlah sel dan kapasitasnya. Jumlah sel dalam baterai LiPo adalah sebesar 3,7 Volt. Sehingga bila menggunakan LiPo 3 sel berarti memiliki ukuran 11,1 Volt. Kapasitas baterai LiPo dinyatakan dalam mAh, dimana semakin besar nilai mAh semakin mampu menyimpan tenaga. Semakin besar kapasitas baterai LiPo maka semakin berat ukuran baterai tersebut sehingga akan menurnkan kemampuan terbang di udara. Umumnya untuk *multirotor*



Gambar 2.11 Konfigurasi Baterai Lipo **[9]**

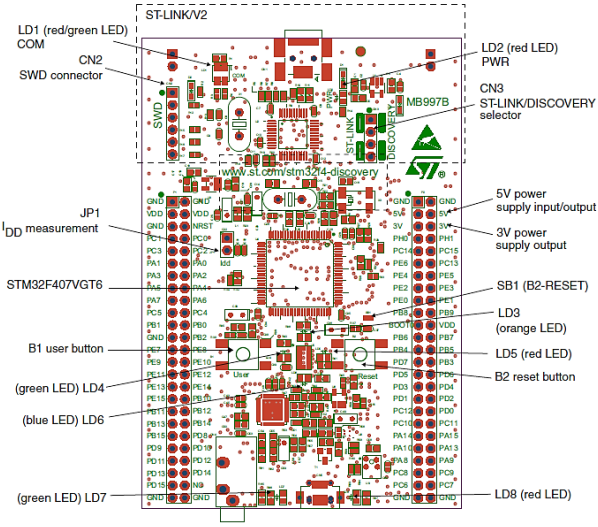
ukuran kecil, menggunakan kapasitas 1500mAh-2200mAh, sedangkan untuk *multirotor* ukuran besar menggunakan kapasitas 4000mAh ke atas [9].

2.9 STM32f4 Discovery

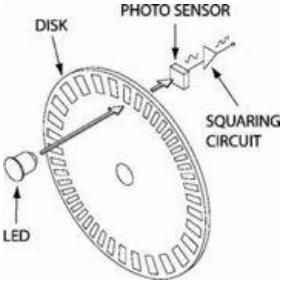
STM32F4 merupakan mikrokontroler berbasis ARM 32 bit. Mikrokontroler jenis STM32F4 menggunakan ARM core-M4F yang dapat melakukan pengolahan sinyal digital. STM32F4 memiliki fitur penambahan kecepatan clock yang lebih tinggi dari STM32F2, 64K CCM static RAM, full duplex I^2S, dan memiliki kecepatan konversi ADC. Gambar 2.12 merupakan gambar bagian dari STM32F4 Discovery. Board developer ini sudah menyediakan downloader ST-LINK sehingga cukup diperlukan koneksi USB untuk melakukan proses download program.

2.10 Rotary Encoder

Rotary encoder adalah salah satu sensor untuk mendeteksi posisi atau pergerakan. Rotary encoder terdiri dari sensor optik berupa LED sebagai pemancar dan photodioda atau phototransistor sebagai penerima. Diantara diantara pemancar dan penerima tersebut terdapat sebuah



Gambar 2.12 STM32F4 - Discovery **[10]**



Gambar 2.13 Rotary Encoder **[11]**

piringan yang memiliki lubang-lubang dengan jarak yang teratur pada daerah sekelilingnya. Saat rotary encoder berputar penerima akan menerima dan tidak menerima cahaya dari LED secara bergantian seiring berputarnya lubang pada piringan. Oleh karenanya output dari photo transistor saat rotary encoder berputar adalah gelombang kotak yang dapat merepresentasi kecepatan putar dari rotary encoder berdasarkan frekuensi yang dihasilkan.

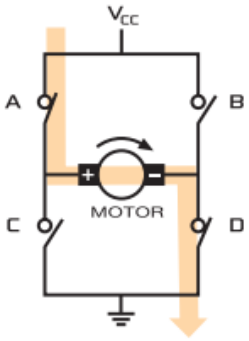
2.11 Mini PC

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. Salah satu bentuk fisik dari computer mini dapat dilihat pada Gambar 2.14.

Gambar 2.14 Mini PC **[12]**

2.12 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* dengan skema ditunjukkan pada Gambar 2.16.



Gambar 2.15 Rangkaian H-Bridge

Seperti pada gambar 2.15, rangkaian dengan konfigurasi *H-Bridge* berasal dari rangkaian *full-bridge* [13]. 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.

2.13 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 2.16.

Gambar 2.16 Motor DC

Halaman ini sengaja dikosongkan

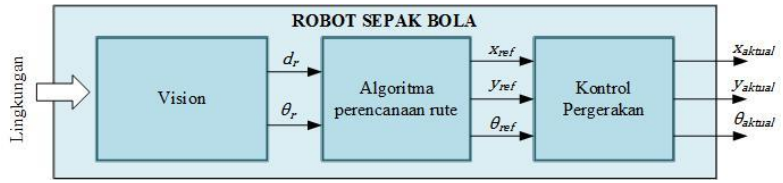
BAB 3
PERANCANGAN SISTEM

Pada bab ini akan dibahas perancangan platfom robot yang akan dibuat meliputi software yang terdiri dari software Mikrokontroller pada robot, pengolahan citra, pengukuran posisi bola, dan learning ANN. Selain itu juga akan dibahas perancangan hardware meliputi desain mekanik robot serta peralatan elektronik yang digunakan pada robot. Pada bab ini akan dijelaskan alur kerja robot melalui blok – blok kerja sistem robot yang meliputi pengolahan citra, pengukuran posisi bola, navigasi, dan sistem roda omni. Lebih jelasnya blok robot di tunjukkan pada Gambar 3.1.

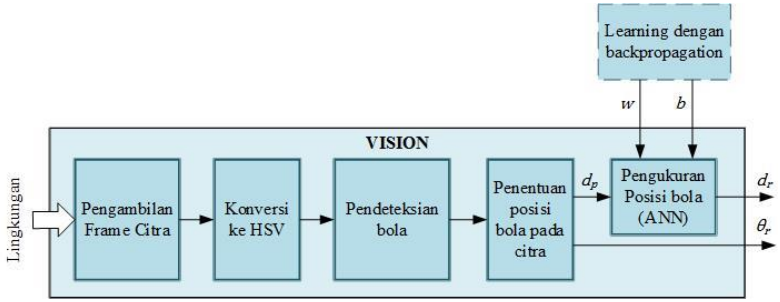
Pada tugas akhir ini akan lebih membahas tentang blok vision yang meliputi pengolahan citra dan pengukuran posisi bola. Teknologi pengolahan citra digunakan untuk mendapatkan informasi posisi bola terhadap robot. Lebih lanjut informasi ini akan digunakan untuk berbagai macam hal seperti menentukan pergerakan untuk mengejar bola, melakukan pemosisian terhadap bola, menentukan prioritas pengejaran bola jika permainan dilakukan dalam 1 tim, pemosisian bola di lapangan untuk lebih lanjut membagikan informasi tersebut kepada robot lain dalam 1 tim dan lain sebagainya.

3.1 Vision

Blok vision pada robot ini merupakan fokus utama pada tugas akhir ini. Blok vision disini mengambil citra dari lapangan sepak bola dan mengolahnya sehingga didapatkan informasi posisi bola terhadap robot. posisi bola terhadap robott dinyatakan dalam koordinat polar yaitu jarak bola terhadap robot yang dinyatakan dalam d_r dan sudut bola terhadap arah depan robot yang dinyatakan dalam θ_r dengan memosikan robot



Gambar 3.1 Diagram Utama Robot

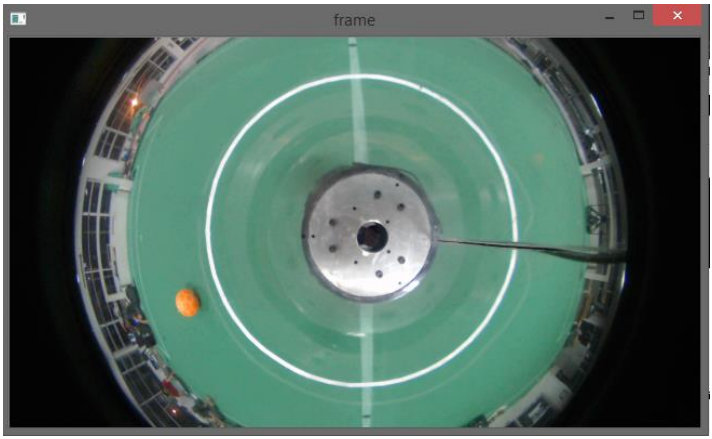


Gambar 3.2 Blok Diagram Vision

sebagai titik origin dari koordinat tersebut. Blok diagram vision lebih jelas dapat ditunjukkan pada Gambar 3.2.

Kamera omnidireksional digunakan agar robot dapat melihat lingkungan sekitar dengan sudut pandang 360 derajat. Pada tugas akhir ini citra kamera omni direksional diciptakan dengan menambahkan lensa fish eye didepan webcam yang digunakan. Sistem ini dapat dilihat pada Gambar 2.9. Tabung akrilik digunakan untuk melindungi kamera tersebut terhadap objek dari luar seperti contohnya tendangan bola dari lawan. Kamera omni direksional diletakkan pada robot bagian paling atas. Peletakan ini dapat dilihat pada Gambar 3.3.Citra yang dihasilkan oleh sistem ini ditunjukkan pada Gambar 3.4. Dapat dilihat bahwa citra yang dihasilkan oleh sistem ini cukup baik. Seluruh area lapangan pada arah horizontal dapat terihat dengan jelas. Namun pada arah vertikal sebagian lapangan tidak terlihat. Hal ini dikarenakan karakteristik dari kamera web cam yang digunakan yang memiliki perbandingan resolusi panjang dan lebar 16:9. Dengan perbandingan resolusi ini menyebabkan citra yang dihasilkan cenderung *wide* dan menyebabkan sudut pandang arah vertikal cenderung sempit dan tak seluas arah horizontal.

Gambar 3.3 Peletakan kamera Omnidireksional



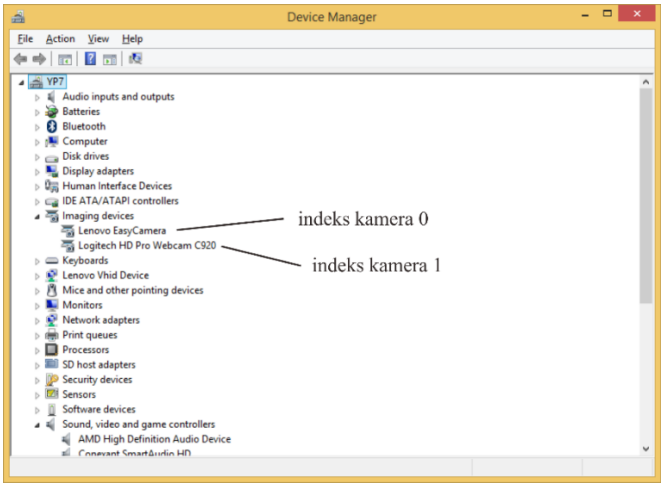
Gambar 3.4 Citra yang dihasilkan oleh kamera omnidireksional dengan menggunakna webcam yang dipasang fish eye

3.1.1 Pengambilan frame Citra (Image Acquisition)

Proses pengambilan *frame* gambar adalah proses pertama yang harus dilakukan pada bagian *in proccess*. Citra gambar di tetapkan dengan resolusi 360 x 640. Semakin tinggi resolusi akan semakin jelas informasi dari citra yang nampak, namun memiliki kelemahan yaitu waktu

pemrosesan yang relatif lebih lama. Sedangkan jika menggunakan resolusi gambar yang terlalu kecil akan semakin banyak informasi dari citra yang hilang meskipun waktu pemrosesan relatif lebih cepat.

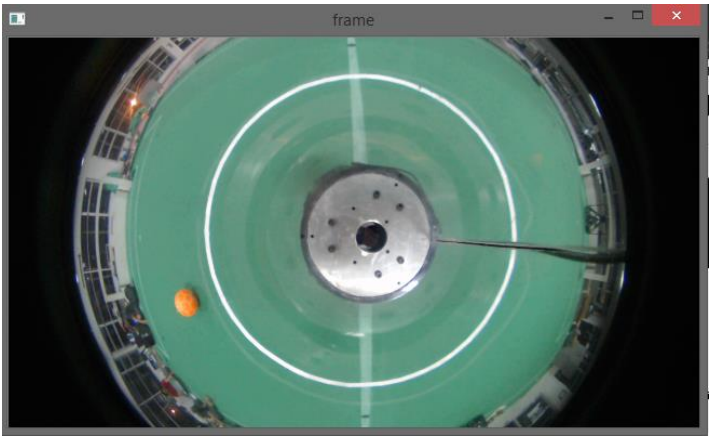
Untuk pengambilan gambar pertama harus di deklarasikan sebuah variabel yang mampu menampung data dari kamera. Open CV telah menyediakan sebuah struktur data untuk menampung gambar dari kamera yaitu `VideoCapture`. Struktur data ini dapat memilih kamera mana yang akan diakses dengan memasukkan indeks kamera pada fungsi open(_n) pada struktur data tersebut. Indeks kamera dapat dilihat pada setting device manager jika menggunakan sistem operasi windows. Indeks kamera dimulai dari 0. Jika kamera berada pada urutan pertama berarti kamera tersebut memiliki indeks 0, jika kamera berada pada urutan kedua maka kamera tersebut memiliki indeks 1, begitu seterusnya. Untuk lebih jelasnya ditunjukkan pada Gambar 3.5.



Gambar 3.5 Penentuan indeks kamera pada device manager

Agar dapat diolah lebih lanjut, citra yang ditangkap dan ditampung sebelumnya perlu dipindahkan ke sebuah struktur data `Mat`. Struktur data ini adalah struktur dari *library* openCV yang menyimpan informasi sebuah citra. Setelah sebuah citra disimpan dalam struktur data tersebut, citra dapat diolah sesuai keinginan penggunanya.

Pada proses pengambilan citra, citra diambil dalam bentuk citra RGB atau BGR. Kedalaman warna yang dipilih untuk setiap kanalnya adalah sebesar 8 bit sehingga setiap kanal dapat merepresetasikan 255 tingkat kecerahan yang berbeda. Dengan adanya 3 kanal yaitu *red*, *green*, dan *blue*, kombinasi atau variasi warna yang dapat dihasilkan adalah sebesar 255^3 atau sebesar 16.581.375 warna. Hasil pengambilan citra pada proses ini ditunjukkan pada Gambar 3.6.



Gambar 3.6 Hasil Pengambilan Citra

3.1.2 Konversi citra RGB ke HSV(RGB to HSV)

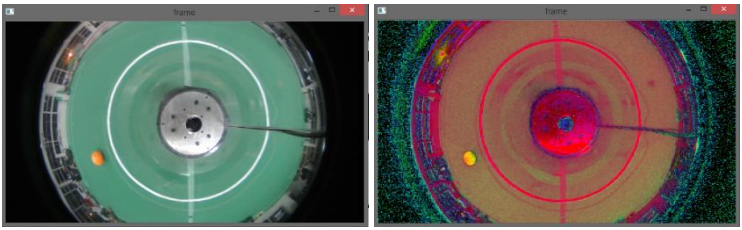
Pada proses ini informasi warna dalam bidang RGB(*red, green, blue*) disusun ulang pada bidang HSV(*hue, saturation, value*). Hal ini perlu dilakukan mengingat informasi yang disajikan oleh format RGB tidak memisahkan antara informasi warna dan informasi kecerahan. Sedangkan pada format HSV informasi tersebut dipisahkan oleh sebuah kanal *value.* Dengan demikian diharapkan informasi warna yang diinginkan tidak terpengaruh oleh intensitas cahaya mengingat kondisi pencahayaan saat kopetisi di lapangan tidak menentu tergantung penyediaan sarana oleh panitia yang bersangkutan.

Proses konversi format RGB ke HSV dibantu oleh sebuah fungsi dari *library* open CV yaitu void `cvtColor(InputArray` **`src,`** `OutputArray` **`dst,`** `int` **`code,`** `int` **`dstCn`**`=0 )`. Fungsi ini memiliki 4 parameter. `InputArray` **`src`** yaitu array input yang dapat diisi oleh variabel citra yang ingin dikonversi formatnya.

`OutputArray` **`dst`** yaitu array output yang dapat diisi oleh variabel dimana hasil konversi ingin disimpan. `int` **`code`** yaitu kode yang menunjukkan proses konversi apa yang akan dilakukan. Kode kode yang dapat dimasukkan dalam parameter ini beserta artinya dijelaskan pada lampiran [2]. `int` **`dstCn`**`=0` merepresentasikan jumlah kanal pada variabel output. Jika parameter tersebut bernilai 0 jumlah kanal ddidapat secara otomatis dari `src` dan `code` [14]. Pada proses ini kode yang digunakan adalah `CV BGR2HSV`. Hasil konversi dari citra dalam format HSV ditunjukkan pada Gambar 3.7 berikut.

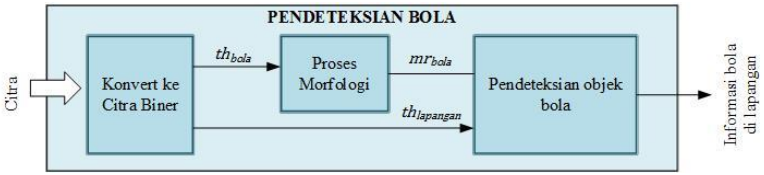
3.1.3 Pendeteksian Bola(Ball detection)

Pada tahap ini dilakukan proses pencarian posisi bola dari citra yang telah di proses sebelumnya. Pendeteksian bola dilakukan dengan memanfaatkan karakteristik warna bola yaitu warna oranye gelap. Untuk mengurangi noise akibat salah deteksi warna oranye lain, pencarian warna oranye bol hanya dilakukan di atas lapangan berwarna hijau. Sehingga jika terdapat warna oranye diatas warna bukan warna hijau lapangan maka bisa dipastikan bahwa objek tersebut bukanlah bola. Oleh karena itu diperlukan juga proses pendeteksian lapangan. Proses pendeteksian objek dapat dijabarkan dalam beberapa poin yaitu konversi ke citra biner, proses morfologi, dan pendeteksian objek. Blok dari sistem ini di tunjukkan pada



(a) (b)

Gambar 3.7 Hasil konversi (a) RGB ke (b) HSV



Gambar 3.8 Blok Diagram Pendeteksian Bola

3.1.3.1 Konvert ke citra biner

Sebelum melakukan proses pendeteksian objek, terlebih dahulu harus dipisahkan antara *background* (bukan objek yang diinginkan) dan objek. *Background* dan objek dapat direpresentasikan oleh citra biner. Putih menunjukkan objek sedangkan hitam menunjukkan *background* atau sebaliknya sesuai dengan pengguna masing-masing. Pada tugas akhir kali ini objek dinyatakan dalam warna putih sedangkan background dinyatakan dalam warna hitam.

Proses pembuatan citra biner dilakukan dengan memisahkan nilai pixel yang berada di antara batasan tertentu dengan yang berada diluar batasan tersebut. Semisal diberikan sebuah batasan untuk kanal HSV dengan nilai *Hue* antara 10 sampai 20, Saturation antara 20 sampai 30, dan value antara 0 samapai 10. Jika terdapat sebuah pixel memiliki nilai HSV masing-masing 15, 25, dan 5, maka pixel tersebut diberikan warna putih atau ditandai sebagai objek karena semua nilai dari pixel tersebut berada diantara batas-batas yang telah ditentukan sebelumnya. Sedangkan jika ada sebuah pixel bernilai HSV masing-masing 15, 25, dan 15, maka pixel tersebut ditandai sebagai background dengan memberi warna hitam. Hal ini dikarenakan salah satu kanal dari pixel tersebut memiliki nilai diluar batas yang telah ditentukana sebelumnya.

Proses pembuatan citra seperti yang dijelaskan di atas dapat diterapkan dengan menggunakan salah satu fungsi *thresholding* yang telah disediakan oleh openCV yaitu **`inRange`**`(InputArray` **`src`**`, InputArray` **`lowerb`**`, InputArray` **`upperb`**`, OutputArray` **`dst`**`)`. Fungsi tersebut terdiri dari 4 parameter. Parameter pertama yaitu `InputArray` **`src`** adalah array input yang ingin dilakukan proses *thresholding.* Parameter kedua dan ketiga adalah `InputArray` **`lowerb`** dan `InputArray` **`upperb`** yang merupakan batas threshold bawah dan atas dari masing-masing kanal.

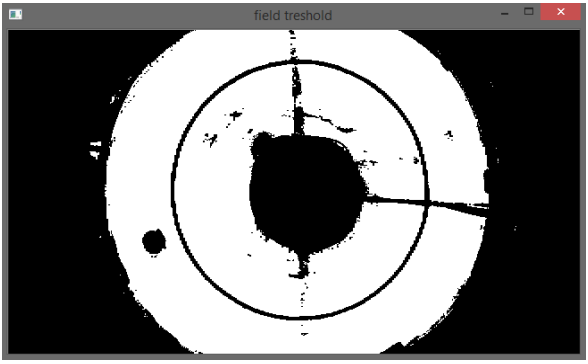
parameter terakhir yaitu `OutputArray` **`dst`** yang merupakan array output yang akan berisi citra biner dari proses thresholding.

Proses thresholding dilakukan dua kali yaitu untuk warna bola dan warna lapangan. Hasil thresholding bola dan lapangan ditunjukkan pada Gambar 3.9 dan Gambar 3.10.

3.1.3.2 Proses morfologi

Proses morfologi yang diberikan pada citra berupa proses morfologi opening. Proses morfologi opening terdiri dari dua proses morfologi dasar yaitu *erotion* dan dilanjutkan dengan *dilation.* Sesuai dengan namnya proses *erosion* adalah proses mengikis atau mengerosi objek pada bagian

Gambar 3.9 Hasil *thresholding* bola



Gambar 3.10 Hasil *thresholding* lapangan

terluarnya. Dalam hal ini daerah berwarna putih berkurang luasannya bahkan hilang untuk objek-objek berukuran kecil. Proses selanjutnya adalah *dilation* yaitu proses pelapisan objek pada bagian terluar. Proses ini menyebabkan luasan objek bertambah. Kedua proses tersebut menyebabkan objek-objek kecil yang biasanya berupa noise menghilang. Dengan proses ini diharapkan pengukuran yang diperoleh lebih akurat.

Proses morfologi hanya dilakukan pada hasil threshol dari bola dan tidak pada threshold lapangan. Hal ini dikarenakan lapangan memiliki ukuran yang relatif besar dan sedikit kemungkinan terganggu oleh *noise-noise* kecil. Selain itu proses morfologi memakan waktu proses yang cukup lama, sehigga jika proses morfologi dilakukan pada kedua citra biner hal tersebut menyebabkan waktu proses yang relatif lebih lama jika dibandingkan proses morfologi yang hanya dilakukan pada satu citra biner.

Pertama sebelum melakukan proses morfologi pada open CV, terlebih dahulu perlu dibentuk sebuah kernel dengan ukuran tertentu untuk melakukan pemindaian dan konvolusi pada citra dengan transformasi morfologi. Ukuran dari kernel mempengaruhi seberapa tebal objek akan terkikis atau terlapisi oleh proses *erotion* dan *dilation*. Pembuatan kernel dapat dilakukan dengan menggunakan fungsi **getStructuringElement**(int **shape**, Size **ksize**, Point **anchor**=Point(-1,-1)). parameter int **shape** mengatur benttuk dari kernel. Parameter ini dapat diisi dengan **MORPH_RECT** untuk kernel berbentuk kotak atau **MORPH_ELLIPSE** untuk kernel berbentuk elips. Kernel berbentuk elips digunakan pada tugas akhir kali ini mengingat bentuk bola adalah lingkaran.

Di openCV telah disediakan fungsi untuk melakukan proses morfologi *erotion* dan *dilation* yaitu **erode**(InputArray **src**, OutputArray **dst**, InputArray **kernel**, Point **anchor** = Point(-1,-1), int **iterations** = 1, int **borderType** = BORDER_CONSTANT, const Scalar& **borderValue** = morphologyDefaultBorderValue()) dan **dilate**(InputArray **src**, OutputArray **dst**, InputArray **kernel**, Point **anchor** = Point(-1,-1), int **iterations** = 1, int **borderType** = BORDER_CONSTANT, const Scalar& **borderValue** = morphologyDefaultBorderValue()). Kedua fungsi tersebut memiliki parameter yang sama. Dari beberapa parameter yang tersedia, di tugas akhir kali ini hanya memanfaatkan 3 parameter pertama dan

membiarkan parameter yang lain pada nilai defaultnya. Parameter pertama adalah `InputArray` **`src`** yaitu array input berupa citra yang ingin diberikan proses morfologi. Selanjutnya adalah `OutputArray` **`dst`** yaitu array output dari proses morfologi. Parameter selanjutnya adalah `InputArray` **`kernel`** yaitu kerne yang digunakan pada proses morfologi. Penulisan kode pada visual studio adalah sebagai berikut:

```
if (opening_ball > 0)
{
erode(ball, ball, getStructuringElement(MORPH_ELLIPSE,
Size(opening_ball, opening_ball)));

dilate(ball, ball, getStructuringElement(MORPH_ELLIPSE,
Size(opening_ball, opening_ball)));
}
```

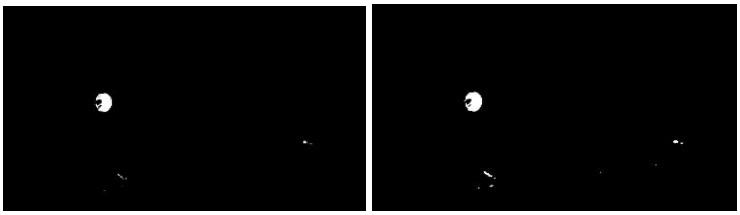
Proses opening dilakukan jika variabel `opening_ball` bernilai lebih dari 0. Jika proses tersebut dilakukan saat variabel `opening_ball` bernilai 0 maka akan terjadi error saat program di run dikarenakan tidak dapat dihasilkannya kernel dengan ukuran 0. Hasil dari proses morfologi ditunjukkan pada gambar dibawah ini

(a) **(b)**

Gambar 3.11 Perbandingan (a) citra biner asli dan (b) hasil erotion dengan kernel elips 2 px

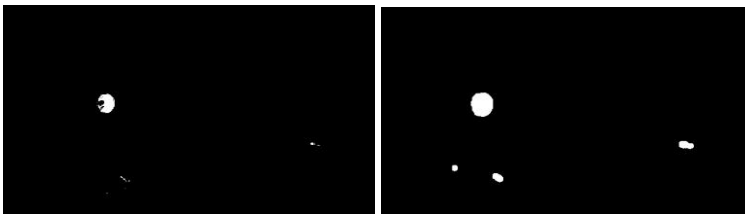
(a) **(b)**

Gambar 3.12 Perbandingan (a) citra biner asli dan (b) hasil erotion dengan kernel elips 10 px



(a) **(b)**

Gambar 3.13 Perbandingan (a) citra biner asli dan (b) hasil dilation dengan kernel elips 2 px



(a) **(b)**

Gambar 3.14 Perbandingan (a) citra biner asli dan (b) hasil dilation dengan kernel elips 10 px

3.1.3.3 Pendeteksian objek bola

Pendeteksian objek dilakukan dengan mencari objek bola terbesar dan melihat apakah disekitar objek tersebut terdapat warna hijau atau tidak. Jika disekirar objek tersebut terdapat warna hijau maka objek

tersebut adalah bola dan ditandai sebagai bola sedangkan jika disekitar objek bola tersebut tidak terdapat warna hijau objek tersebut diabaikan dan tidak ada bola terdeteksi.

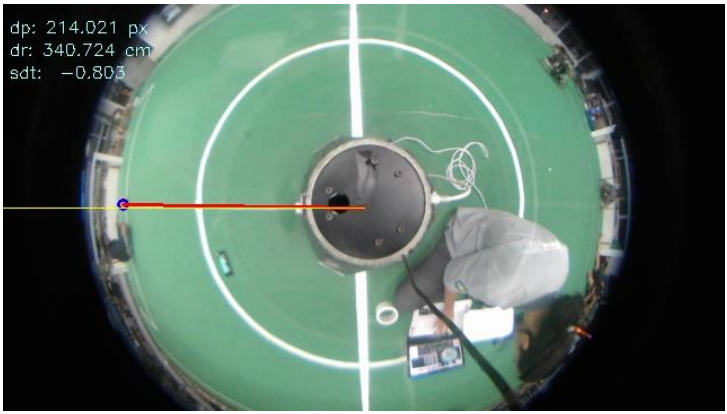
Sebelum melakukan seleksi terlebih dahulu dilakukan proses menandai setiap objekberdasarkan letaknya pada citra dan radiusnya. Untuk itu digunakan fungsi dari openCV yaitu `minEnclosingCircle ((Mat)contours[i], center[i], radius[i])`. Fungsi tersebut bekerja untuk membuat lingkaran terkecil yang mampu melingkupi seluruh bagian objek (untuk satu objek). Data yang dikeluarkan daari fungsi tersebut berupa radius lingkaran yang melingkari objek yang akan disimpan pada variabel `radius` dan titik tengah dari lingkaran tersebut pada citra yang akan disimpan pada variabel `center`.

Setelah didapatkan beberapa lingkaran yang melingkari objek beserta informasi posisi dan radiusnya, proses selanjutnya adalah melakukan proses seleksi untuk menentukan objek bola. Objek bola adalah objek terbesar yang berada diatas lapangan berwarna hijau. Jika objek tersebut tidak berada diatas lapangan hijau maka objek tersebut bukan bola. Berikut adalah potongan kode C++ untuk seleksi objek bola diatas lapangan:

```
check_x_positive = center[i].x + (radius[i] + 5);
check_x_negative = center[i].x - (radius[i] + 5);
check_y_positive = center[i].y + (radius[i] + 5);
check_y_negative = center[i].y - (radius[i] + 5);

if (field.at<uchar>(check_y_positive, center[i].x) == 0 &&
field.at<uchar>(check_y_negative, center[i].x) == 0 &&
field.at<uchar>(center[i].y, check_x_positive) == 0 &&
field.at<uchar>(center[i].y, check_x_negative) == 0) radius[i] =
0;
```

`check_x_positive`, `check_x_negative`, `check_y_positive`, `check_y_negative` adalah variabel yang berisi titik pixel mana yang akan diperiksa apakah terdapat warna lapangan atau tidak. Dari proses tersebut jika baik di kanan, kiri, atas, dan bawah objek bola tidak terdapat warna lapangan maka nilai radius dari objek tersebut dibuat 0. Hal tersebut menandai bahwa objek tersebut bukanlah bola. Proses tersebut di ulang untuk semua objek yang terdeteksi. Jika setelah semua proses tersebut tidak ada objek yang memiliki radius lebih dari 0, program memastikan bahwa tidak ada objek bola pada citra. Sedangkan jika setelah proses tersebut terdapat beberapa objek bola, program menentukan objek bola



Gambar 3.15 Pendeteksian Bola

adalah objek dengan ukuran radius paling besar. Hasil pendeteksian bola ditunjukkan pada Gambar 3.15.

3.1.4 Penentuan Posisi bola pada citra

Setelah bola dapat diteksi, langkah selanjutnya adalah menentukan dimana posisi bola tersebut pada citra. Pada proses pendeteksian bola bisa didapatkan informasi ada atau tidaknya bola. Proses penentuan posisi bola ini dilakukan jika terdapat bola terdeteksi dan di abaikan jika tidak ada bola terdeteksi.

Pada tahap ini proses penentuan posisi bola adalah dengan merubah koordinat bola pada citra ke koordinat sudut dan jari-jari terhadap titik tengah citra. Proses ini dilakukan mengingat kamera yang digunakan adalah kamera omni direksionnal. Pada kondisi ideal mengabaikan segala kekeliruan mekanik, sudut posisi bola terhadap depan bagian depan kamera (dalam hal ini ditandai dengan garis kuning seperti pada Gambar 3.16) akan sama dengan sudut posisi bola terhadap bagian depan robot. Oleh karenanya proses ini diperlukan untuk mempermudah proses selanjutnya.

Saat terdapat bola terdeteksi, posisi bola pada citra akan diketahui juga dengan melihat variabel `center`. Diasusikan posisi bola dari proses pendeteksian bola dinyatakan dengan x_o dan y_o. Dan posisi tengah citra diasumsikan dengan x_c dan y_c. Untuk menkonversi posisi tersebut menjadi jari-jari dan jarak terhadap titik origin yang merupakan titik tengah citra digunakan persamaan (3.1) dan (3.2) berikut:

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

$$\theta = \tan^{-1}\frac{(y_o - y_c)}{-(x_o - x_c)} \quad (3.2)$$

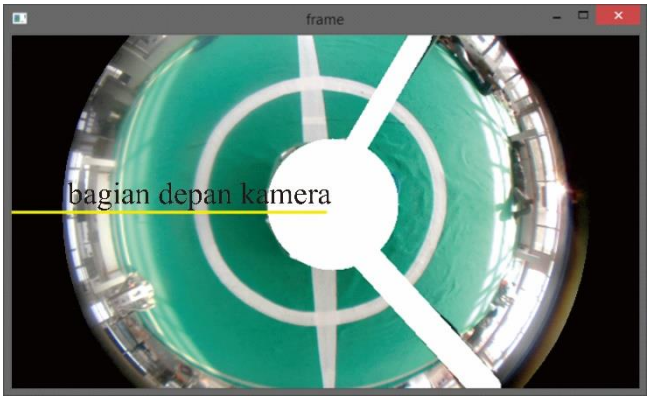
3.1.5 Pengukuran Posisi bola terhadap robot

Pada tahap sebelumnya didapatkan informasi tentang posisi bola terhadap titik tengah pada citra. Namun jari-jari yang diperoleh pada informasi ini tidak merepresentasikan nilai jarak sesungguhnya. Hubungan antara jari-jari bola terhadap titik tengah pada citra dan jarak bola sesungguhnya terhadap robot juga tidak linear. Oleh karenanya diperlukan sebuah proses untuk menentukan posisi bola terhadap robot dari informasi yang ada.

Salah satu metode yang dapat dilakukan adalah dengan menggunakan sistem jaringan saraf buatan. Hasil w dan b pada *preproccess* digunakan pada bagian ini.

Berikut adalah desain ANN yang akan diterapkan pada tahap berikut digambarkan pada Gambar 3.22.

3.1.6 Learning dengan menggunakan backpropagation



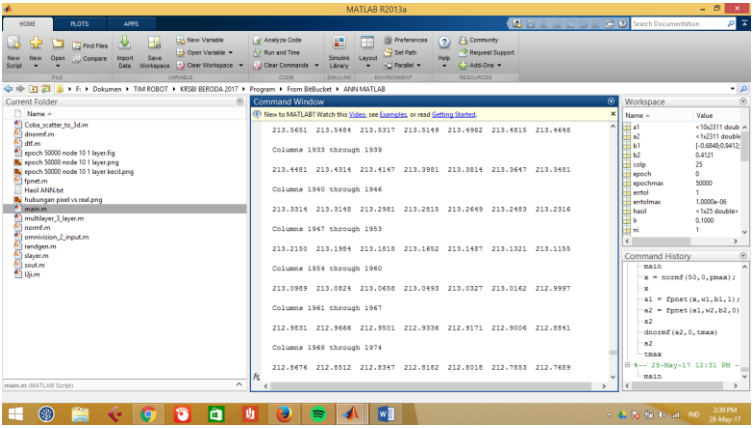
Gambar 3.16 Penentuan bagian depan kamera pada kamera omnidireksional

Back propagation adalah metode untuk mendapatkan nilai – nilai dari w dan b di masing-masing layer pada jaringan saraf tiruan. Nilai w dan b pada masing-masing layer diperoleh memalui proses pembelajaran.

Proses pembelajaran dilakukan dengan memberikan sebuah training set yang terdiri dari input dan output yang seharusnya diberikan oleh sistem. Proses learning menggunakan metode back propagation dibantu oleh software matlab 2013.

3.1.6.1 Inisialisasi nilai w dan b

Sebelum semua proses dilakukan pertama perlu dilakukan proses inisialisasi nilai w dan b pada setiap layer jaringan saraf tiruan. Proses inisialisasi dilakukan dengan cara memberikan nilai acak antara -1 sampai 1 untuk setiap w dan b di setiap layer. Pengacakan ini dilakukan dengan memanfaatkan fungsi `rand()` pada matlab. Fungsi rand mengeluarkan nilai acak antara 0 sampai 1 yang terdistribusi merata [15]. Karena nilai random yang dikeluarkan oleh fungsi tersebut tidak sesuai dengan kebutuhan maka diperlukan pembuatan sebuah fungsi baru yang memanfaatkan fungsi `rand()` tersebut dan dapat menghasilkan nilai keluaran antara -1 sampai 1. Pada tugas akhir kali ini dibuat sebuah fungsi bernama `randgen(row,col)`. Parameter "row" dan "col" menunjukan ukuran kolom dan baris matriks yang ingin diacak nilainya. Isi dari fungsi tersebut adalah sebagai berikut:



Gambar 3.17 Tampilan Software Matlab 2013

```
function r = randgen(row,col)
r = 2*rand(row,col)-1;
```

Fungsi tersebut akan mengisi matriks r dengan nilai acak sejumlah "row" baris dan "col" kolom.

3.1.6.2 Normalisasi nilai input dan output

Normalisasi nilai input dan output diperlukan pada proses forward propagation agar nilai input dan output tersebut berada pada daerah kerja dari *transfer function* baik log sigmoid, tangen sigmoid. Grafik transfer function log sigmoid dan tangen sigmoid ditunjukkan pada Gambar 3.18 dan Gambar 3.19. Proses normalisasi perlu dilakukan kareana jika tidak dilakukan *transfer fungtion* sigmoid akan nampak seperti *transfer function* hard limiter jika nilai rentang input terlalu besar dan tidak akan pernah tercapai jika nilai output dari lebih besar dari 1 atau kurang dari -1. Oleh karenanya normalisasi membawa nilai input dan output berada pada daerah kerja dari *transfer function* tersebut.

Pada tugas akhir kali ini dibuat sebuah fungsi khusu untuk melakukan proses normalisasi yaitu `normf(x,xmin,xmax)`. Fungsi ini membawa nilai input x yang berada di antara rentang xmin dan xmax pada rentang diantara -1 sampai 1.

Isi dari fungsi tersebut ditunjukkan dibawah ini:

```
function [nx] = normf(x,xmin,xmax)
% NORMF - normalization function
nx = 2*(x-xmin)./(xmax-xmin) - 1;
```

3.1.6.3 Foward propagation

Forward propagation adalah proses memasukkan data input pada layer input atau layer pertama dan menyebarkannya ke layer selanjutnya yang berada di deapannya. Di sebut sebagai *forward propagation* karena arah propagasi inputnya adalah maju mulai dari layer pertama menuju layer output. Di layer output akan didapatkan nilai keluaran dari keseluruhan sistem jaringan saraf tiruan.

Setiap layer dapat berbeda-beda baik dari segi jumlah neuronnya ataupun transfer functionnya. Terdapat beberapa jenis transfer function yang biasanya digunakan pada layer sistem jaringan saraf tiruan yaitu: log sigmoid, tangen log sigmoid, linear, dan hard limiter. Transfer function tersebut masing-masing dapat dinyatakan dalam fungsi berikut ini:

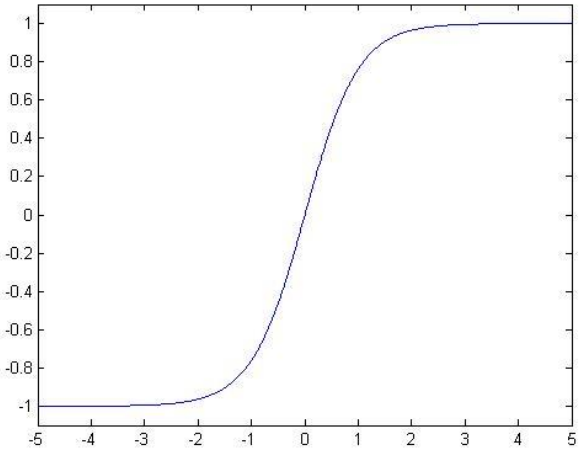
$a = f(n) = \begin{cases} 1, n \geq 0 \\ 0, n < 0 \end{cases}$ hard limiter

$a = f(n) = \frac{1}{1+e^{-n}}$ log sigmoid

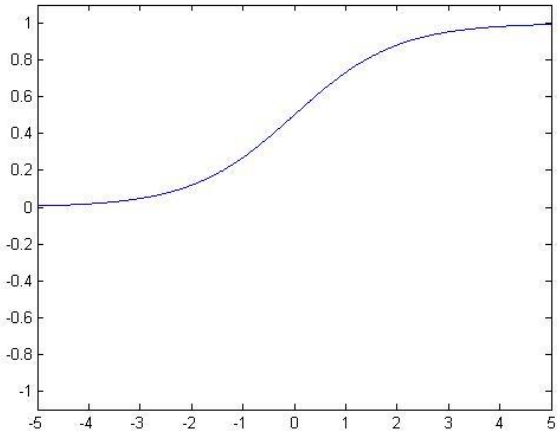
$a = f(n) = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$ tangen log sigmoid

$a = f(n) = n$ linear

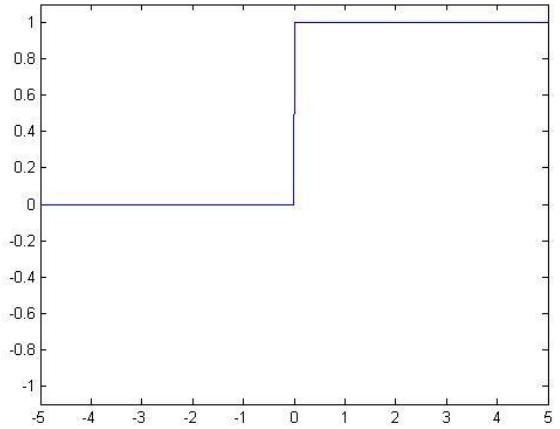
Grafik dari masing masing transfer function ditunjukkan pada Gambar 3.18, Gambar 3.19, Gambar 3.20, dan Gambar 3.21.



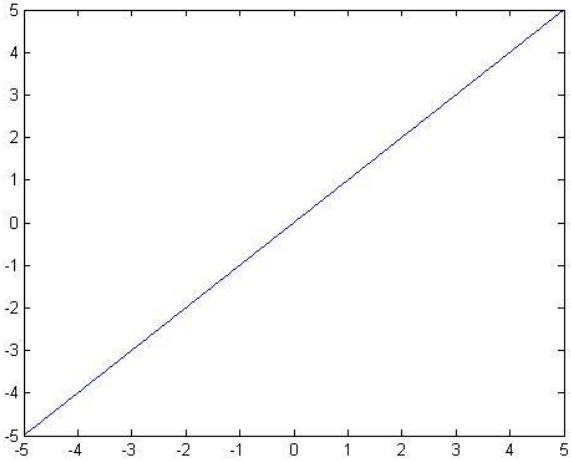
Gambar 3.18 Grafik *Transfer Function* Log Sigmoid



Gambar 3.19 Grafik *Transfer Function* Log Tangean Sigmoid



Gambar 3.20 Grafik *Transfer Function* Hard Limiter



Gambar 3.21 Grafik *Transfer Function* Linear

Untuk untuk merealisasikan proses ini dalam Matlab maka dibuat sebuah fungsi yaitu `fpnet (p,w,b,tftype)`. Fungsi ini memiliki beberapa parameter yaitu `p` yaitu vektor input, `w` yaitu matriks w yang akan digunakan pada layer tersebut, `b` yaitu matriks b yang akan digunakan pada layer tersebut, dan `tftype` yaitu jenis *transfer function*. Jika `tftype` bernilai 0 maka *transfer function* yang digunakan adalah linear, jika `tftype` bernilai 1 maka jenis *transfer function* yang digunakan adalah log sigmoid, jika `tftype` bernilai 2 maka jenis *transfer function* yang digunakan adalah tangen sigmoid dan jika `tftype` bernilai 3 maka jenis *transfer function* yang digunakan adalah *hardlimiter*. Kode matlab untuk fungsi tersebut dapat dilihat dibawah ini:

```
function [a] = fpnet (p,w,b,tftype)
% FPNET - forward propagation of neural nerwork
[~,col] = size(p);
o = ones(1,col);
a = [];
n = w*p + b*o;
if tftype == 0,     % pure linear
```

```
    a = n;
end
if tftype == 1,     % log sigmoid
    a = 1 ./ (1+exp (-n));
end
if tftype == 2,     % tangent sigmoid
    a = (exp(n)-exp(-n)) ./ (exp(n)+exp(-n));
end
if tftype == 3,     % hard limiter
    a = hardlim(n);
end
```

3.1.6.4 Backward propagation

Pada proses selanjutnya yang dilakukan setelah *foward propagation* adalah *backward propagation* atau propagasi balik, diaktakan demikian karena propagasi proses dilakukan dimulai dari layer output menuju layer input. Proses ini dilakukan untuk mendapatkan nilai dari sensitifitas dari setiap layer berdasarkan error yang diperoleh dari target dan output sistem jaringan saraf tiruan.

Sensitifitas dari layer output dirumuskan dengan persamaan berikut:

$$s^M = -2\dot{F}^M(n^M)(t-a) \tag{3.3}$$

Keterangan:

M : layer ke M

$\dot{F}^M$: Matriks diagonal yang merupakan turunan dari transfer function layer *M*

n^M : input *transfer function* pada layer ke *M*

t : target

a : output dari proses *forward propagation*

Berbeda dengan layer output, sensitifitas dari layer lain dapat dinyatakan dengan persamaan berikut:

$$s^m = \dot{F}^m(n^m)(w^{m+1})s^{m+1} \tag{3.4}$$

Keterangan:

m : layer ke m (m = M − 1, … , 2,1)

$\dot{F}^m$: Matriks diagonal yang merupakan turunan dari transfer function layer *m*

n^m : input *transfer function* pada layer ke *m*

w^{m+1} : matriks w untuk layer m + 1
s^{m+1} : sensitivitas untuk layer ke m + 1

Untuk lebih jelasnya nilai dari $\dot{F}^m$dapat dijabarkan sebagai berikut:

$$\dot{\mathbf{F}}^m(\mathbf{n}^m)=\begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \dots & 0 \\ 0 & \dot{f}^m(n_2^m) & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \dot{f}^m(n_{s^m}^m) \end{bmatrix} \tag{3.5}$$

Dengan:

$$\dot{f}^m(n_j^m)=\frac{\partial f^m(n_j^m)}{\partial n_j^m} \tag{3.6}$$

Dari persamaan tersebut maka nilai parsial derivatif untuk setiap transfer function dapat dijabarkan sebagai berikut:

log *sigmoid*	$a = f(n) = \frac{1}{1+e^{-n}}$	$\dot{f}(n) = a(1-a)$
tangen sigmoid	$a = f(n) = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$	$\dot{f}(n) = 1-a^2$
linear	$a = f(n) = n$	$\dot{f}(n) = 1$
hard limiter	$a = f(n) = \begin{cases}1, n \geq 0\\0, n < 0\end{cases}$	$\dot{f}(n) = 0$

Untuk mendapatkan nilai $\dot{F}^m$ pada matlab dibuat sebuah fungsi dengan nama `dtf(a,tftype)`. Fungsi tersebut memiliki 2 parameter yang masing –masing adalah `a` adalah input dari layer yang bersangkutan dan `tftype` adalah tipe dari transer function yang digunakan pada layer yang bersangkutan. terdapat nilai yang dapat dimasukkan pada `tftype` yaitu 0 untuk linear, 1 untuk log sigmoid dan 2 untuk tangen sigmoid. Kode untuk fungsi ini ditunjukkan dibawah ini:

```
function [dv] = dtf(a,tftype)
dv=[];
% DTF - derivative of tansfer function
if tftype == 0,
    dv = diag(ones(size(a)));   % Linear transfer function
end
if tftype == 1,
```

```
    dv = diag(a .* (1-a));      % Logsig transfer function
end
if tftype == 2,
    dv = diag(1 - a .* a);     % tansig transfer function
end
```

Selanjutnya perlu dibuat sebuah fungsi yang dapat menghasilkan nilai sensitivitas dari setiap layer. Untuk mendapatkan nilai sensitivitas pada layer output maka dibuat sebuah fungsi yaitu `sout(a,t,tftype)`. Fungsi ini memiliki 3 parameter masing-masing adalah `a` yaitu output dari layer output, `t` yaitu terget pada training set, dan `tftype` adalah jenis transfer function yang digugnakan pada layer output. Nilai dan jenis transfer function yang dimaksud pada fungsi ini sama dengan nilai dan jenis transfer function pada fungsi `dtf(a,tftype)` yang telah dijelaskan sebelumnya. Kode untuk fungsi ini ditunjukkan dibawah ini:

```
function [so] = sout(a,t,tftype)
% SOUT - Output sensitifity
so = -2*dtf(a,tftype)*(t – a);
```

Selain itu perlu dibuat jugas sebuah fungsi untuk mendapatkan nilai sensitivitas dari layer lain selain layer output yaitu `slayer(a,w,s,tftype)`. Fungsi ini memiliki 4 parameter masing-masing adalah `a` yaitu nilai output dari sebuah layer, `w` yaitu matriks w dari sebuah layer, `s` yaitu nilai sensitivitas dari sebuah layer, dan `tftype` yaitu tipe transfer fungction pada sebuah layer. Nilai dan maksud dari nilai yang dapat dimasukkan ke dalam `tftype` sama dengan nilai dan maksud dari nilai tersebut pada fungsi `dtf(a,tftype)` yang telah di jelaskan sebelumnya. Kode dari fungsi ini dituliskan dibawah ini:

```
function [sl] = slayer(a,w,s,tftype)
% SLAYER – Layer sensitifity
sl = dtf(a,tftype)*(w')*s;
```

Setelah proses backward propagation dilakukan, setiap layer akna memiliki nilai sensitivitas masing-masing. Nilai tersebut akan digunakan untuk bahan pertimbangan memperbarui nilai w dan b pada setiap layer pada tahap selanjutnya.

3.1.6.5 Update nilai w dan b

Pada proses ini dilakukan proses pembaruan nilai w dan b pada setiap layer dengan mempertimbangkan sensitivitas dari setiap layer. Dengan proses ini diharapkan nilai error yang dihasilkan pada layer output terhadap target yang telah ditentukan pada *trainin set* akan semakin kecil.

Proses pembaruan dari nilai bobot w dan b berdasarkan persamaan berikut:

$$w^m(k+1) = w^m(k) - \alpha s^m (a^{m-1})^T \tag{3.7}$$

$$b^m(k+1) = b^m(k) - \alpha s^m \tag{3.8}$$

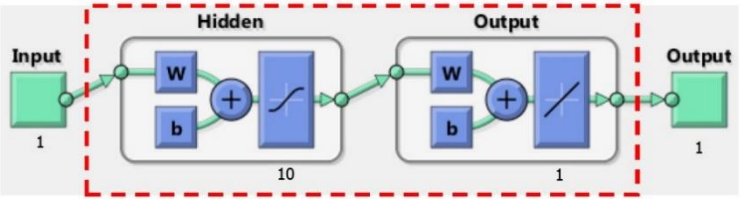
dengan α adalah *learning rate.*

Kode untuk proses update nilai w dan b pada matlab ditunjukkan dibawah ini:

```
w2 = w2 - lr*s2*a1';
b2 = b2 - lr*s2;
w1 = w1 - lr*s1*a0';
b1 = b1 - lr*s1;
```

setelah dilakukan proses pembaruan nilai bobot w dan b langkah selanjutnya adalah melakukan proses pengecekan apakah nilai error yang dihasilkan oleh nilai w dan b yang diterapkan pada sistem kurang dari nilai error yang diinginkan. Jika nilai error kurang dari nilai error yang diinginkan proses pembelajaran selesai dan nilai w dan b dapat digunakan pada sistem jaringan saraf tiruan. Sedangkan jika nilai error masih lebih besar dari nilai error yang diinginkan, maka proses diulang lagi dari poin 3.1.6.3.

Terkadang proses pembelajaran memakan waktu yang sangat lama. Proses pembelajaran tidak menghasilkan nilai error kurang dari nilai error yang diharapkan. Untuk mengatasi hal tersebut kondisi yang mengindikasikan berhentinya proses pembelajaran tidak hanya bergantung pada nilai error yang kurang dari nilai error yang diharapkan namun juga bergantung pada jumlah perulangan yang telah dilakukan apakah telah melebihi batas tertentu atau tidak. Jika salah satu dari kedua hal tersebut terpenuhi, hal itu mengindikasikan harus dihentikannya proses pembelajaran.



Gambar 3.22 Desain ANN yang akan diterapkan pada robot

Sistem ANN secara keseluruhan terdiri atas 1 input dan 1 output. ANN terdiri dari 1 hidden layer dan 1 output layer. Hidden layer memiliki 1 input dan 10 output. Transfer function yang digunakan pada layer ini adalah log sigmoid. Output layer terdiri dari 10 input dan 1 output. Layer ini memiliki transfer function linear.

Untuk menerapkan sistem perhitungan matrix yang lebih mudah, maka digunakan bntuan library yaitu *Amadillo*. Library ini menyediakan operasi perhitungan matriks tanpa harus kita membuat sendiri dengan penggunaan sintaks perulangan yang cukup rumit.

Sebelum data dari proses sebelumnya masuk ke ANN, terlebih dahulu harus dilakukan proses normalisasi agar transfer function dapat bekerja secara optimal. Untuk melakaukan proses tersebut, maka dibuat sebuah fungsi bernama `normalisasi(float in, float inMin, float inMax)`. Berikut adalah kode C++ untuk fungsi tersebut:

```
float omnivision::normalisasi(float in, float inMin, float inMax)
{
  float out = 0;
       out = (2.0 * (in - inMin) / (inMax - inMin)) - 1;
  return out;
}
```

Fungsi tersebut terdiri dari 3 parameter yaitu `float in` adalah data input, `float inMin` adalah input minimal, `float inMax` adalah input maksimal.

Berikutnya adalah proses memasukkan data yang telah di normalisasi melalui kedua layer yang telah tersedia. Berikut adalah kode C++ untuk proses tersebut:

```
layer1 = (w1 * normalizedN) + b1; //pembobotan w dan bias b L1
layer1 = 1 / (1 + exp(-layer1)); //TF log sigmoid
layer2 = (w2 * layer1) + b2; //pembobotan w dan bias b L1
float out = layer2.at(0.0); // ambil nilai output
```

`normalizedN` adalah variabel dengan nilai input yang telah ternormalisasi. `w1`, `b1`, `w2`, dan `b2` adalah nilai w1, b1, w2, dan b2 untuk sistem ANN yang di dapatkan saat *preproccess*. `layer1` adalah matriks 10 x 1 untuk menampung data output dari layer 1(*hidden layer*). `Layer2` adalah matriks 1 x 1 untuk menampung data output dari layer 2(*output layer*). Variabel out digunakan untuk menyimpat data output dari pengolahan ANN agar tidak dalam bentuk matriks sehingga dapat dilakukan proses pengolahan selanjutnya.

Setelah didapatkan output dari ANN diperlukan proses denormalisasi untuk menjadikan nilai tersebut berada pada batas sesungguhnya. Untuk melakukan proses tersebut dibuat sebuah fungsi yaitu `denormalisasi(float in, float inMin, float inMax)`. Fungsi tersebut memeliliki 3 parameter. Parameter `float in` adalah data yang ingin diberikan proses denormalisasi. `float inMin` adalah batas minimal denormalisasi, `float inMax` adalah batas maksimal denormalisasi. Berikut adalah kode untuk fungsi denormalisasi pada C++:

```
float omnivision::denormalisasi(float in, float inMin, float inMax)
{
  float out = 0;
  out = 0.5*(in + 1) * (inMax - inMin) + inMin;
  return out;
}
```

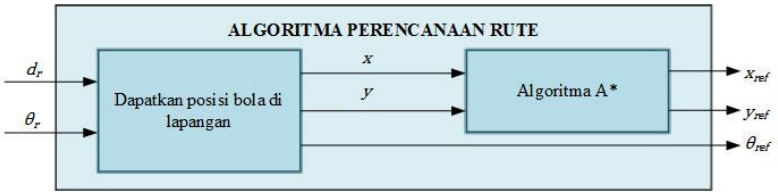
Proses denormalisasi ini adalah prosess akhir untuk mendapatkan jarak sesungguhnya dari bola. Selanjutnya informasi yang diperoleh dari serangkaian proses ini dapat digunakan untuk bermacam-macam keperluan lain seperti koordinasi, menentukan posisi bola di lapangan dan lain sebagainya.

3.2 Algoritma Perencanaan Rute

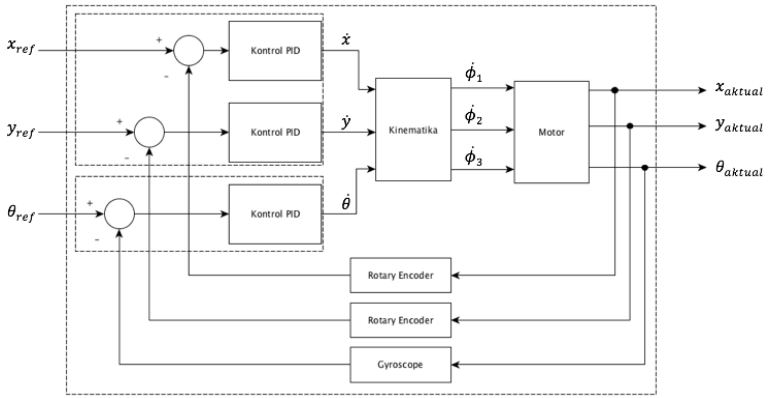
Algoritma perencanaan rute atau gerak robot di gunakan untuk membuat rencana pergerakan robot yang efektif untuk menuju sebuah titik tertentu. Pergerakan yang direncanakan juga mempertimbangkan adanya objek penghalanag semisal objek robot lain sehingga rute dibuat tidak boleh melewati robot tersebut. Pada robot ini algoritma A* digunakan untuk melakukan penentuan arahh gerak robot untuk mengejar bola. Diagram blok dari sistem ini ditunjukkan pada

3.3 Kontrol Pergerakan

Bagian ini berfungsi untuk menggerakkan robot menuju posisi yang diinginkan sesuai keluaran dari proses sebelumnya. Kontrol PID digunakna untuk mendapatkan posisi yang diinginkan. Feedback posisi robot berada menggunakan sistem odometry dengan rotary encoder yang dipasang di bagian bawah robot. Selain itu arah dari robot bisa diperoleh dengan menggunakan sensor IMU MPU 6050. Blok kontrol pergerakan lebih jelasnya ditunjukkan pada Gambar 3.24.



Gambar 3.23 Blok Diagram Algoritma Perencanaan Gerak



Gambar 3.24 Blok Kontrol Robot

3.3.1 Perancangan Sistem Gerak

Robot di desain dengan menggunakan penggerak 3 WD dengan menggunakan motor penggerak utama PG-45 (dapat dilihat pada) dengan tegangan kerja sebesar 24 volt dan kecepatan putar 400rpm. Motor ini telah dilengakapi dengan gearbox planetari dengan perbandingan gear 1:19. Sebagai driver motor untuk menentukan kecepatan dan arah putar motor digunakan modul driver motor BTN 7970. Lebih jelas tentang penggunaan driver motor ini akan dibahas pada subsubsubbab 3.5.10.

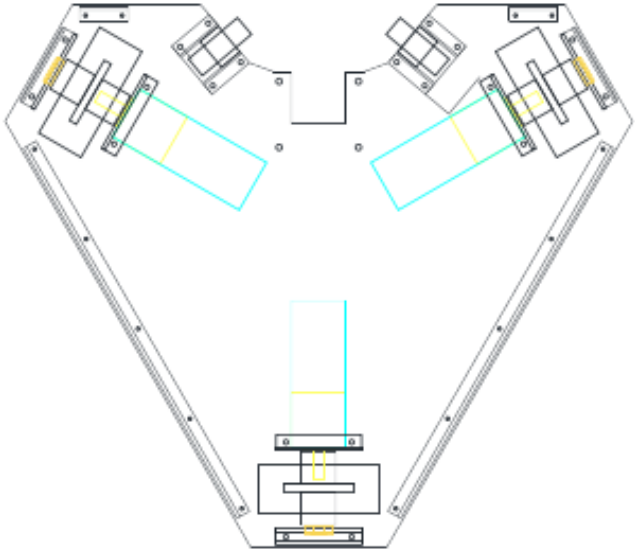
Robot di desain menggunakan sistem penggerak roda omni dengan jarak antar roda berjarak 120 derajat. Desain pemasangan motor pada

Gambar 3.25 Motor DC PG-45

bagain dasar robot dapat dilihat pada . Roda omni yang digunakan pada robot memiliki spesifikasi sebagai berikut:

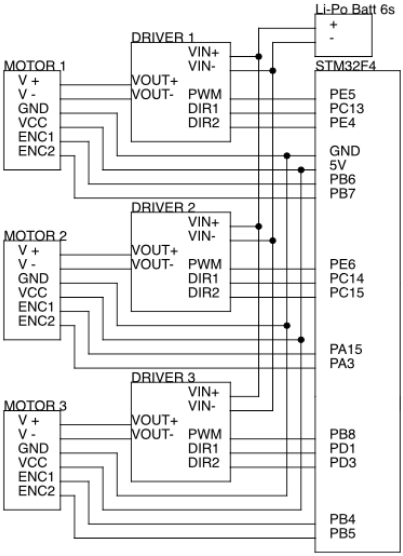
- Diameter : 10 cm
- Tebal 3 cm
- Jumlah piringan 2 buah
- Jumlah roller 5 buah / piringan
- Material bodi : Plastik
- Material roller : Karet
- Diameter roller

Untuk lebih jelasnya roda omni yang digunakan pada robot ditunjukkan pada Gambar 3.27.



Gambar 3.26 *Base* robot

Gambar 3.27 Roda Omni



Gambar 3.28 Interkoneksi antara Motor, Driver Motor dan STM32f4

Dengan desain penempatan motor seperti pada Gambar 3.26, akan diperoleh kinematika pergerakan robot sebagai berikut:

$$\begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \\ \dot{\phi}_3 \end{bmatrix} = \begin{bmatrix} -\sin(\theta) & \cos(\theta) & R \\ -\sin(\theta+\alpha_2) & \cos(\theta+\alpha_2) & R \\ -\sin(\theta+\alpha_3) & \cos(\theta+\alpha_3) & R \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \quad (3.9)$$

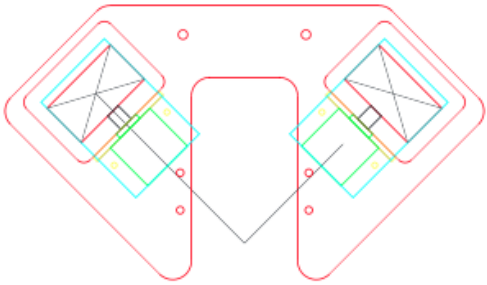
Interkoneksi antar motor penggerak, driver motor, dan mikrokontroller STM32f4 ditunjukkan pada Gambar 3.28.

3.3.2 Perancangan Sistem Odometri

Sistem odometri digunakan untuk mendapatkan posisi robot dilapangan dan dijadikan feedback untuk kontrol posisi. Sistem odometri dibentuk dengan menggunakan 2 rotary encoder yang dipasang pada bagian bawah robot. Rotary encoder yang digunakan pada robot dapat dilihat pada Gambar 3.45. Desain base untuk rotary encoder dapat dilihat pada Gambar 3.29. Peletakan rotary encoder satu dengan yang lainnya berjarak 90 derajat. Dengan desain tersebut pergerakan dari robot akan dapat dinyatakan dengan persamaan sebagai berikut:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -\cos(225° + \theta) & -\cos(135° + \theta) \\ -\sin(225° + \theta) & -\sin(135° + \theta) \end{bmatrix} \quad (3.10)$$

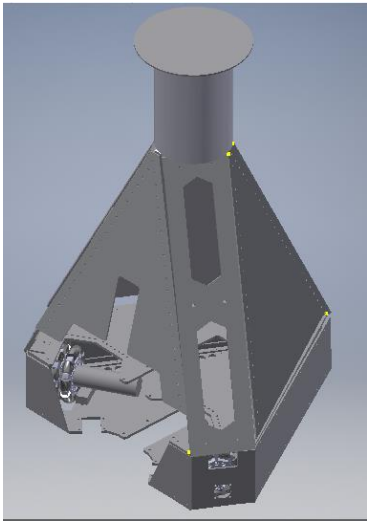
Sudut dari robot diperoleh melaui sensor IMU MPU 6050 dan dapat dilihat pada Gambar 3.39.



Gambar 3.29 Desain base untuk rotary encoder

3.4 Desain mekanik robot

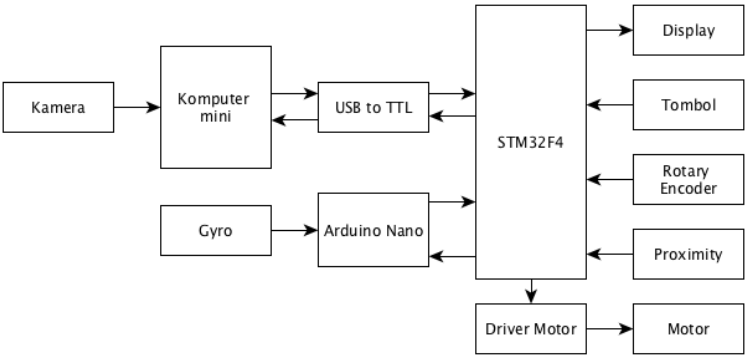
Mekanik robot didesain menggunakan bahan dasar alumunium padat untuk bodynya dan akrilik untuk beberapa bagian lain semisal penyangga sensor dan alat elektronik lainnya. Desain dari *base body* robot ditunjukkan pada Gambar 3.26. Robot di desain dengan menggunakan Robot sepak bola ini di desain mengerucut yaitu bagian bawah besar dan semakin ke atas semakin kecil dan disatukan oleh lempeng segitiga diatas robot. Dibagian atas diberikan silinder tabung akrilik sebagai pelindung kamera dan bagian paling atas adalah tempat kamera omnidireksional. Tujuan robot disesain mengerucut adalah memberikan kamera omnidireksional sudut pandang yang yang lebih luas serta meminimalisir titik buta terutama di daerah yang dekat dengan robot. Desain 3D dari robot dapat dilihat pada Gambar 3.30.



Gambar 3.30 Desain 3D Robot

3.5 Desain elektronik robot

Skema dari desain elektronik dari robot ditunjukkan pada Gambar 3.31. Sistem elektronik yang ditunjukkan pada gambar tersebut adalah sistem elektronik keseluruhan dari platfom robot yang dibuat sendiri. Namun pada tugas akhir kali ini fokus pengerjaannya pada bagian kamera dan komputer mini.



Gambar 3.31 Blok diagram sistem elektronik robot

3.5.1 Supply

Sumber energi utama dari robot adalah 2 baterai lipo 6 cell dan 1 baterai lipo 3 cell. Dari ketiga baterai tersebut 1 baterai lipo 6 cell digunakan sebagai sumber daya mini PC. Dikarenakan mini PC memiliki tegangan kerja 19 V maka tegangan baterai lipo 6 cell yang dengan kapasitas penuh yaitu 25.2 V terlebih dahulu harus di regulasi menjadi 19 V. Untuk melakukan proses regulasi tersebut digunakan modul buck converter variabel seperti ditunjukkan pada Gambar 3.43 dan akan dibahas lebih jelas pada subsubsubbab 0. Selain itu 1 baterai lipo 6 cell digunakan sebagai sumber daya untuk ketiga motor penggerak. 1 baterai

Gambar 3.32 Baterai Lipo Sebagai sumber energi utama robot

lipo 3 cell digunakan sebagai sumber daya dari sistem elektronik seperti sensor dan mikrokontroller. Sistem elektronik membutuhkan 2 nilai sumber tegangan yang berbeda yaitu 5v dan 12 V. Oleh karena itu diperlukan sebuah regulator untuk meregulasi tegangan menjadi 5v. Sedangakan tegangan 12V diperoleh langsung dari tegangan baterai. Meskipun tegangan baterai penuh mancapai 12.6V, namun tegangan tersebut masih berada di area kerja sesor yang digunakan sehingga tetap aman untuk digunakan. Desain regulator 5V menggunakan IC regulator linear LM7805 dengan tambahan transistor TIP3055 sebagai penguat arus. Sistem tersebut akan dibahas lebih jelas pada subsubsubbab 0.

3.5.2 Mini PC Intel Nuc

Mini PC Intel Nuc adalah CPU yang digunakan untuk mengambil dan memroses gambar dari kamera sehingga diperoleh data yang diinginkan terutama posisi bola di lapangan. Selain itu mini PC ini dijadikan sebagai pemroses utama dari data-data yang telah didapat dari seluruh sensor untuk menentukan aksi yang harus dilakukan robot baik berupa gerakan robot ataupun tendangan.

Spesifikasi dasar dari Intel Nuc yang digunakan adalah sebagai berikut:

- Prosesor Intel® Core™ i7-6770HQ (6M Cache, hingga 3.50 GHz)
- RAM 8GB DDR4
- M2 SSD 120GB
- TDP 45W
- Input DC 19V

Gambar 3.33 Mini PC Intel NUC 5i7 KYK

- 4 PORT USB
- HDMI

3.5.3 STM32f4

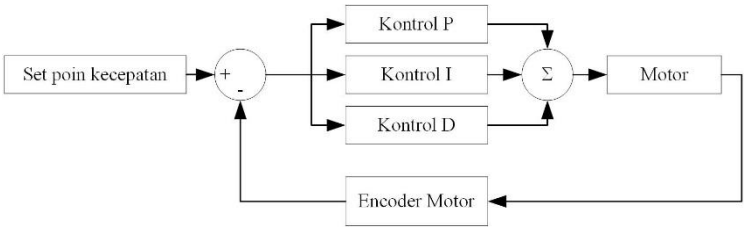
Pada tugas akhir ini STM32f4 digunakan sebagai slave dari Master mini PC intel Nuc. STM32f4 berfungsi sebagai perantara antara sensor-sensor dengan PC. Tugas dari STM32f4 adalah untuk menakses sensor serta mendapatkan data dari sensor tersebut serta mengumpulkan data dari sistem mikro kontroller lain untuk lebih lanjut dikirimkan datanya ke PC melalui USB to TTL. Sensor atau mikrokontroler lain yang terkait dengan STM32f4 diantaranya adalah sensor proximity, rotary encoder, dan encoder motor. Selain itu peran dari STM32f4 adalah menerima data hasil olahan PC untuk menggerakkan motor dan akuator lainnya.

Untuk pergerakan motor PC mengirimkan data berupa kecepatan arah sumbu x, kecepatan arah sumbu y, kecepatan rotasi, arah depan robot, dan mode gyro(digunakan untuk menentukan apakah arah sumbu cartesian berdasarkan robot ata berdasarkan lapangan). Lebih lanjut dari data yang diperoleh tersebut STM32f4 mengolah nilai-nilai tersebut menggunakan invers kinematik sesuai dengan persamaan berikut:

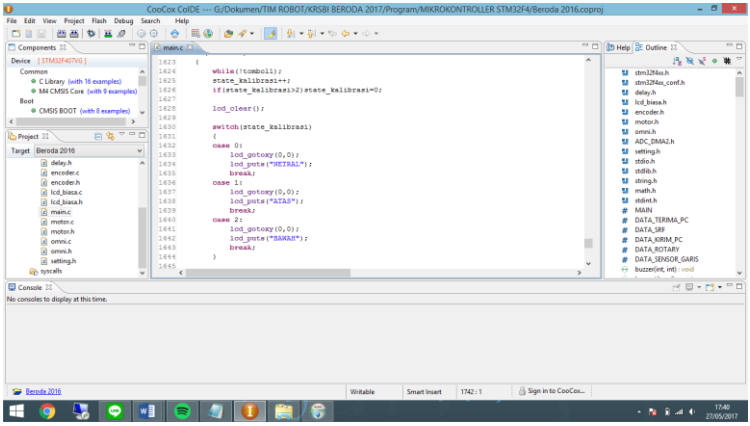
$$\begin{aligned} Ref_1 &= K\left(C\omega - v_x sin30° - v_y cos30°\right) \\ Ref_2 &= K\left(C\omega - v_x sin30° + v_y cos30°\right) \\ Ref_3 &= K(C\omega + 2v_x) \end{aligned} \tag{3.11}$$

Dari persamaan tersebut didapatkan kecepatan kecepatan-kecepatan yang harus diberikan kepada motor.

Untuk mendapatkan kecepatan motor yang sesuai tugas STM32f4 adalah melakukan proses kontrol terhadapat kecepatan motor dengan feedback sensor encoder motor. Kecepatan motordidapatkan dengan memanfaatkan counter pada timer STM32f4 yang nilainya di ambil dan kemudian di reset dengan periode tertentu. Kontrol kecepatan yang dilakukan oleh STM32f4 menggunakan PID dengan diagram blok ditunjukkan pada Gambar 3.34.

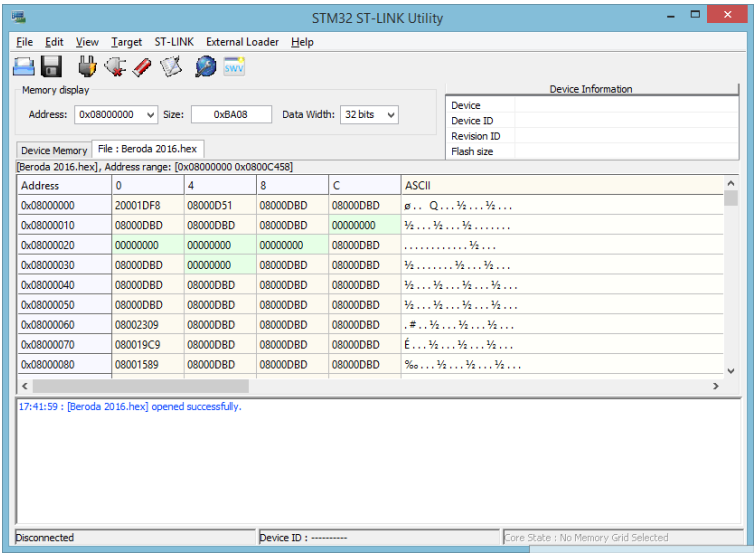


Gambar 3.34 Diagram kontrol PID pada motor



Gambar 3.35 Tampilan IDE Coocox

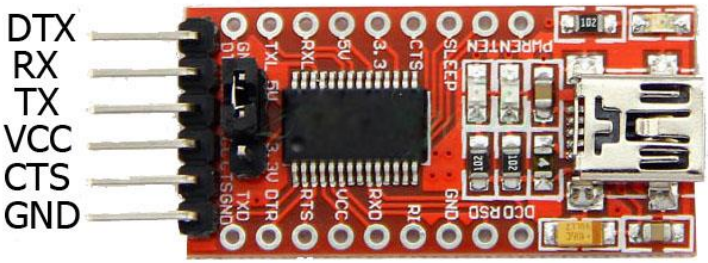
Proses pemrograman stm32f4 menggunakan software IDE coocox. Compiler yang digunakan adalah "GNU Tools for ARM Embedded Processor v 5.4". Kompiler ini membantu IDE Cocox untuk menghasilkan file hex dari kode yang telah dibuat. Proses mendownload program dilakukan dengan menggunakan kabel USB to mini-USB. Selain itu proses download juga dibantu oleh software ST-LINK untuk membuka file hex dan mendownloadnya ke chip mikrokontroler.



Gambar 3.36 Tampilan Software ST-LINK

3.5.4 USB to TTL

Perangkat USB to TTL digunakan sebagai alat bantu komunikasi antara mikro kontroller STM32F4 dan mini PC. Melalui alat ini data-data dari STM32f4 dapat dikirimkan k Mini PC dan begitu pula sebaliknya. Cara kerja alat ini adalah dengan merubah data USB menjadi level



Gambar 3.37 USB to TTL

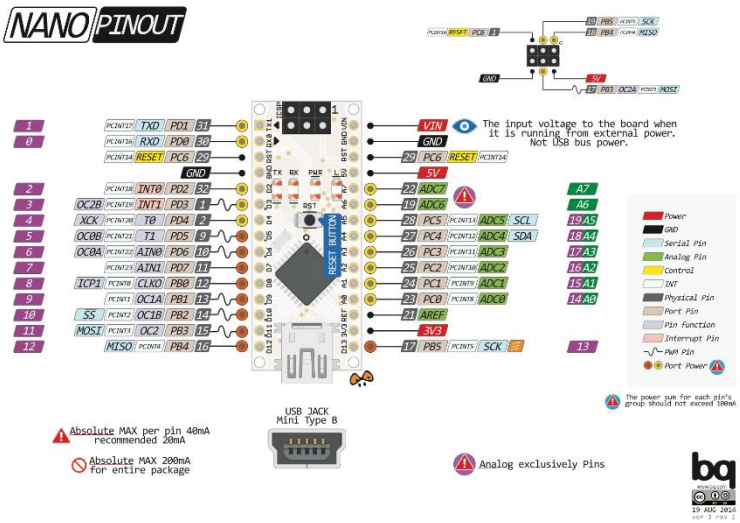
tegangan TTL begitu pula sebaliknya. Seri USB to TTL yang dipakai pada TA kali ini adalah FTDI FT232R. Penggunaan alat ini adalah dengan menghubungkan alat tersebut dengan komputer melalui USB dan menhubungkan device melaui pin TX RX dengan device lain seperti mikrokontroler atau sejenisnya.

3.5.5 Arduino Nano

Pada tugas akhir kali ini arduino nano digunakan sebagai pengakses dan pengolah data dari sensor IMU MPU6050. Setelah selesai diolah data tersebut kemudian dikirimkan ke STM32f4. Data yang dikirimkan hanyalah data rotasi robot pada sumbu z. Karena data rotasi pada sumbu yang lain tidak digunakan pada robot ini. Data rotasi tersebut selanjutnya akan digunakan pada proses perhitungan odometri dan penentuan orientasi robot untuk menentukan dimana arah gawang lawan. Dengan demikian diharapkan robot tidak akan melakukan gol bunuh diri.

3.5.6 Sensor IMU

Sensor IMU yang digunakan pada tugas akhir kli ini adalah MPU 6050. Sensor ini adalah sensor MEMS(*Micro Electro Mechanical System*).



Gambar 3.38 Arduino nano **[16]**

Sensor ini berkomuniasi dengan perangkat lain menggunakan interface I2C. Sensor ini adalah salah satu sensor cerdas karena dapat menyediakan data yang telah diolah melalui sebuah pemrosesan internal yang lebih dikenal dengan DMP(*Digital Motion Processor*). Sensor ini mengkombinasikan antara 3 axis gyroskop dan 3 axis akselerometer dalam sebuah chip kecil. Selain itu dapat pula ditambahkan sensor magnetometer eksternal melalui jalur auxiliary master I2C sehingga memungkinkan divais mendapatkan data lengkap tanpa mengganggu sistem prosesor [17]. Konfigurasi pin saat penggunaan pada robot adalah pin VCC dihubungkan dengan supply 5V sedangakan pind GND disambungkan dengan GND pada supply. Untuk komunikasi pin SDA

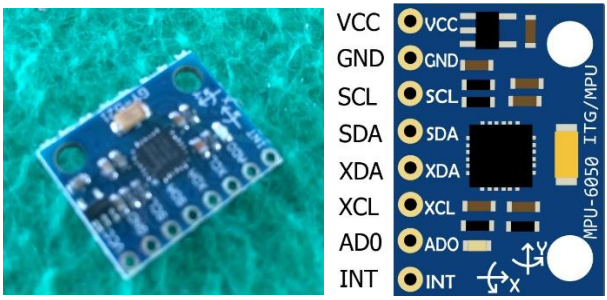
dihubungkan dengan pin SDA pada arduino yaotu pada pin A4. Sedangkan pin SCL pada modul dihubungkan dengan pin SCL pada arduino yaitu pada pin A5.

3.5.7 Sensor garis

Sensor garis pada sistem robot digunakan sebagai alat kalibrasi posisi robot terhadap lapangan memanfaatkan garis-garis yang disediakan. Sensor garis yang digunakan pada robot kali ini adalah sensor autonic tipe BF5R. Output dari sensor ini dihubungkan dengan pin logic

pada STM32F4 yang diberi pull up internal sehingga mendeteksi logic 0 saat mendeteksi garis dan logic 1 saat tidak mendeteksi garis.

Konfigurasi dari sensor garis ini adalah GND dihubungkan dengan GND supply, VCC pada sensor dihubungkan dengan supply 12 V, sedangkan out pada sensor dihubungkan dengan pin logic pada STM32F4.



Gambar 3.39 MPU-6050 dan pin outnya

Gambar 3.40 *Wire out* modul sensor garis

Gambar 3.41 Modul Sensor Garis

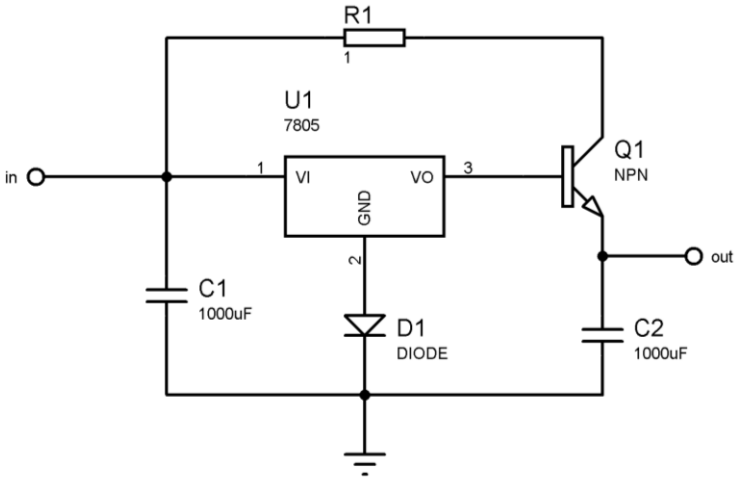
3.5.8 Regulator Sistem Mikrokontroler dan sensor

Sistem mikrokontroller dan sensor beberapa memerlukan tegangan sebesar 5V. Tegangan tersebut tidak dapat diberikan langsung oleh sumber baterai Lipo 3 Sel yang memiliki tegangan penuh sebesar 12.6V. oleh karena itu diperlukkan sebuah regulator tegangan agar dapat dihasilkan tegangan 5V dengan input tegangan dari baterai 3 sel.

Desain dari regulator di tunjukkan pada Gambar 3.42. Regulator didesain menggunakan regulator linear 7805 dengan tambahan transistor penguat arus TIP 3055. Resistor R1 digunakan untuk membantu transitor TIP 3055 untuk mendisipasi daya dalam bentuk panas sehingga dapat meminimalisir disipasi daya panas pada transistor TIP 3055. Dioda D1 diberikan dengan tujuan menaikkan tegangan yang dihasilkan regulator

sebesar tegangan forward dioda atau sekitar 0.7 V. Sehingga tegangan keluaran dari dioda terhadap ground adalah sebesar 5.7 V. Tegangan ini akan mengalami drop saat melalui transistor sebesar V_{BE} yang pada umumnya adalah sebesar 0.7. Sehingga tegangan output dari

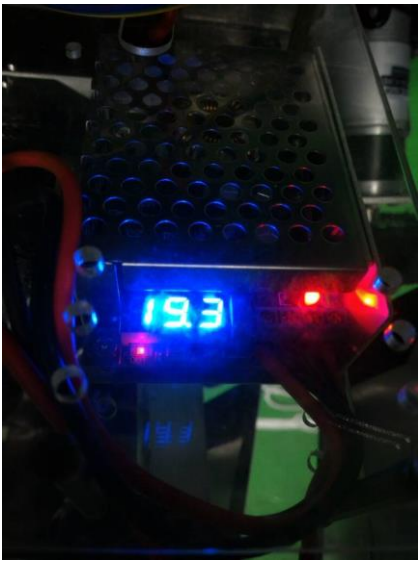
sistem regulator ini adalah sebesar 5V.



Gambar 3.42 Desain Regulator

3.5.9 Buck Converter untuk Mini PC

Buck converter ini digunakan sebagai reguulator tegangan dari sumber utama yaitu baterai lipo 6 Sel. Mini PC intel Nuc yang digunakan pada tugas akhir kali ini memiliki tegangan kerja 19V DC. Oleh karena itu diperlukan sebuah regulator untu menurunkan tegangan penuh baterai Lipo 6 Sel yaitu 25.2 V menjadi 19V. Tagangan input dari buckk converter ini adalah 10 – 45V. Arus maksimal yang mampu dihasilkan dari buck converter ini adalah sebesar 10 A.



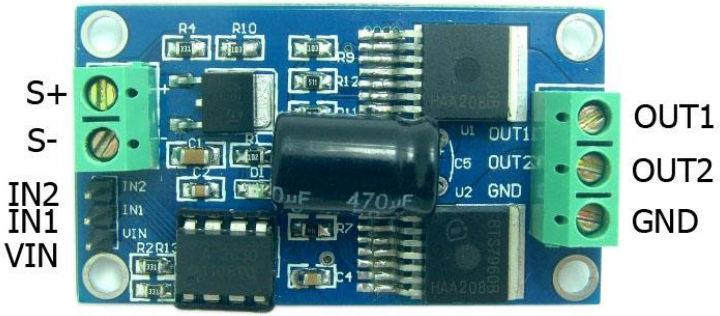
Gambar 3.43 Modul Buck Converter

3.5.10 Driver Motor

Driver motor digunakan untuk mengaktifkan motor serta menentukan arah putar motor sesuai sinyal diberikan oleh mikro kontroler. Driver motor yang digunakan pada tugas akhir kali ini adalah module driver motor BTN7970 sebanyak 3 buah. Modul driver motor ini terdiri dari 2 IC *half-bridge* BTN7970 sehingga memungkinkan untuk melakukan kontrol putaran dua arah baik serah jarum jam ataupun berlawanan arah jarum jam. Driver motor ini memiliki kemampuan mengalirkan arus hingga 50A.

Modul driver motor ini memiliki 5 input yang terdiri dari 3 pin kontrol (VIN, IN1, dan IN2) dan 2 pin supply (+ dan -). Selain itu modul ini memiliki 3 output yang ber label GND, OUT1, dan OUT2.

Modul ini memeliki kemampuan untuk men-*drive* 2 motor sekaligus dengan menyambungkan input motor masing-masing ke output OUT1 – GND dan OUT2 – GND. Kontol dilakukan melaui pin VIN sebagai enable serta IN1 dan IN2 sebagai kontrol kecepatan biasanya menggunakan PWM. Namun jika itu dilakukan setiap motor hanya dapat berputar ke satu arah karena masing-masing motor hanya di *drive* oleh



Gambar 3.44 Modul Driver Motor BTN 7970

satu IC half bridge. Untuk melakukan kontrol full dua arah satu motor harus dihubungkan dengan output OUT1-OUT2. Kontrol kecepatan dan arah putar motor dilakukan melalui pin kontrol IN1 dan IN2. Jika IN 1 berlogika 1 dann IN2 berlogika 0 maka arahh putar motor adalah berlawanan arah jarum jam sedangkan jika input IN1 berlogika 0 dan input IN2 berlogika 1 maka arah putaran motor adalah serah dengan jarum jam. Selain itu input VIN digunakan sebagai kontrol kecepatan menggunakan PWM. Jika PWM memiliki duty cycle rendah maka

kecepatan motor relatif pelan sedangakan jika PWM memiliki duty cycle tinggi kecepatan motor relatif tinggi.

3.5.11 Rotary encoder

Rotary encoder yang digunakan pada tugas akhir kali ini adalah autonic seri E30. rotary encoder digunakan sebagai komponen untuk odometry robot yang akan menjadi *feedback* posisi robot. Rotary encoder yang digunakan pada tugas akhir ini memiliki spesifikasi tegangan kerja 5 - 24V DC ± 5% dan 200 pulsa per rotasi.

Terdapat 4 kabel yang menjuntai dari sensor tersebut masing-masing adalah supply +(VCC), supply –(GND), A, dan B. Supply + dan supply – adalah jalur supply untuk sensor yang pada desain kali ini dihubungkan langsung pada baterai lipo 3 sel. Sedangkan A dan B adalah output dari sensor yang dihubungkan pada counter melalui pin di STM32F4 dimana pulsa dari kedua output itu berbeda fasa sebesar 90 derajat. Perbedaan fasa ini digunakan sebagai tanda arah putaran dari motor teresebut apakah berlawanan ataukah serah dengan arah putaran jarum jam.

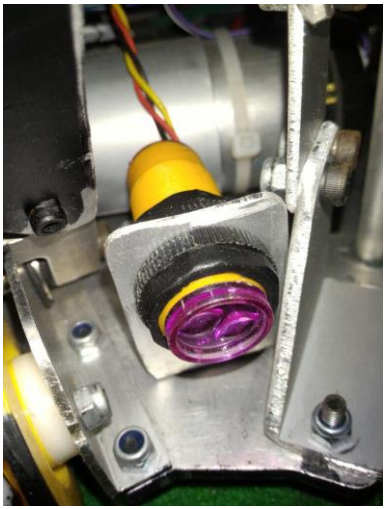
Gambar 3.45 *Rotary Encoder*

Gambar 3.46 *Wire out* dari rotary encoder

3.5.12 Sensor IR proximity

Sensor IR proximity digunakan untuk mendeteksi ada atau tidaknya objek. Sensor ini bekerja dengan tegangan 5V. Terdapat potensiometer yang dapat digunakan untuk mengatur batas *threshold* pendeteksian objek. Jika terdapat objek yang berjarak kurang dari batas *threshold* maka sensor akan mengeluarkan logika 1 dan ditandai dengan adanya led merah yang menyala pada sensor. Sedangakan saat tidak ada objek yang berjarak kurang dari batas *threshold* sensor, maka sensor akan mengelurkan logika 0 dan juga ditandai led merah pada sensor mati. Konfigurasi dari sensor ini adalah VCC dihubungkan pada supply 5V dan GND dihubungkan pada supply GND. Sedangkan out pada sensor dihubungkan dengan pin digital pada STM32F4 untuk membaca output logic dari sennsor tersebut.

Pada tugas akhir kali ini sensor IR proximity digunakan untuk mendeteksi ada atau tidaknya bola pada penggiring bola. Informasi ini



Gambar 3.47 Sensor IR Proximity

Gambar 3.48 *Wire Out* Sensor Proximity

akan digunakan lebih lanjut oleh robot untuk mengaktifkan penggiring, membawa bola ke gawang lawan, melakukan proses umpan dan lain sebagainya.

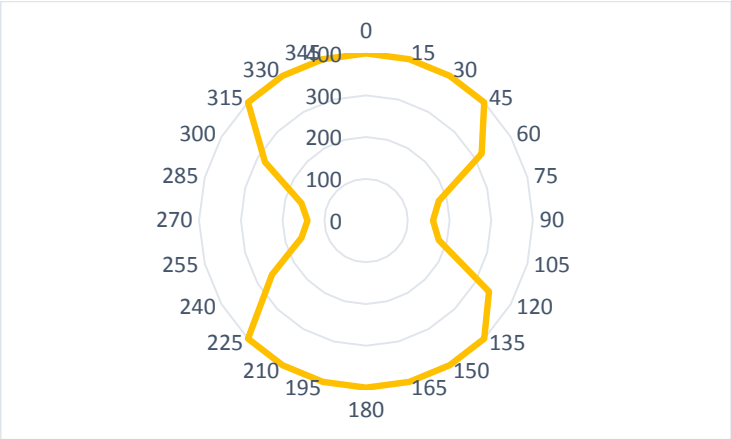
BAB 4
PENGUJIAN

Pada bab ini berisi tentang pengujian dari sistem yang telah dibuat. Pengujian ini bertujuan untuk mengetahui tingkat ketercapaian tujuan dari sistem yang dirancang. Pengujian bab ini meliputi pengujian jarak pengukuran dalam satuan pixel. Pengujian pengukuran jarak sistem ANN satu input satu output, dan pengujian pengukuran jarak sistem ANN satu input satu output dengan penambahan offset.

4.1 Uji pendeteksian bola

Beriku disajikan data pengujian pendeteksian bola dari sistem yang telah dibuat. Pengujian dilakukan dengan meletakkan bola pada jarak 40 cm sampai 400 cm dengan dilakukan setiap interval 20 cm. Percobaan tersebut dilakukan disetiap sisi kamera mulai dari sudut 0 sampai 180 derajat dilanjutkan dari -180 derajat samapai 0 dan dilakukan dengan interval sudut 15 derajat.

Data dari pengujian pendeteksian bola direpresentasikan pada grafik berikut ini:



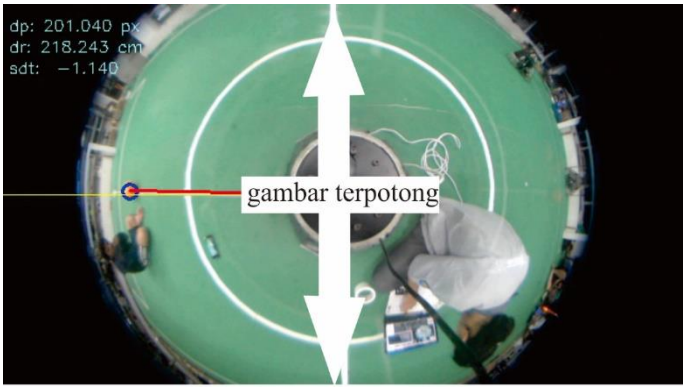
Gambar 4.1 Uji Pendeteksian Bola

Dari percobaan yang dilakukan, sistem mampu mendeteksi bola dari jarak 40 – 400 cm. Namun pada sudut tertentu kamera sistem tidak mampu mendeteksi bola dikarenakan bola tidak dapat terlihat pada citra. Hal ini dikarenakan perbandingan resolusi dari kamera yang digunakan adalah *wide* yaitu 16 : 9. Hal ini menyebabkan citra kamera bagian atas dan bawah terpotong. Lebih jelasnya dapat dilihat pada Gambar 4.2 dibawah.

4.2 Pengujian hubungan jarak pixel dan jarak sesungguhnya

Pada bagian ini dilakukan pengujian untuk mendapatkan relasi dan karakteristik dari data jarak bola terdeteksi terhadap titik tengah robot dengan satuan pixel dibandingkan dengan jarak sesungguhnya. Berikut disajikan data percobaan dari pengujian tersebut disajikan pada Tabel 4.1 sedangkan plot dari hubungan antara jarak pixel dan jarak sesungguhnya disajikan pada Gambar 4.3.

Dari tabel dan grafik yang telah disajikan di atas nampak bahwa bahwa karakteristik hubungan antara jarak sesungguhnya dan jarak yang terukur dengan satuan pixel kamera menunjukkan sifat eksponensial. Pada nilai pixel rendah perubahan jarak dalam satuan pixel yang sedikit mengindikasikan perubahan jarak yang sedikit pula pada jarak sesungguhnya. Namun pada nilai jarak satuan pixel yang besar, perubahan kecil mengindikasika perubahan jarak yang signifikan pada jarak yang sesungguhnya. Hal ini menunjukkan bahwa sistem memiliki

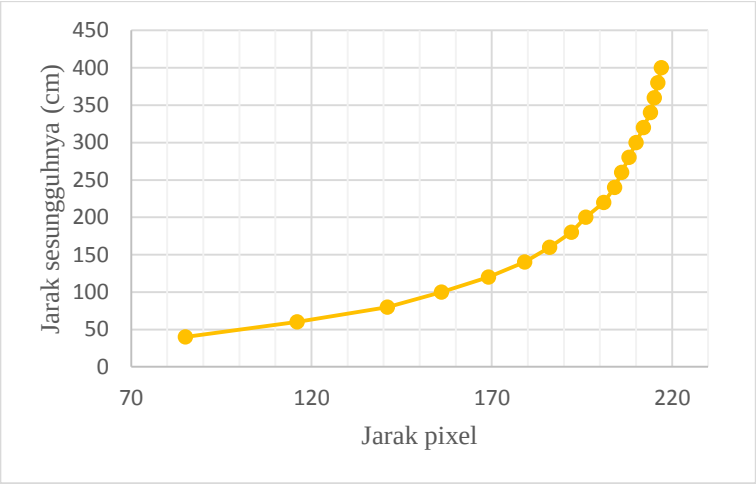


Gambar 4.2 Citra terpotong pada bgaian atas dan bawah

ketelitian pengukuran yang semakin kecil seiring bertambahnya jarak bola terhadap robot.

Tabel 4.1 Hubungan jarak pixel dan jarak sesungguhnya

No	jarak pixel(dp)	jarak sesungguhnya(cm)
1	85.024	40
2	116.039	60
3	141.032	80
4	156.051	100
5	169.074	120
6	179.07	140
7	186.043	160
8	192.042	180
9	196.064	200
10	201.04	220
11	204.022	240
12	206.039	260
13	208.038	280
14	210.038	300
15	212.038	320
16	214.021	340
17	215.021	360
18	216.037	380
19	217.021	400



Gambar 4.3 grafik hubungan jarak pixel dan jarak sesungguhnya

4.3 Pengukuran Jarak Dengan Sistem ANN

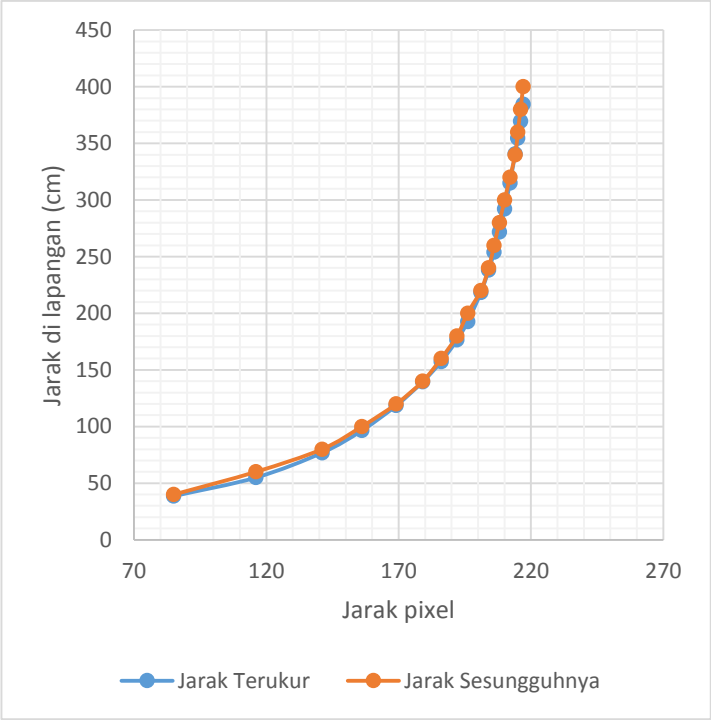
Berikut ini adalah hasil pengujian sistem pengukuran jarak dengan menggunakan sistem ANN. Bobot w dan b dihasilkan dari preproccess di matlab. Data berikut adalah hasil uji sistem ANN yang dijalankan dengan menggunakan Visual Studio saat running program secara *realtime*. Data Uji disajikan pada tabel dibawah ini.

Tabel 4.2 Hasil Uji ANN

No	dp (cm)	dm (cm)	dr (cm)	error(%)
1	85.024	38.593	40	3.5175
2	116.039	54.943	60	8.428333333
3	141.032	76.832	80	3.96
4	156.051	96.418	100	3.582
5	169.074	118.517	120	1.235833333
6	179.07	139.487	140	0.366428571
7	186.043	157.301	160	1.686875
8	192.042	176.401	180	1.999444444
9	196.064	192.496	200	3.752

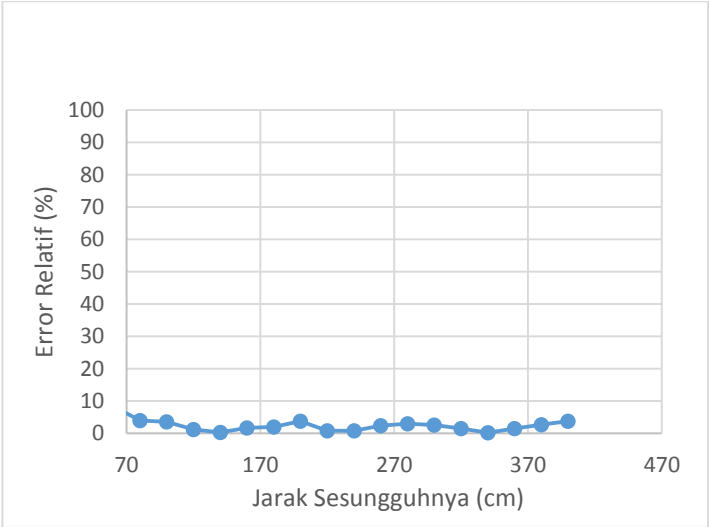
10	201.04	218.243	220	0.798636364
11	204.022	238.053	240	0.81125
12	206.039	253.847	260	2.366538462
13	208.038	271.734	280	2.952142857
14	210.038	292.09	300	2.636666667
15	212.038	315.119	320	1.5253125
16	214.021	340.724	340	0.212941176
17	215.021	354.683	360	1.476944444
18	216.037	369.577	380	2.742894737
19	217.021	384.651	400	3.83725
		Error rata-rata		***2.520473257***

Dari pengujian yang telah dilakukan didapatkan bahwa sistem ANN yang dibuat telah mampu mengikuti karakteristik hubungan antara jarak bola terukur dalam pixel dan jarak bola sesungguhnya. Hal ini ditunjukkan pada Gambar 4.4 yaitu jarak terukur dan jarak sesungguhnya pada rentang 40-400 cm menunjukkan selisih yang relatif kecil. Dari pengujian tersebut terlihat bahwa pengukuran tidak benar-benar sempurna. Pengukuran menunjukkan nilai error relatif rata-rata sebesar 2.52 %. Selain itu pada gambar Gambar 4.5 menunjukkan grafik perbandingan antara jarak terukur dan jarak sesungguhnya yang relatif linear. Hal ini menunjukkan bahwa sistem yang telah dibangun mampu melinearkan hasil pengukuran.



Gambar 4.4 Grafik perbandingan jarak terukur dan jarak sesungguhnya

Gambar 4.5 Grafik perbandingan jarak hasil pengukuran dan jarak sesungguhnya



Gambar 4.6 Grafik error relatif terhadap jarak sesungguhnya

Halaman ini sengaja dikosongkan

BAB 5
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan yang dapat diambil oleh penulis dari serangkaian pengujian yang telah dilakukan adalah sebagai berikut:

1. Sistem kamera 360 derajat mampu mendeteksi bola pada jarak 40 – 400 cm dari robot. Namun di sudut tersentu terhadap depan robot, sistem hanya mampu mendeteksi bola pada jarak maksimal 160 cm dikarenakan citra gambar yang terpotong. Hal ini disebabkan spesifikasi kamera yang digunakan adalah *wide resolution* sehingga bagian atas dan bawah citra sudut pandangnya terbatas.
2. Karakteristik hubungan antara jarak sesungguhnya dan jarak yang terukur dengan satuan pixel kamera menunjukkan sifat eksponensial. Pada nilai pixel rendah perubahan jarak dalam satuan pixel yang sedikit mengindikasikan perubahan jarak yang sedikit pula pada jarak sesungguhnya. Namun pada nilai jarak satuan pixel yang besar, perubahan kecil mengindikasika perubahan jarak yang signifikan pada jarak yang sesungguhnya. Hal ini menunjukkan bahwa sistem memiliki ketelitian pengukuran yang semakin kecil seiring bertambahnya jarak bola terhadap robot.
3. Sistem ANN mampu mengikuti karakteristik hubungan antara jarak bola yang terukur dengan satuan pixel dan jarak sesungguhnya dengan kesalahan relatif pengukuran rata-rata sebesar 2.52 %.

5.2 Saran

Beberapa saran yang dapat diberikan penulis untuk pengembangan tugas akhir ini adalah sebagai berikut:

1. Penggunaan lensa fish eye tugas akhir ini dapat digantikan dengan penggunaan cermin atau lensa fish eye lain yang lebih luas sehingga sistem mampu mendeteksi bola dalam lingkup yang lebih luas.
2. Penggunaan metode learning back propagation cenderung membutuhkan waktu relatif lama. Oleh karena itu dapat digunakan metode lain selain back propagation yang dapat mempelajari karakteristik hubungan jarak terukur dengan

satuan pixel kamera terhadap jarak sesungguhnya dengan lebih cepat dan efektif.

DAFTAR PUSTAKA

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

[2] A. K. Gary Bradski, Learning OpenCV, Sebastopol: O'Reilly Media, Inc, 2008, p. 60.

[3] T. d. V. K. Kumar, "A Theory Based on Conversion of RGB Image to Gray Image," *Internaional Journal of Computer Application,* vol. 7, pp. 7-10, 2010.

[4] V. K. Kumar T, "A Theory Based on Conversion of RGB Image to Gray Image," *International Journal of Computer Applications,* vol. 7, no. 2, pp. 7-10, 2010.

[5] OpenCV, "OpenCV," OpenCV, 2017. [Online]. Available: http://opencv.org/. [Diakses 31 Mei 2017].

[6] L. Fausett, Fundamentals of Neural Network, Prentice Hall, 1994.

[7] H. W, Applied Nonparametric Regression, Cambridge: Cambridge University Press, 1990.

[8] openFramework, "openFramework," 14 April 2017. [Online]. Available: http://openframeworks.cc/. [Diakses 18 5 2017].

[9] B. Jati Utomo, "Rancang Bangun UAV (Unmanned Aerial Vehicle) Model Quadcopter Dengan Menggunakan Algoritma Proportional Integral Derivative," Universitas Telkom, Bandung, 2014.

[10] STMicroelectronics, "STM32F4DISCOVERY," STMicroelectronics, 2017. [Online]. Available: http://www.st.com/content/ccc/resource/technical/document/user_manual/70/fe/4a/3f/e7/e1/4f/7d/DM00039084.pdf/files/DM00039084.pdf/jcr:content/translations/en.DM00039084.pdf. [Diakses 12 Juli 2017].

[11] E. Hobby, "Schematic Circuit," 18 April 2017. [Online]. Available: http://schematiccircuit.com/optical-rotary-encoder/. [Diakses 12 Juli 2017].

[12] I. Corporation, "INTEL® NUC KIT NUC6I7KYK," Intel Corporation, [Online]. Available:

https://www.intel.com/content/www/us/en/products/boards-kits/nuc/kits/nuc6i7kyk.html. [Diakses 12 Juli 2017].

13] R. Semiconductor, Controlling DC Brush Motors with H-Bridge Driver ICs, San Diego: ROHM Semiconductor, 2009.

14] openCV, "Miscellaneous Image Transformations," openCV, 28 Mei 2017. [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html. [Diakses 29 Mei 2017].

15] MathWorks, "Create Arrays of Random Numbers," MathWorks, 2017. [Online]. Available: https://www.mathworks.com/help/matlab/math/create-arrays-of-random-numbers.html. [Diakses 28 Mei 2017].

16] Arduino, "Arduino NANO Pinout Diagram," Arduino, 2017. [Online]. Available: https://forum.arduino.cc/index.php?topic=147582.0. [Diakses 12 Juli 2017].

17] T. InvenSense, "MPU-6050 Six-Axis (Gyro + Accelerometer) MEMS MotionTracking™ Devices," TDK InvenSense, 2017. [Online]. Available: https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/. [Diakses 27 Mei 2017].

18] C. Saravanan, "Color Image to Grayscale Image Conversion," dalam *Second International Conference on Computer Engineering and Applications*, Durgapur, 2010.

19] Satriotiotioso, "Hubungan Antara Saturation, Luminance, Hue, Contrast dan Brightness dengan RGB Color Model," Unair, 5 Juni 2012. [Online]. Available: http://satriotiotioso-fst09.web.unair.ac.id/artikel_detail-47728-Komputasi%20Biomedis-Hubungan%20Antara%20Saturation,%20Luminance,%20Hue,%20Contrast%20dan%20Brightness%20dengan%20RGB%20Color%20Model.html. [Diakses 18 5 2017].

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

LAMPIRAN

Program matlab untuk learning

```
% --- input and target of learning set
p1 = [-2]; p2 = [-1]; p3 =[0]; p4 = [1]; p5 = [2];
t1 = [-4]; t2 = [0]; t3 = [0]; t4 = [2]; t5 = [12];
po = [0 74.03 92.02 109.02 122.03 137.06 146.05 156.05
165.04 177.01 189.03 195.01 201 205 210 212 215 217 220 221
223 224 225 226 231];
to = [0 30 40 50 60 70 80 90 100 120 140 160 180 200 220 240
260 280 300 320 340 360 380 400 500];
% --- normalization of input adn target
pmin = min(min(po)); pmax = max(max(po));
tmin = min(min(po)); tmax = max(max(po));

p = normf(po,0,pmax);
t = normf(to,0,tmax);
%inisialisasi weight and bias
[rowp,colp] = size(p);
[rowt,~] = size(t);
ni  = rowp;
nt  = rowt;
nls = colp;
w1 = randgen(10,ni);
b1 = randgen(10,1);
w2 = randgen(nt,10);
b2 = randgen(nt,1);
%--- Learning process
lr = 0.1;
errtol=1; errtolmax=1e-6; epoch=0; epochmax=50000;
while errtol > errtolmax && epoch < epochmax
    tampil = ['epoch: ',num2str(epoch), ' error :',
num2str(errtol)];
    epoch = epoch + 1; disp(tampil);
    errtol = 0;
    for i = 1:nls
        % --- forward propagation
        a0 = p(:,i);
        a1 = fpnet(a0,w1,b1,1); % logsig TF
        a2 = fpnet(a1,w2,b2,0); % linear TF
        a = a2;
        % --- backward propagation
        e = t(:,i)-a;
        J = sse(e);
        if J > errtolmax/nls,
            % --- sensitivity calculation
            s2 = sout(a2,t(:,i),0);
            s1 = slayer(a1,w2,s2,1);
```

```
            % --- weight and bias update
            w2 = w2 - lr*s2*a1';
            b2 = b2 - lr*s2;
            w1 = w1 - lr*s1*a0';
            b1 = b1 - lr*s1;
        end
        errtol = errtol + J;
    end
end

disp('end of learning');
disp('w1 = ');disp(w1); disp('b1 = ');disp(b1);
disp('w2 = ');disp(w2); disp('b2 = ');disp(b2);
% --- verification process
a1 = fpnet(p,w1,b1,1);
a2 = fpnet(a1,w2,b2,0); disp(a2);
hasil = dnormf(a2,tmin,tmax);disp(hasil);

% --- uji
uji =0:0.1:231;
uji_n = normf(uji,0,pmax);


a1 = fpnet(uji_n,w1,b1,1);
a2 = fpnet(a1,w2,b2,0); disp(a2);
y = dnormf(a2,tmin,tmax);disp(y);

%plot(po,to,uji,y), legend('asli','ANN');
plot(po,to);
```

Program Visual studio

Omnivision.h

```
#pragma once
#include "Timer.h"
#include "ofThread.h"
#include <armadillo>
#include <opencv2\opencv.hpp>
#include "struct.h"
#include "filter.h"

using namespace cv;
using namespace std;

class omnivision :public ofThread
{
private:
	int width, height, center_x, center_y;
	Mat frame, frameHSV, ball, field, obstacle;
	VideoCapture cap;

	int blur, opening_ball, closing_ball, opening_field,
closing_field,
		hl_ball, hh_ball, sl_ball, sh_ball, vl_ball, vh_ball,
		hl_field, hh_field, sl_field, sh_field, vl_field,
vh_field;

	int x_circle, y_circle, radius_circle,
		x_circle2, y_circle2, radius_circle2, thickness_circle2,
		hl_obstacle, hh_obstacle, sl_obstacle, sh_obstacle,
vl_obstacle, vh_obstacle,
		x1_line1, y1_line1, x2_line1, y2_line1, t1, x1_line2,
y1_line2, x2_line2, y2_line2, t2;;

	bool object;

	void find_ball(Mat, Mat, Mat &_frame, int &_x, int &_y);
	void threshold_object_distance();
	void find_obstacle(vector<int> &_object_distance, vector<int>
&_object_tetha);
	void find_closer_object(vector<int> &_object_distance, vector<int>
&_object_tetha, int& _jarak_objek_terdekat, int& _sudut_objek_terdekat);
	void find_closer_object2(float* _object_distance, float*
_object_tetha, float threshold);
	void find_obstacle2(float* _object_distance, float*
_object_tetha);
	float convert_distance2real(float _distance_pixel);
	void open_setting();
	void open_setting_obstacle();
	void save_setting();
	void save_setting_obstacle();
	float normalisasi(float in, float inMin, float inMax);
	float denormalisasi(float in, float inMin, float inMax);
```

```
        bool arah_tendang_penalty();
        int garis[4] = {0, 0, 0, 0};

        int font = FONT_HERSHEY_SIMPLEX;
        float tempFPS = 0;
        char FPS[10];
        Timer FPSCOUNT;

        //array filter
        float array_filter_sudut_obstacle[1000];

public:
        omnivision();
        ~omnivision();
        void threadedFunction();
        void setup(int _n = 0, int _width = 640, int _height = 640);
        float ball_distance_pixel, ball_tetha,ball_distance_real;
        vector<int> object_distance, object_tetha;
        int jarak_objek_terdekat_pixel, jarak_objek_terdekat_real,
sudut_objek_terdekat;
        bool arah_tendang;

        //array jarak dan sudut obstacle
        float array_jarak_obstacle[36];
        float array_sudut_obstacle[36];
};
```

omnivision.cpp

```
#include "omnivision.h"

omnivision::omnivision()
{
}

omnivision::~omnivision()
{
	save_setting();
	save_setting_obstacle();
}

void omnivision::setup(int _n, int _width, int _height)
{
	cap.open(_n);
	if (!cap.isOpened()) cerr << "ERROR! Unable to open camera\n";
	cap.set(CV_CAP_PROP_FRAME_WIDTH, _width);
	cap.set(CV_CAP_PROP_FRAME_HEIGHT, _height);
	width = _width;
	height = _height;
	center_x = _width / 2;
	center_y = _height / 2;

	open_setting();
	open_setting_obstacle();

	namedWindow("setting",0);
	createTrackbar("blur", "setting", &blur, min(_width, _height));
	createTrackbar("+ ball", "setting", &opening_ball, min(_width,
_height));
	createTrackbar("- ball", "setting", &closing_ball, min(_width,
_height));
	createTrackbar("+ field", "setting", &opening_field, min(_width,
_height));
	createTrackbar("- field", "setting", &closing_field, min(_width,
_height));

	namedWindow("ball",0);
	createTrackbar("H-", "ball", &hl_ball, 255);
	createTrackbar("H+", "ball", &hh_ball, 255);
	createTrackbar("S-", "ball", &sl_ball, 255);
	createTrackbar("S+", "ball", &sh_ball, 255);
	createTrackbar("V-", "ball", &vl_ball, 255);
	createTrackbar("V+", "ball", &vh_ball, 255);

	namedWindow("field",0);
	createTrackbar("H-", "field", &hl_field, 255);
	createTrackbar("H+", "field", &hh_field, 255);
	createTrackbar("S-", "field", &sl_field, 255);
	createTrackbar("S+", "field", &sh_field, 255);
	createTrackbar("V-", "field", &vl_field, 255);
	createTrackbar("V+", "field", &vh_field, 255);
```

```
		namedWindow("obstacle",0);
		createTrackbar("H-", "obstacle", &hl_obstacle, 255);
		createTrackbar("H+", "obstacle", &hh_obstacle, 255);
		createTrackbar("S-", "obstacle", &sl_obstacle, 255);
		createTrackbar("S+", "obstacle", &sh_obstacle, 255);
		createTrackbar("V-", "obstacle", &vl_obstacle, 255);
		createTrackbar("V+", "obstacle", &vh_obstacle, 255);


		namedWindow("line", 0);
		createTrackbar("x11", "line", &x1_line1, _width);
		createTrackbar("y11", "line", &y1_line1, _height);
		createTrackbar("x21", "line", &x2_line1, _width);
		createTrackbar("y21", "line", &y2_line1, _height);
		createTrackbar("t1", "line", &t1, _height);
		createTrackbar("x12", "line", &x1_line2, _width);
		createTrackbar("y12", "line", &y1_line2, _height);
		createTrackbar("x22", "line", &x2_line2, _width);
		createTrackbar("y22", "line", &y2_line2, _height);
		createTrackbar("t2", "line", &t2, _height);


		namedWindow("circle", 0);
		createTrackbar("x", "circle", &x_circle, _width);
		createTrackbar("y", "circle", &y_circle, _height);
		createTrackbar("r", "circle", &radius_circle, _height);


		namedWindow("circle2", 0);
		createTrackbar("x", "circle2", &x_circle2, _width);
		createTrackbar("y", "circle2", &y_circle2, _height);
		createTrackbar("r", "circle2", &radius_circle2, _height);
		createTrackbar("t", "circle2", &thickness_circle2, _height);


		return;
}

void omnivision::threadedFunction()
{
		while (isThreadRunning())
		{
			FPSCOUNT.startTimer();
			cap.read(frame);

			circle(frame, Point(x_circle, y_circle), radius_circle,
Scalar(255, 255, 255), -1);
			line(frame, Point(x1_line1, y1_line1), Point(x2_line1,
y2_line1), Scalar(255, 255, 255), t1);
			line(frame, Point(x1_line2, y1_line2), Point(x2_line2,
y2_line2), Scalar(255, 255, 255), t2);
			circle(frame, Point(x_circle2, y_circle2),
radius_circle2, Scalar(0, 0, 0), thickness_circle2);

			if (blur % 2 == 0) blur += 1;
```

```
                    GaussianBlur(frame, frame, Size(blur, blur), 0, 0);

                    cvtColor(frame, frameHSV, CV_BGR2HSV);

                    inRange(frameHSV, Scalar(hl_ball, sl_ball, vl_ball),
Scalar(hh_ball, sh_ball, vh_ball), ball);
                    if (opening_ball > 0)
                    {
                              erode(ball, ball,
getStructuringElement(MORPH_ELLIPSE, Size(opening_ball, opening_ball)));
                              dilate(ball, ball,
getStructuringElement(MORPH_ELLIPSE, Size(opening_ball, opening_ball)));
                    }
                    if (closing_ball > 0)
                    {
                              dilate(ball, ball,
getStructuringElement(MORPH_ELLIPSE, Size(closing_ball, closing_ball)));
                              erode(ball, ball,
getStructuringElement(MORPH_ELLIPSE, Size(closing_ball, closing_ball)));
                    }

                    inRange(frameHSV, Scalar(hl_field, sl_field, vl_field),
Scalar(hh_field, sh_field, vh_field), field);
                    if (opening_field > 0)
                    {
                              erode(field, field,
getStructuringElement(MORPH_ELLIPSE, Size(opening_field, opening_field)));
                              dilate(field, field,
getStructuringElement(MORPH_ELLIPSE, Size(opening_field, opening_field)));
                    }
                    if (closing_field > 0)
                    {
                              dilate(field, field,
getStructuringElement(MORPH_ELLIPSE, Size(closing_field, closing_field)));
                              erode(field, field,
getStructuringElement(MORPH_ELLIPSE, Size(closing_field, closing_field)));
                    }

                    int temp_ball_x, temp_ball_y;
                    find_ball(ball, field, frame, temp_ball_x, temp_ball_y);

                    if (temp_ball_x != -1 && temp_ball_y != -1)
                    {
                              lock();
                              ball_distance_pixel = sqrt(pow((temp_ball_x -
center_x), 2) + pow((temp_ball_y - center_y), 2));
                              ball_tetha = atan2((temp_ball_y - center_y), -
(temp_ball_x - center_x)) * 57.295779513;
                              ball_distance_real =
convert_distance2real(ball_distance_pixel);
                              unlock();

                              line(frame, Point(center_x, center_y),
Point(temp_ball_x, temp_ball_y), Scalar(0, 0, 255), 2);
                    }
```

```
                else
                {
                        lock();
                        ball_distance_pixel = -1000;
                        ball_tetha = -1000;
                        ball_distance_real = -1000;
                        unlock();
                }

                line(frame, Point(center_x, center_y), Point(0,
center_y), Scalar(0, 255, 255), 2);

                inRange(frameHSV, Scalar(hl_obstacle, sl_obstacle,
vl_obstacle), Scalar(hh_obstacle, sh_obstacle, vh_obstacle), obstacle);
                obstacle.convertTo(obstacle, CV_8UC1);
                float temp_object_distance[36], temp_object_tetha[36];
                int temp_jarak_terdekat, temp_sudut_terdekat;

                find_obstacle2(temp_object_distance,temp_object_tetha);

                for (int i = 0; i < 36; i++) temp_object_distance[i] =
convert_distance2real(temp_object_distance[i]);

                //find_closer_object(temp_object_distance,
temp_object_tetha, temp_jarak_terdekat, temp_sudut_terdekat);
                //find_closer_object2(temp_object_distance,
temp_object_tetha, temp_jarak_terdekat, temp_sudut_terdekat);

                lock();
                for (int i = 0; i < 36; i++)
                {
                        array_jarak_obstacle[i] =
temp_object_distance[i];
                        array_sudut_obstacle[i] = temp_object_tetha[i];

                }

                //for (int i = 0; i < 5; i++)
                //        cout << temp_object_distance[i] << " ";
                //cout << endl;

                //object_distance = temp_object_distance;
                //object_tetha = temp_object_tetha;
                //sudut_objek_terdekat = temp_sudut_terdekat;
                //jarak_objek_terdekat_pixel = temp_jarak_terdekat;
                //jarak_objek_terdekat_real =
convert_distance2real((float)jarak_objek_terdekat_pixel);
                arah_tendang = arah_tendang_penalty();
                unlock();

                if (FPSCOUNT.StartTimeCounter(500)) {
                        sprintf(FPS, "%3.2f", tempFPS);
                        FPSCOUNT.ResetTimeCounter();
                }
```

```
                    putText(frame, FPS, Point(10, 25), font, 1, Scalar(160,
160, 160), 2, 8, false);

                    if (frame.empty()) cerr << "ERROR! blank frame
grabbed\n";
                    else imshow("frame", frame);
                    if (ball.empty()) cerr << "ERROR! blank frame grabbed\n";
                    else imshow("ball treshold", ball);
                    if (field.empty()) cerr << "ERROR! blank frame
grabbed\n";
                    else imshow("field treshold", field);
                    if (obstacle.empty()) cerr << "ERROR! blank frame
grabbed\n";
                    else imshow("obstacle treshold", obstacle);
                    waitKey(1);
                    tempFPS = 1/FPSCOUNT.stopTimer();
          }
}

void omnivision::find_closer_object(vector<int> &_object_distance,
vector<int> &_object_tetha, int& _jarak_objek_terdekat, int&
_sudut_objek_terdekat)
{
          //find closer obstacle
          auto n = distance(_object_distance.begin(),
min_element(_object_distance.begin(), _object_distance.end()));

          if (n > 0)
          {
                    _jarak_objek_terdekat = _object_distance[n];
                    _sudut_objek_terdekat = _object_tetha[n];
                    line(frame, Point(320 + (_jarak_objek_terdekat)*
cos(_sudut_objek_terdekat*CV_PI / 180), obstacle.rows / 2 +
(_jarak_objek_terdekat)* sin(_sudut_objek_terdekat*CV_PI / 180)),
Point(obstacle.cols / 2 + radius_circle * cos(_sudut_objek_terdekat * CV_PI
/ 180), obstacle.rows / 2 + radius_circle * sin(_sudut_objek_terdekat *
CV_PI / 180)), Scalar(255, 255, 0), 2);

                    // disesuaikan dengan sudut kameranya untuk bola
                    _sudut_objek_terdekat = 180 - _sudut_objek_terdekat;
                    if (_sudut_objek_terdekat < 0)_sudut_objek_terdekat +=
360;
          }

          else
          {
                    _jarak_objek_terdekat = -1000;
                    _sudut_objek_terdekat = -1000;
          }
}

void omnivision::find_closer_object2(float* _object_distance, float*
_object_tetha, float threshold)
{
```

```
	//treshold

}

void omnivision::find_obstacle2(float* _object_distance, float*
_object_tetha)
{
	for (int tetha = 0; tetha < 360; tetha += 10)
	{
		object = false;
		int temp = 0;
		for (int i = radius_circle; i < radius_circle2 -
(thickness_circle2 / 2); i++)
		{
			int x = obstacle.cols / 2 - i * cos(tetha *
CV_PI / 180);
			int y = obstacle.rows / 2 + i * sin(tetha *
CV_PI / 180);
			if (x < obstacle.cols && x > 0 && y <
obstacle.rows && y > 0)
			{
				if (obstacle.at<unsigned char>(y, x)
== 255)
				{
					object = true;
					cv::circle(frame, Point(x,
y), 5, Scalar(255, 0, 0));
					_object_distance[tetha / 10]
= i;
					_object_tetha[tetha / 10] =
tetha;
					break;
				}
				temp = i;
				cv::line(frame, Point(x, y),
Point(obstacle.cols / 2 - radius_circle * cos(tetha * CV_PI / 180),
obstacle.rows / 2 + radius_circle * sin(tetha * CV_PI / 180)), Scalar(255,
0, 255));
			}
		}

		if (!object)
		{
			_object_distance[tetha / 10] = 500;
			_object_tetha[tetha / 10] = tetha;
		}
	}
}

void omnivision::find_ball(Mat _ball, Mat _field, Mat &_frame, int &_x, int
&_y)
{
	vector<vector<Point>> contours;
	vector<Vec4i> hierarchy;
```

```
        findContours(_ball, contours, hierarchy, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
        if (contours.size() < 1)
        {
                _x = -1;
                _y = -1;
                return;
        }
        vector<Point2f> center(contours.size());
        vector<float> radius(contours.size());
        int check_x_positive, check_x_negative, check_y_positive,
check_y_negative;
        Scalar color = Scalar(255, 0, 0);
        for (int i = 0; i < contours.size(); i++)
        {
                //find minimum enclosing circle for all contours
                minEnclosingCircle((Mat)contours[i], center[i],
radius[i]);

                check_x_positive = center[i].x + (radius[i] + 5);
                check_x_negative = center[i].x - (radius[i] + 5);
                check_y_positive = center[i].y + (radius[i] + 5);
                check_y_negative = center[i].y - (radius[i] + 5);

                if (check_x_positive >= width) check_x_positive = 0;
                if (check_x_negative < 0) check_x_negative = 0;
                if (check_y_positive >= height) check_y_positive =
center[i].y;
                if (check_y_negative < 0) check_y_negative = center[i].y;

                if (field.at<uchar>(check_y_positive, center[i].x) == 0
&&
                        field.at<uchar>(check_y_negative, center[i].x)
== 0 &&
                        field.at<uchar>(center[i].y, check_x_positive)
== 0 &&
                        field.at<uchar>(center[i].y, check_x_negative)
== 0) radius[i] = 0;
        }
        //find largest radius from all minimum enclosing circle
        auto max_radius = distance(radius.begin(),
max_element(radius.begin(), radius.end()));

        cv::circle(_frame, center[max_radius], (int)radius[max_radius],
color, 2);

        _x = center[max_radius].x;
        _y = center[max_radius].y;
        return;
}

void omnivision::find_obstacle(vector<int> &_object_distance, vector<int>
&_object_tetha)
{
        /*_object_distance.clear();
```

```
        _object_tetha.clear();*/
        _object_distance.erase(_object_distance.begin(),
_object_distance.end());
        _object_tetha.erase(_object_tetha.begin(), _object_tetha.end());
        vector<int> object_xy(2, 0);
        vector<vector <int>> object_coord;
        for (int tetha = 0; tetha < 360; tetha += 10)
        {
                object = false;
                int temp = 0;
                for (int i = radius_circle; i < radius_circle2 -
(thickness_circle2 / 2); i++)
                {
                        int x = obstacle.cols / 2 + i * cos(tetha *
CV_PI / 180);
                        int y = obstacle.rows / 2 + i * sin(tetha *
CV_PI / 180);
                        if (x < obstacle.cols && x > 0 && y <
obstacle.rows && y > 0)
                        {
                                //cout <<
(int)frame_result.at<uchar>(y, x) << endl;
                                if (obstacle.at<unsigned char>(y, x)
== 255)
                                {
                                        object = true;
                                        circle(frame, Point(x, y), 5,
Scalar(255, 0, 0));

        _object_distance.push_back(i);

        _object_tetha.push_back(tetha);

                                        object_xy[0] = y;
                                        object_xy[1] = x;

        object_coord.push_back(object_xy);

                                switch(tetha){
                                        case (160):
                                                garis[0] = i;
                                        break;
                                        case (170):
                                                garis[1] = i;
                                        break;
                                        case (190):
                                                garis[2] = i;
                                        break;
                                        case (200):
                                                garis[3] = i;
                                        break;
                                }

                                        break;
                                }
```

```
                                        temp = i;
                                        line(frame, Point(x, y),
Point(obstacle.cols / 2 + radius_circle * cos(tetha * CV_PI / 180),
obstacle.rows / 2 + radius_circle * sin(tetha * CV_PI / 180)), Scalar(255,
0, 255));
                                        //line(frame, Point(x, y), Point(3,3),
Scalar(255, 0, 255));
                                }
                        }

                        if(!object){
                                switch(tetha){
                                        case (160):
                                                garis[0] = radius_circle2 -
(thickness_circle2 / 2);
                                        break;
                                        case (170):
                                                garis[1] = radius_circle2 -
(thickness_circle2 / 2);
                                        break;
                                        case (190):
                                                garis[2] = radius_circle2 -
(thickness_circle2 / 2);
                                        break;
                                        case (200):
                                                garis[3] = radius_circle2 -
(thickness_circle2 / 2);
                                        break;
                                        }
                        }
                        /*int x1 = frame_result.cols / 2 + radius_circle *
cos(tetha * CV_PI / 180);
                        int y1 = frame_result.cols / 2 + radius_circle *
sin(tetha * CV_PI / 180);
                        int x2 = frame_result.cols / 2 + temp * cos(tetha * CV_PI
/ 180);
                        int y2 = frame_result.cols / 2 + temp * sin(tetha * CV_PI
/ 180);
                        line(frame, Point(x1, y1), Point(x2, y2), Scalar(0, 0,
255));*/
                }

                //auto n = distance(object_distance.begin(),
min_element(object_distance.begin(), object_distance.end()));
                //cout << object_distance[n] << " " << object_tetha[n] << endl;
                //circle(frame, Point(object_coord[n][1], object_coord[n][0]), 15,
Scalar(0, 255, 0), 2);
}

void omnivision::threshold_object_distance()
{

}

bool omnivision::arah_tendang_penalty(){ // return 0 kalo tendang kiri
```

```
        if (garis[0] + garis[1] > garis[2] + garis[3]) return false;
        else return true;
}

void omnivision::open_setting()
{
        ifstream input_file;
        input_file.open("setting_omnivison.ini");
        if (input_file.is_open())
        {
                input_file >> blur;
                input_file >> opening_ball;
                input_file >> closing_ball;
                input_file >> opening_field;
                input_file >> closing_field;
                input_file >> hl_ball;
                input_file >> hh_ball;
                input_file >> sl_ball;
                input_file >> sh_ball;
                input_file >> vl_ball;
                input_file >> vh_ball;
                input_file >> hl_field;
                input_file >> hh_field;
                input_file >> sl_field;
                input_file >> sh_field;
                input_file >> vl_field;
                input_file >> vh_field;
                input_file.close();
        }
        else cout << "gagal membuka file setting_omnivison.ini" << endl;
        return;
}

void omnivision::open_setting_obstacle()
{
        ifstream input_file;
        input_file.open("setting_obstacle.ini");
        if (input_file.is_open())
        {
                input_file >> hl_obstacle;
                input_file >> hh_obstacle;
                input_file >> sl_obstacle;
                input_file >> sh_obstacle;
                input_file >> vl_obstacle;
                input_file >> vh_obstacle;
                input_file >> x_circle;
                input_file >> y_circle;
                input_file >> radius_circle;
                input_file >> x_circle2;
                input_file >> y_circle2;
                input_file >> radius_circle2;
                input_file >> thickness_circle2;
                input_file >> x1_line1;
                input_file >> y1_line1;
                input_file >> x2_line1;
```

```
                input_file >> y2_line1;
                input_file >> t1;
                input_file >> x1_line2;
                input_file >> y1_line2;
                input_file >> x2_line2;
                input_file >> y2_line2;
                input_file >> t2;
                input_file.close();
        }
        else cout << "gagal membuka file setting_obstacle.ini" << endl;
        return;
}

void omnivision::save_setting()
{
        ofstream output_file;
        output_file.open("setting_omnivison.ini");
        if (output_file.is_open())
        {
                output_file << blur << endl;
                output_file << opening_ball << endl;
                output_file << closing_ball << endl;
                output_file << opening_field << endl;
                output_file << closing_field << endl;
                output_file << hl_ball << endl;
                output_file << hh_ball << endl;
                output_file << sl_ball << endl;
                output_file << sh_ball << endl;
                output_file << vl_ball << endl;
                output_file << vh_ball << endl;
                output_file << hl_field << endl;
                output_file << hh_field << endl;
                output_file << sl_field << endl;
                output_file << sh_field << endl;
                output_file << vl_field << endl;
                output_file << vh_field << endl;
                output_file.close();
        }
        else cout << "gagal mengupdate file setting_omnivison.ini" <<
endl;
        return;
}

void omnivision::save_setting_obstacle()
{
        ofstream output_file;
        output_file.open("setting_obstacle.ini");
        if (output_file.is_open())
        {
                output_file << hl_obstacle << endl;
                output_file << hh_obstacle << endl;
                output_file << sl_obstacle << endl;
                output_file << sh_obstacle << endl;
                output_file << vl_obstacle << endl;
                output_file << vh_obstacle << endl;
```

```
                output_file << x_circle << endl;
                output_file << y_circle << endl;
                output_file << radius_circle << endl;
                output_file << x_circle2 << endl;
                output_file << y_circle2 << endl;
                output_file << radius_circle2 << endl;
                output_file << thickness_circle2 << endl;
                output_file << x1_line1 << endl;
                output_file << y1_line1 << endl;
                output_file << x2_line1 << endl;
                output_file << y2_line1 << endl;
                output_file << t1 << endl;
                output_file << x1_line2 << endl;
                output_file << y1_line2 << endl;
                output_file << x2_line2 << endl;
                output_file << y2_line2 << endl;
                output_file << t2 << endl;
                output_file.close();
        }
        else cout << "gagal mengupdate file setting_obstacle.ini" << endl;
        return;
}

float omnivision::convert_distance2real(float _distance_pixel)
{
        arma::Mat<float> w1, w2, b1, b2, layer1, layer2;

        w1 << -0.5390 << arma::endr
                << 0.1234 << arma::endr
                << 4.1846 << arma::endr
                << 0.1496 << arma::endr
                << 0.2232 << arma::endr
                << -3.6618 << arma::endr
                << -2.3694 << arma::endr
                << -0.9918 << arma::endr
                << -0.3709 << arma::endr
                << -13.1333 << arma::endr;

        b1 << -1.0291 << arma::endr
                << -1.8058 << arma::endr
                << -3.8613 << arma::endr
                << -1.7830 << arma::endr
                << -1.8824 << arma::endr
                << 1.5590 << arma::endr
                << 0.2587 << arma::endr
                << -0.9881 << arma::endr
                << -1.4771 << arma::endr
                << 14.3722 << arma::endr;

        w2 << 0.1919 << 0.5468 << 1.2799 << 0.5196 << 0.6143 << 0.2477 <<
-0.4992 << -0.2190 << 0.1106 << -4.3261;

        b2 << 3.3527;

        float normalizedN = normalisasi(_distance_pixel, 0, 231.00);
```

```
        layer1 = (w1 * normalizedN) + b1;
        layer1 = 1 / (1 + exp(-layer1));
        layer2 = (w2 * layer1) + b2;
        float out = layer2.at(0.0);
        return denormalisasi(out, 0, 500);
}

float omnivision::normalisasi(float in, float inMin, float inMax) {
        float out = 0;
        out = (2.0*(in - inMin) / (inMax - inMin)) - 1;
        return out;
}

float omnivision::denormalisasi(float in, float inMin, float inMax) {
        float out = 0;
        out = 0.5*(in + 1) * (inMax - inMin) + inMin;
        return out;
}
```

Halaman ini sengaja dikosongkan

BIODATA PENULIS

Yohan Prakoso lahir di lamongan 9 Desember 1994 merupakan anak pertama dari 2 bersaudara dari pasangan Yoyok Joko Setiono dan Nur Kholifah. Penulis menghabiskan banyak kehidupan di tanah Madura tepatnya di kota Sumenep sejak berumur 2 tahun. Penulis menyelesaikan pendidikan dasar di SDN Pajagalan II Sumenep pada tahun 2007 dialnjutkan dengan pendidikan menengah di SMPN1 Sumenep pada 2010 dan SMAN 3 Pamekasan pada tahun 2013. Penulis memulai pendidikan di Institut Teknologi Sepuluh Nopember pada tahun 2013. Selama mengenyam pendidikan disana penulis aktif mengikuti kegiatan robotika ITS dan sempat menjadi tim inti pada ajang KKCTBN 2014, KRPAI 2015, DECONBOTION 2015, KRPAI 2016, AUVSI ROBOBOAT CONTEST 2016, dan KRSBI Beroda 2017. Selain itu penulis juga aktif menjadi asistem di lab Elektronika Dasar.

Email:
prakoso.yohan@gmail.com