



TUGAS AKHIR - MN 184802

**PERANCANGAN OBJECT DETECTION SYSTEM UNTUK
COLLISION AVOIDANCE SYSTEM KAPAL DENGAN
PEMANFAATAN KAMERA DAN LIGHT DETECTION AND
RANGING (LiDAR)**

**Anthony Suryajaya
NRP 0411184000035**

**Dosen Pembimbing
Totok Yulianto. S.T., M.T.
Dr. Eng. Yuda Apri Hermawan, S.T.,M.T**

**DEPARTEMEN TEKNIK PERKAPALAN
FAKULTAS TEKNOLOGI KELAUTAN
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2022**



TUGAS AKHIR - MN 184802

**PERANCANGAN *OBJECT DETECTION SYSTEM* UNTUK
COLLISION AVOIDANCE SYSTEM KAPAL DENGAN
PEMANFAATAN KAMERA DAN *LIGHT DETECTION AND
RANGING (LiDAR)***

**Anthony Suryajaya
NRP 0411184000035**

**Dosen Pembimbing
Totok Yulianto, S.T., M.T.
Dr. Eng. Yuda Apri Hermawan, S.T.,M.T.**

**DEPARTEMEN TEKNIK PERKAPALAN
FAKULTAS TEKNOLOGI KELAUTAN
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2022**



FINAL PROJECT - MN 184802

**DESIGNING AN OBJECT DETECTION SYSTEM FOR
SHIP'S COLLISION AVOIDANCE SYSTEM USING CAMERA
AND LIGHT DETECTION AND RANGING (LiDAR)**

**Anthony Suryajaya
NRP 04111840000035**

**Supervisor
Totok Yulianto, S.T., M.T.
Dr. Eng. Yuda Apri Hermawan, S.T.,M.T.**

**DEPARTMENT OF NAVAL ARCHITECTURE
FACULTY OF MARINE TECHNOLOGY
SEPULUH NOPEMBER INSTITUTE OF TECHNOLOGY
SURABAYA
2022**

LEMBAR PENGESAHAN

PERANCANGAN OBJECT DETECTION SYSTEM UNTUK COLLISION AVOIDANCE SYSTEM KAPAL DENGAN PEMANFAATAN KAMERA DAN LIGHT DETECTION AND RANGING (LiDAR)

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Teknik
pada
Program Sarjana Departemen Teknik Perkapalan
Fakultas Teknologi Kelautan
Institut Teknologi Sepuluh Nopember

Oleh:

ANTHONY SURYAJAYA
NRP 04111840000035

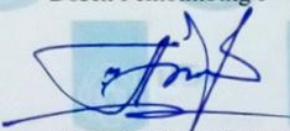
Disetujui oleh:

Dosen Pembimbing II



Dr. Eng. Yuda Apri Hermawan, S.T., M.T.
NIP 1989202011007

Dosen Pembimbing I



Totok Yulianto, S.T., M.T.
NIP 19700731 199512 1 001

Mengetahui,

Kepala Departemen Teknik Perkapalan



Jr. Wasis Dwi Aryawan, M.Sc., Ph.D.
NIP 19640210 198903 1 001

SURABAYA, 27 Juli 2022

LEMBAR REVISI

PERANCANGAN OBJECT DETECTION SYSTEM UNTUK COLLISION AVOIDANCE SYSTEM KAPAL DENGAN PEMANFAATAN KAMERA DAN LIGHT DETECTION AND RANGING (LiDAR)

TUGAS AKHIR

Telah direvisi sesuai dengan hasil Ujian Tugas Akhir
Tanggal 13 Juli 2022

Program Sarjana Departemen Teknik Perkapalan
Fakultas Teknologi Kelautan
Institut Teknologi Sepuluh Nopember

Oleh:


ANTHONY SURYAJAYA
NRP 04111840000035

Disetujui oleh Tim Penguji Ujian Tugas Akhir:

1. Prof. Ir. R Sjarief Widjaja, Ph.D 
2. Dedi Budi Purwanto S.T., M.T. 
3. Danu Utama, S.T., M.T. 

Disetujui oleh Dosen Pembimbing Tugas Akhir:

1. Totok Yulianto, S.T., M.T. 
2. Dr. Eng. Yuda Apri Hermawan, S.T., M.T. 

SURABAYA, 27 Juli 2022

HALAMAN PERUNTUKAN

Dipersembahkan kepada kedua orang tua atas segala dukungan dan doanya

KATA PENGANTAR

Puji syukur kepada Tuhan Yang Maha Esa karena atas karunianya Tugas Akhir ini dapat diselesaikan dengan baik.

Pada kesempatan ini Penulis ingin mengucapkan terima kasih kepada pihak-pihak yang membantu penyelesaian Tugas Akhir ini, yaitu:

1. Orang tua dan keluarga Penulis atas doa dan dukungannya selama penulisan Tugas Akhir
2. Totok Yulianto, S.T., M.T. dan Dr. Eng. Yuda Apri Hermawan, S.T.,M.T selaku Dosen Pembimbing atas bimbingan, kritikan dan saran selama penggerjaan dan penyusunan Tugas Akhir ini;
3. Bapak Dedi Budi Purwanto S.T., M.T. selaku Dosen Pengaji yang telah memberikan kritik dan sarannya untuk perbaikan Laporan Tugas Akhir ini;
4. Ir. Hesty Anita Kurniawati, M.Sc. selaku Dosen Wali selama masa perkuliahan di Departemen Teknik Perkapalan FTK ITS;
5. Teman-teman seperjuangan P58 yang telah menemani Penulis sejak awal selama masa perkuliahan di Departemen Teknik Perkapalan FTK ITS;
6. Christian Andrew dan Shintya Rezqi yang telah menjadi mentor Penulis dalam pengembangan program Tugas Akhir ini; ← tambah kadep, dosen pengaji
7. Andhika Asmara dan Ahmad Lazuardi Iman yang telah menjadikan mentor Penulis dalam Penggerjaan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih jauh dari kesempurnaan, sehingga kritik dan saran yang bersifat membangun sangat diharapkan. Akhir kata semoga laporan ini dapat bermanfaat bagi banyak pihak.

Surabaya,2022

Anthony Suryajaya

PERANCANGAN OBJECT DETECTION SYSTEM UNTUK COLLISION AVOIDANCE SYSTEM KAPAL DENGAN PEMANFAATAN KAMERA DAN LIGHT DETECTION AND RANGING (LiDAR)

Nama Mahasiswa : Anthony Suryajaya
NRP : 04111840000035
Departemen / Fakultas : Teknik Perkapalan / Teknologi Kelautan
Dosen Pembimbing : 1. Totok Yulianto, S.T., M.T.
2. Dr. Eng. Yuda Apri Hermawan, S.T.,M.T

ABSTRAK

Pada era revolusi industri 4.0 ini, perkembangan teknologi menjadi semakin pesat, khususnya pada bagian komputasi *Artifical Inteligence* (AI) dan *Machine Learning* (ML) dan pada saat penulisan laporan ini sudah banyak yang mengaplikasikan teknologi ini di industri transportasi baik itu untuk mobil, pesawat dan bahkan kapal. Dalam dunia pelayaran, kejadian tubrukan antar kapal itu sering terjadi karena kelalaian manusia, sehingga diperlukannya sistem bantuan mengemudi untuk meminimalisir kejadian ini sebelum terjadi. Oleh karena itu, penulis mengaplikasikan teknologi *object detection*, LiDAR, dan microprocessor untuk sistem pencegahan tubrukan yang sederhan. Untuk membuat sistem ini akan dikembangkan program yang bisa memadukan *object detection* dan pengukuran jarak melalui LiDAR yang akan diuji dengan menggunakan model kapal yang sudah ada dan untuk parameter jarak aman penghindaran akan menggunakan metode *blocking area*. Dengan hasil, setelah pengujian 60 kali dengan variasi kecepatan dengan *froude numbe* (Fn) dari 0.36 hingga 0.72 sistem berhasil 100% menghindari tubrukan dengan sudut *rudder* paling optimal 10 derajat untuk skenario *head-on collision* dan setelah membanding kan 60 data terhadap sudut *heading* kapal maka didapat rentan sudut 8.9-14.6 derajat untuk *heading* kapalnya. ← kurang detail meotde yang digunakan, tambahkan metode *object detection*nya

Kata kunci: *Artificial Inteligence* (AI), *Machine Learning* (ML), *Object Detection* (OD), *microprocessor*, LiDAR, Kamera

DESIGNING AN OBJECT DETECTION SYSTEM FOR SHIP'S COLLISION AVOIDANCE SYSTEM USING CAMERA AND LIGHT DETECTION AND RANGING (LiDAR)

Author : Anthony Suryajaya
Student Number : 04111840000035
Department / Faculty : Naval Architecture / Marine Technology
Supervisor :
1. Totok Yulianto, S.T., M.T.
2. Dr. Eng. Yuda Apri Hermawan, S.T.,M.T.

ABSTRACT

In this era of industrial revolution 4.0, the advancement of technology is at an astonishing pace especially in the area of Artificial Intelligence (AI) and Machine Learning (ML) computation. And at the time of writing this report, there are already many applications of these amazing technologies in the world of transportation be it for cars, planes or even ships. In the world of sailing, collisions happen most of the time because of the negligence of the driver. Therefore, the writer wishes to implement Object Detection, LiDAR and microprocessor to make an anti collision system. To make the system, first is to develop a program that can both detect objects and read data from the LiDAR sensor then tested on an already built ship model, which the safety range for avoidance parameter is calculated by the blocking area method developed by (Kijima & Furukawa, 2003). After completing 60 tests with a variation on speed, which is then converted into froudes number between 0.36 to 0.72, the results indicates that after testing the system 60 times it has successfully avoided collision 100% of the time with the optimal rudder angle of 10 degrees for the head-on collision scenario and after comparing the 60 data points to the ship's heading angle, it is known that the ship has an average heading angle of 8.9-14.6 degrees.

Keywords: *Artificial Intelligence (AI), Machine Learning (ML), Object Detection (OD), microprocessor, LiDAR, Camera.*

DAFTAR ISI

LEMBAR PENGESAHAN	v
LEMBAR REVISI.....	ix
HALAMAN PERUNTUKAN.....	xi
KATA PENGANTAR.....	xiii
ABSTRAK.....	xv
ABSTRACT	xvii
DAFTAR ISI	xix
DAFTAR GAMBAR.....	xxi
DAFTAR TABEL	1
BAB 1 PENDAHULUAN	1
1.1. Latar Belakang Masalah	1
1.2. Perumusan Masalah	2
1.3. Tujuan	2
1.4. Batasan Masalah	2
1.5. Manfaat	3
1.6. Hipotesis	3
BAB 2 STUDI LITERATUR	5
2.1. Dasar Teori	5
2.1.1. <i>Machine Learning</i>	5
2.1.2. Dasar Matematis <i>Machine Learning</i>	6
2.1.3. Model <i>Machine Learning</i> untuk <i>Object Detection</i>	9
2.1.4. <i>MobileNet</i>	13
2.1.5. Aplikasi <i>Machine Learning</i> dalam masalah <i>Computer Vision</i>	17
2.1.6. Sensor Kamera	23
2.1.7. Sensor LiDAR.....	25
2.2. Tinjauan Pustaka.....	27
2.2.1. <i>Automatic Collision Avoidance System</i> dengan Konsep <i>Blocking Area</i>	27
2.2.2. <i>Collision Avoidance System for Autonomous Vehicles</i>	32
2.2.3. <i>LiDAR and Camera Detection Fusion in a Real-Time Industrial Multi-Sensor Collision Avoidance System</i>	32
2.2.4. <i>COLREG-Compliant Collision Avoidance for Unmanned Surface Vehicle using Deep Reinforcement Learning</i>	34
BAB 3 METODOLOGI	37
3.1. Bagan Alir.....	37
3.2. Analisis Permasalahan	38
3.3. Studi Literatur	38
3.4. Pembuatan Program.....	38
3.5. Perakitan Sistem Elektronik	38
3.6. Perakitan Sistem	38
3.7. Trial Darat (<i>System Function Test</i>).....	39
3.8. Trial Air	39
3.9. Kesimpulan dan Saran	39
BAB 4 PEMBAHASAN DAN ANALISIS.....	41

4.1. Perencanaan Sistem <i>Collision Avoidance</i>	41
4.2. Pemilihan Model <i>Machine Learning</i>	41
4.3. Perakitan Sistem Elektronik.....	42
4.4. Pengujian <i>Code</i>	Error! Bookmark not defined.
4.5. Pembuatan <i>Code</i> Sensor dan <i>Actuator</i>	45
4.5.1. Penjelasan dan Pengujian <i>Code</i> Kamera dan LiDAR	45
4.6. Trial Darat	56
4.6.1. Pengujian LiDAR.....	57
4.6.2. Pengujian Servo.....	59
4.7. Trial Air.....	62
4.7.1. Variasi Fn 0.36 dengan Parameter Jarak Penuh	64
4.7.2. Variasi Fn 0.36 dengan 75% Parameter Jarak.....	67
4.7.3. Variasi Fn 0.58 dengan Parameter Jarak Penuh	69
4.7.4. Variasi Fn 0.58 dengan 75% Parameter Jarak.....	71
4.7.5. Variasi Fn 0.72 dengan Parameter Jarak Penuh	73
4.7.6. Variasi Fn 0.72 dengan 75% Parameter Jarak.....	75
4.7.7. Evaluasi Kinerja Sistem	77
BAB 5 KESIMPULAN DAN SARAN	79
5.1. Kesimpulan	79
5.2. Saran.....	80
DAFTAR PUSTAKA	81
LAMPIRAN	
LAMPIRAN A <i>Code Collision Avoidance</i>	
BIODATA PENULIS	

DAFTAR GAMBAR

Gambar 2.1 Perbedaan AI,ML, dan DL	5
Gambar 2.2 Perbedaan Cara Pemrograman Konvensional VS Pemrograman ML	5
Gambar 2.3 Contoh Distribusi Probabilitas <i>Multivariate</i>	8
Gambar 2.4 Cara Kerja R-CNN Secara Umum.....	11
Gambar 2.5 Cara Kerja <i>Fast R-CNN</i> Secara Umum.....	11
Gambar 2.6 Cara Kerja <i>Faster R-CNN</i> Secara Umum	12
Gambar 2.7 Cara Kerja Model YOLO Secara Umum.....	13
Gambar 2.8 Memisahkan <i>Kernel</i> Spasial 3x3	14
Gambar 2.9 Cara Kerja <i>Depthwise Convolutions</i> pada <i>MobileNetV1</i>	15
Gambar 2.10 Perbedaan <i>Convolution Standard</i> dengan <i>Deepwise Separable Convolution</i> dengan BN dan ReLU.....	16
Gambar 2.11 Ukuran Model <i>MobileNet</i> dan <i>Conv MobileNet</i>	17
Gambar 2.12 Contoh Ringkasan dari Arsitektur <i>Covnet</i>	17
Gambar 2.13 Data Gambar yang Dipecah menjadi Pola Unik	18
Gambar 2.14 Representasi dari Hirarki Spasial dari Sebuah Konsep Visual sebuah Kucing ..	19
Gambar 2.15 Contoh dari <i>Response Map</i>	20
Gambar 2.16 Visualisasi Lokasi dari Tambal 3x3 pada Input Feature Map Berukuran 5x5 ...	20
Gambar 2.17 Cara Kerja <i>Convolution</i>	21
Gambar 2.18 Guna <i>Padding</i> pada <i>Input</i> 5x5 untuk Ekstrasi 25 Tambal Berukuran 3x3.....	21
Gambar 2.19 Tambal <i>Convolution</i> dengan ukuran 3x3 dan dengan <i>stride</i> 2x2	22
Gambar 2.20 Cara Kerja Kamera Analog/Film (a) dan cara Kerja Kamera Digital (b).....	23
Gambar 2.21 Diagram <i>Illuminance</i>	24
Gambar 2.22 Raspberry Pi Camera Module V2	24
Gambar 2.23 Sensor LiDAR TF-Luna	25
Gambar 2.24 Visualisasi Pengukuran Sensor LiDAR	26
Gambar 2.25 <i>Blocking Area</i>	30
Gambar 2.26 Perluasan <i>Blocking Area</i>	31
Gambar 2.27 (a) dan (b) Sajian Grafis Data dari LiDAR yang Digunakan pada Percobaan ..	32
Gambar 2.28 Block Diagram <i>Fusion System</i> yang dirancang	33
Gambar 2.29 Pemakaian <i>Passive Beacon</i> sebagai Penanda Daerah yang Dilarang Masuk ..	33
Gambar 2.30 Penjelasan Proses <i>Fusion System</i> yang Dirancang	34
Gambar 2.31 Skenario Tabrakan Kapal yang Telah Didefinisikan pada COLREGs.....	35
Gambar 3.1 Bagan Alir Pengerjaan Tugas Akhir.....	37
Gambar 4.1 <i>Flowchart</i> Sistem <i>Collision Avoidance</i>	41
Gambar 4.2 Diagram Fungsi dari <i>GPIO Port Raspberry Pi</i>	42
Gambar 4.3 <i>Wiring Diagram</i> Sensor dan <i>Actuator</i> pada <i>Raspberry Pi</i>	43
Gambar 4.4 Model Kapal yang Telah Dimodifikasi	44
Gambar 4.5 Tampak atas <i>Superstructure</i> Model Kapal	44
Gambar 4.6 Perintah <i>Python</i> untuk <i>import</i> dalam Sistem Program	46
Gambar 4.7 <i>Python Class</i> untuk LiDAR	48
Gambar 4.8 Pemanggilan <i>Object</i> pada Sistem <i>Autonomous</i>	49
Gambar 4.9 <i>Object Class VideoStream</i>	49
Gambar 4.10 Program menerima Argumen Tambahan sebelum Dieksekusi	50

Gambar 4.11 Cara <i>Setup Framework TensorFlow Lite untuk Membaca Model ML</i>	51
Gambar 4.12 Cara <i>Import Model ML yang Sudah di Pre-Trained</i>	51
Gambar 4.13 Perintah untuk Menyalakan Kamera dalam Program Sistem <i>Object Detection</i>	51
Gambar 4.14 Perbedaan Proses <i>Concurrency</i> dan <i>Parallel</i>	52
Gambar 4.15 Fungsi Jeda Pembacaan LiDAR	52
Gambar 4.16 Perintah untuk Membuat GUI Sederhana	53
Gambar 4.17 <i>Graphical User Interface</i> dari <i>Object Detector</i>	53
Gambar 4.18 (a)Proses <i>Looping</i> untuk Menampilkan Hasil Prediksi Model ML (b) Perintah untuk Menggambar <i>Boundix Box</i> dari Objek Prediksi	54
Gambar 4.19 (a) Fungsi untuk Membaca <i>Serial Data</i> LiDAR dan (b) Untuk Menampilkan Data tersebut secara Terus Menerus	54
Gambar 4.20 Perhitungan <i>Blocking Area</i> untuk Kapal Nyata	55
Gambar 4.21 Perhitungan <i>Blocking Area</i> untuk Model.....	56
Gambar 4.22 Sistem Kondisi <i>Collision Avoidance</i>	56
Gambar 4.23 GUI Pengukuran dengan LiDAR pada <i>Raspberry Pi</i>	57
Gambar 4.24 Uji Pengukuran Sensor LiDAR dengan Meteran.....	58
Gambar 4.25 Akurasi Pembacaan LiDAR.....	58
Gambar 4.26 Pengujian Konfigurasi <i>Rudder</i>	60
Gambar 4.27 Perubahan <i>Rudder</i> ke Arah <i>Ports</i>	60
Gambar 4.28 Perubahan Sudut <i>Rudder</i> Sebanyak 1 Derajat	61
Gambar 4.29 Perubahan <i>Rudder</i> ke Arah <i>Starboard</i>	62
Gambar 4.30 (a) Kolam Uji Kecepatan (b) Ilustrasi Ukuran Kolam Uji	63
Gambar 4.31 (a) Grafik Sudut <i>Rudder</i> dari Gerakan Servo. (b) Grafik Kecepatan Kapal Terhadap Perubahan CDC.	64
Gambar 4.32 Bentuk Lintasan Variasi Fn 0.36- Jarak Parameter Penuh.....	65
Gambar 4.33 Bentuk Lintasan Fn 0.36-Jarak Parameter	67
Gambar 4.34 Bentuk Lintasan Fn 0.58-Jarak Parameter Penuh	69
Gambar 4.35 Bentuk Lintasan Fn 0.58-75% Jarak Parameter	71
Gambar 4.36 Bentuk Lintasan Fn 0.72-Jarak Parameter Penuh	73
Gambar 4.37 Bentuk Lintasan Fn 0.72-75% Jarak Parameter	75
Gambar 4.38 Plot Success Rate System Collision Avoidance	77
Gambar 4.39 <i>Plotting Trendline</i> tiap Variasi Kecepatan dan Jarak.....	78

DAFTAR TABEL

Tabel 2-1 Contoh Penyajian Data Tabular	7
Tabel 2-2 Arsitektur <i>Layer</i> dari <i>MobileNet</i>	15
Tabel 4-1 <i>Main Dimension</i> Kapal Contoh.....	55
Tabel 4-2 Pemeriksaan Akurasi LiDAR.....	59
Tabel 4-3 Konfigurasi <i>Servo</i>	61
Tabel 4-4 Data Kecepatan Kapal dengan Nilai CDC	63
Tabel 4-5 Sudut <i>Heading</i> Kapal untuk Fn 0.36 dengan Jarak Parameter Penuh.....	65
Tabel 4-6 Hasil Data Koordinat dari Video <i>Trial Air</i> Fn 0.36-Jarak Parameter Penuh	66
Tabel 4-7 Hasil Data Koordinat dari Video <i>Trial Air</i> Fn 0.36-Jarak Parameter 75%	66
Tabel 4-8 Sudut Heading Kapal untuk Fn 0.36 dengan 75% Jarak Parameter	68
Tabel 4-9 Hasil Data Koordinat dari Video <i>Trial Air</i> Fn 0.36-75% Jarak Parameter Penuh ...	68
Tabel 4-10 Hasil Data Koordinat dari Video <i>Trial Air</i> Fn 0.36-75% Jarak Parameter.....	69
Tabel 4-11 Sudut Heading Kapal untuk Fn 0.58-Dengan Jarak Parameter Penuh.....	70
Tabel 4-12 Hasil Data Koordinat dari Video <i>Trial Air</i> Fn 0.58-Jarak Parameter Penuh	70
Tabel 4-13 Hasil Data Koordinat dari Video <i>Trial Air</i> Fn 0.58-Jarak Parameter Penuh	71
Tabel 4-14 Sudut <i>Heading</i> Kapal untuk Fn 0. 58 dengan 75% Jarak Parameter	72
Tabel 4-15 Hasil Data Koordinat dari Video <i>Trial Air</i> Fn 0.57-75% Jarak Parameter.....	72
Tabel 4-16 Hasil Data Koordinat dari Video <i>Trial Air</i> Fn 0.57-75% Jarak Parameter.....	73
Tabel 4-17 Sudut <i>Heading</i> Kapal untuk Fn 0.72 dengan Jarak Parameter Penuh.....	74
Tabel 4-18 Hasil Data Koordinat dari Video <i>Trial Air</i> Fn 0.72-Jarak Parameter Penuh	74
Tabel 4-19 Hasil Data Koordinat dari Video <i>Trial Air</i> Fn 0.72-Jarak Parameter Penuh	75
Tabel 4-20 Sudut <i>Heading</i> Kapal untuk Fn 0.72 dengan 75% Jarak Parameter	76
Tabel 4-21 Hasil Data Koordinat dari Video <i>Trial Air</i> Fn 0.72-75% Jarak Parameter.....	76
Tabel 4-22 Hasil Data Koordinat dari Video <i>Trial Air</i> Fn 0.72-75% Jarak Parameter.....	77
Tabel 4-23 Rata-Rata Sudut <i>Heading</i> Kapal	78

BAB 1

PENDAHULUAN

1.1. Latar Belakang Masalah

Industri Pelayaran merupakan salah satu pekerjaan yang paling beresiko di dunia, karena apabila terjadi kecelakaan berupa tubrukan bisa berdampak kerugian material, ekologis serta bisa menyebabkan kecacatan fisik dan bahkan kehilangan nyawa. Oleh karena itu, sebuah program navigasi *autonomous* perlu diciptakan sehingga apabila faktor manusia gagal, maka kapal bisa mengetahui dan bertindak untuk menyelamatkan pelaut dan barang muatannya dari kecelakaan.

Pada dunia pelayaran, laut Indonesia merupakan salah satu laut yang memiliki lalu lintas kapal tertinggi di dunia, dimana sekitar 40% kargo dunia lewat laut Indonesia (Anwar, 2020), tentu pada dari padatnya lalu lintas laut ini menimbulkan potensi kecelakaan berupa tubrukan antar kapal, terutama kapal niaga yang berlayar di pelabuhan Indonesia. Mengingat juga bahwa apabila terjadi tubrukan kapal bisa mengancam tidak hanya distribusi barang namun juga bisa mengancam nyawa pelaut dan ekosistem maritim Indonesia.

Beberapa tahun terakhir ini, banyak penilitian untuk pengembangan *Maritime Autonomous Surface Ship* (MASS) dimana tujuan MASS ini sendiri adalah untuk mengurangi *human error* saat pelayaran sehingga banyak prototipe MASS yang sudah dibuat sebelumnya (Liu et al., 2016) (Schiaretti et al., 2017). Tujuan dari studi-studi yang sudah dilakukan sebelumnya adalah untuk menghilangkan manusia dari operasi pengendalian, namun hal ini belum bisa di capai.

Ada beberapa cara yang bisa digunakan untuk menghindari kecelakaan ini (Huang et al., 2020) (Lagisetty et al., 2013), seperti meningkatkan otomasi pada navigasi kapal dengan pengembangan sistem *autonomous* kapal (Akdağ et al., 2022). Dari perkembangan teknologi yang semakin pesat ini terutama di bidang *Artificial Intelligence* (AI) menghasilkan metode *Machine Learning* yang memiliki potensi pemanfaatan besar aplikasinya di bidang *autonomous vehicle* (Catapang & Ramos, 2017). Namun rata-rata pengembangan *autonomous vehicle* masih terlalu fokus dengan mobil, sehingga pengembangan untuk MASS masih kurang dan hanya menghasilkan simulasi (Zhou et al., 2021).

Oleh karena itu, dalam tugas akhir ini akan merancang sebuah konsep sistem kontrol otomatis pada kapal yang berdasarkan *Machine Learning Object Detection*, dimana sistem yang dirancang akan menggunakan konsep *Machine Learning* dan *Object Detection* dari sensor kamera supaya kapal bisa membuat pilihan untuk menghindari tubrukan yang meminimalisir kerugian pelaut, ekonomi dan ekosistem *maritime* Indonesia. ↵kurang object detection

1.2. Perumusan Masalah

Perumusan masalah yang dibahas dalam Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana merancang pemrograman komputer yang mampu mendeteksi objek dengan kamera dan sensor LiDAR?
2. Bagaimana merakit sistem *object detection* untuk *collision avoidance system*?
3. Bagaimana pengujian sistem sensor dan juga aktuator?
4. Bagaimana perfoma sistem *object detection* untuk *collision avoidance*?

1.3. Tujuan

Tujuan dari penulisan Tugas Akhir ini adalah sebagai berikut:

1. Merancang program yang mampu mendeteksi objek dengan sensor berupa kamera dan LiDAR.
2. Merakit sistem *Object Detection*.
3. Melakukan pengujian sistem di darat dan air untuk mendapat data sensor dan aktuator.
4. Mengukur kinerja sistem yang dibuat di air dengan eksperimen pada kolam yang terkendali.

1.4. Batasan Masalah

Batasan masalah yang akan dibahas dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Sudut penghindaran kapal merupakan variable konstan karena Tugas Akhir berfokus pada perancangan sistem Objek Deteksi. Sudut yang maksimal yang bisa dicapai rudder adalah 10°
2. Pembuatan program *Collision Avoidance* tidak mempertimbangkan potensi *Cyber Security*.

3. Program telah dibuat namun perlu dikembangkan untuk aplikasinya di kapal dan perpaduan sensor.
4. Model *Machine Learning* yang digunakan sudah dikembangkan oleh pihak Google dan sistem *Object Detection* sudah ada yaitu *single shot detection* dalam *TensorFlow*.
5. *Prototype* model kapal bukanlah fokus dari Tugas Akhir ini, jadi tidak diperhitungkan sifat dari modelnya.
6. Kecepatan kapal tidak berkurang, sesuai dengan Rules pada COLREGs *Section 2*

1.5. Manfaat

Manfaat yang diharapkan dari pengerajian Tugas Akhir ini adalah sebagai Berikut:

1. Secara akademik, diharapakan dapat membantu berkembangnya ilmu pengetahuan tentang perkapanan serta *Machine Learning* di Indonesia.
2. Dari segi praktek, diharapkan Tugas akhir ini bisa menjadi referensi pembuatan model sistem *collision avoidance* yang lebih optimal dan kompleks serta mampu meningkatkan potensi penggunaan *Machine Learning* di dalam Industri Pelayaran.

1.6. Hipotesis

Hipotesis awalnya adalah pengunaan sensor kamera dan LiDAR sebagai variabel *input* dan metode *Machine Learning* untuk memproses data input tersebut mampu membuat program sederhana Object Detection untuk *Ship Collision Avoidance*.

Halaman ini sengaja dikosongkan

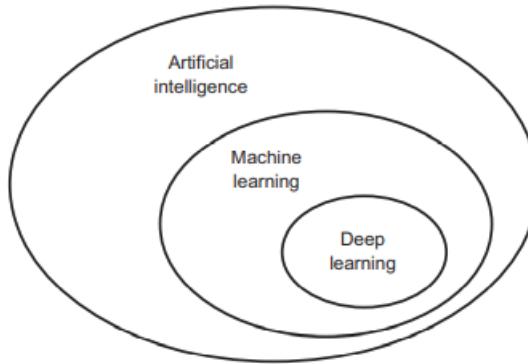
BAB 2

STUDI LITERATUR

2.1. Dasar Teori

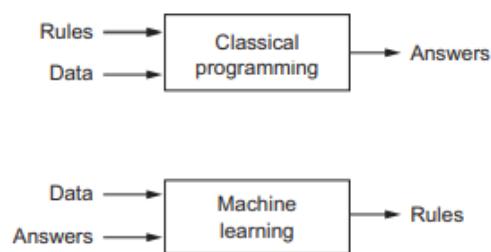
Dalam subab Dasar Teori berisi landasan teori yang berkaitan secara langsung dan digunakan dalam penyelesaian permasalahan dalam Tugas Akhir ini.

2.1.1. *Machine Learning*



Gambar 2.1 Perbedaan AI,ML, dan DL
(Sumber: Chollet, 2018)

Artificial Inteligence (AI) adalah sebuah program komputer yang mampu melakukan tugas yang biasanya menggunakan pengetahuan intelektual manusia dengan menggunakan batasan peraturan-peraturan yang telah ditentukan sebelumnya oleh manusia. Dimana tujuan dari AI ini adalah untuk membantu manusia untuk mengotomatiskan proses yang biasanya memakan sumber daya manusia dan durasi waktu yang panjang apabila dikerjakan oleh manusia.



Gambar 2.2 Perbedaan Cara Pemrograman Konvensional VS Pemrograman ML
(Sumber: Chollet, 2018)

Machine Learning (ML) merupakan bagian dari AI dimana bedanya adalah AI merupakan sebuah program yang di ciptakan dengan mempergunakan peraturan-peraturan yang sudah diketahui sebelumnya (Chollet, 2018). ML digunakan untuk mencari pola atau peraturan yang

tersembunyi dari sekumpulan data yang telah disiapkan sebelumnya, perbedaan norma pemrograman ini bisa dilihat di Gambar 2.2, namun dengan tujuan yang sama seperti AI, ML juga digunakan untuk mengotomasi pekerjaan intelektual manusia namun untuk masalah yang spesifik, seperti menemukan trend dalam harga rumah dan deteksi sell kanker dari sebuah foto X-Ray. Untuk mengetahui bagian *learning* dari *Machine Learning* ini harus mengetahui cara kerja dari algoritma dari ML, ML ini memerlukan 3 (tiga) bagian yaitu:

1. *Input* data : Data yang ingin diproses dalam algoritma ML, bisa dalam bentuk angka numerik, gambar maupun suara.
2. Contoh dari *Output* yang diinginkan : Sering disebut *label*, keluaran dari sebuah data ini digunakan untuk memberitahu algoritma representasi data apa yang ingin dicapai setelah menerima *input*.
3. Pengukuran Performa Model : Komponen ini diperlukan supaya manusia mampu mengukur seberapa jauhnya output sebuah data hasil olahan algoritma dengan output data ekspektasi output yang telah disiapkan di *point* ke 2.

Deep Learning (DL) adalah bagian dari ML dan AI, dimana DL ini merupakan sekumpulan *node* yang saling terhubung untuk membentuk sebuah jaringan yang bisa belajar pola/peraturan yang tersirat dari sekumpulan data yang sangat besar, model ini sering juga disebut dengan *Neural Network*. Yang membuat DL berbeda dari ML adalah DL lebih cenderung menekankan pada pembelajaran yang dilakukan secara bertahap (*layer*) untuk membuat representasi yang berarti dari sebuah kumpulan data tertentu. Dimana relasi AI, ML dan DL ini bisa dilihat di **Error! Reference source not found..**

2.1.2. Dasar Matematis *Machine Learning*

Di balik operasi model ML terdapat algoritma kompleks yang bekerja untuk melakukan prediksi terhadap data yang tidak pernah dilihat model tersebut. Dalam *Machine Learning*, yang dimaksud dengan Algoritma *Machine Learning* adalah *Training* dan *Prediction* (Deisenroth, 2020). Pada awal pembuatan model ML, Data yang dipersiapkan itu diasumsi bisa dibaca oleh komputer dan direpresentasikan dalam format numerikal yang sesuai. Dalam arti data disajikan dalam format tabular, dimana data dalam tiap barisan itu terkait dengan sebuah fitur atau label pada kolom penyimpanan data tersebut (seperti contoh Tabel 2-1). Tentu tidak semua data bisa disajikan dalam format ini, seperti urutan genom DNA, teks dan juga gambar. Untuk penentuan fitur-fitur penting yang berhubungan dengan data itu perlu dilakukannya

feature engineering yang gunanya menentukan fitur mana yang berdampak signifikan dalam sebuah kumpulan data tertentu. Meskipun terdapat data yang bukan berupa numerik (seperti jenis kelamin) bisa di representasikan sebagai angka seperti 1 untuk pria dan 0 untuk perempuan.

Tabel 2-1 Contoh Penyajian Data Tabular

Fitur 1	Fitur 2	Fitur 3	Fitur 4	Fitur 5
Data 1-1	Data 2-1	Data 3-1	Data 4-1	Data 5-1
Data 1-2	Data 2-2	Data 3-2	Data 4-2	Data 5-2
Data 1-3	Data 2-3	Data 3-3	Data 4-3	Data 5-3
Data 1-4	Data 2-4	Data 3-4	Data 4-4	Data 5-4

Meskipun data yang didapat bisa dibaca algoritma ML, tetapi perlu diperhatikan satuan/*units*, skala/*scaling* dan juga batasan/*constraints* data tersebut. Untuk merepresentasikan data terhadap *domain*/fiturnya maka bisa di rumuskan menjadi persamaan (2.1)

$$x \in R^{NxD} \quad (2.1)$$

Dimana $X_{(n)}$ adalah data input dari sebuah *vector* dengan D-dimensional dari angka *real* (R) sebanyak N untuk jumlah data yang ada dalam sebuah kumpulan data dan untuk merepresentasikan indeks data tersebut akan di denotaskan sebagai $n=1,\dots,N$. Serta untuk indeks fitur akan di denotaskan dengan $d=1,\dots,D$. Setelah data vektor telah direpresentasikan, maka perlu dibuatnya fungsi prediksi (*predictor*) dengan menggunakan model, yang bisa berarti sebuah fungsi dan model probabilitas. Dimana *predictor* sebagai fungsi itu bekerja dengan menerima data input (fitur dari vektor) akan menghasilkan *output*. Contoh paling sederhana dari fungsi (2.2).

$$f_{(x)} = \theta^T x + \theta_0 \quad (2.2)$$

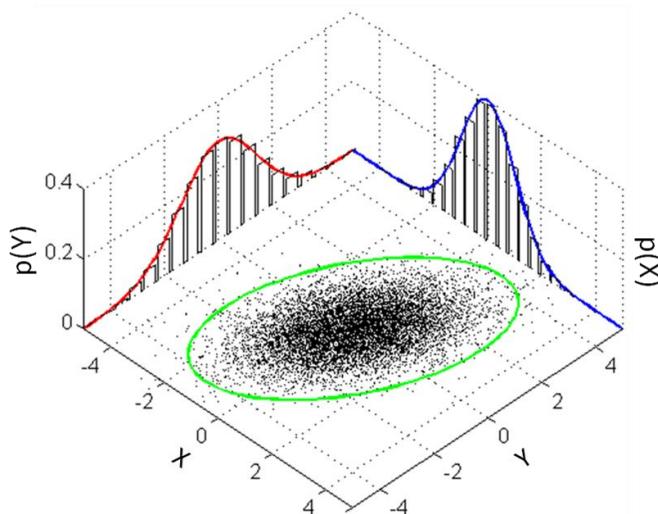
Dimana fungsi akan menghasilkan sebuah output dari inputnya, x , dengan bantuan variable yang tidak diketahui (θ dan θ_T). Fungsi linear itu memberikan keseimbangan yang baik antara generalisasi sebuah masalah yang bisa di selesaikan dan juga jumlah rumus matematis yang diperlukan untuk memecahkan masalah tersebut.

Model yang dilihat sebagai distribusi probabilitas itu berawal dari data yang penuh dengan *observation noise*, kesalahan dari jumlah asli sebuah sistem dan jumlah yang dipantau dari pengukuran yang tidak akurat, sehingga dengan penerapan ML diharapkan bisa mengidentifikasi signal data dari *noise*. Berarti perlu ada acara untuk mengkuantifikasikan

dampak dari *noise* tersebut, maka model perlu bisa mengekspresikan ketidakpastian atau keyakinan terhadap nilai yang di prediksi untuk sebuah titik data uji. Untuk melakukan hal ini, maka model lebih baik tidak dianggap sebagai sebuah fungsi tunggal melainkan sebuah distribusi dari fungsi yang mungkin, dalam implementasi TA ini akan di batasi kasusnya sebagai *finite-dimensional parameter*, yang memungkinkan penggunaan model probabilitas tanpa menggunakan proses *stochastic* dan pengukuran acak. Dalam kasus khusus ini, model probabilitas bisa di anggap sebagai distribusi probabilitas *multivariate*, generalisasi distribusi normal satu dimensional (*univariate*) ke dimensi yang lebih tinggi. Dimana distribusi ini bisa di rumuskan menjadi persamaan (2.3).

$$p(x) = \frac{1}{\sqrt{(2\pi)^N |\Sigma|}} \exp \left[-\frac{(x - \mu)^T \Sigma^{-1} (x - \mu)}{2} \right] \quad (2.3)$$

Dari fungsi (2.3) maka bisa dibuat visualisasi dari distribusi *multivariate* menjadi seperti Gambar 2.3.



Gambar 2.3 Contoh Distribusi Probabilitas *Multivariate*

(Sumber: *University of British Columbia*)

Dari pemodelan, maka selanjutnya adalah Langkah *learning* atau pembelajaran. Tujuan utama dari *learning* untuk menemukan model dan parameternya yang sesuai sehingga bisa menghasilkan prediski yang bekerja dengan baik terhadap data yang tidak pernah dilihatnya. Terdapat 3 fase algorithmic yang harus dihadapi saat melakukan pekerjaan *Machine Learning*, yakni:

1. *Predictions* atau *inferences* : Penggunaan *predictor* terhadap data yang tidak pernah dilihatnya, berarti parameter dan pilihan model sudah pasti dan implementasi *predictor* terhadap vektor baru akan menghasilkan input data baru.

2. *Training* atau Estimasi Parameter : Tahap penyesuaian model terhadap data Latihan, berarti tujuannya adalah mencari model yang bekerja bagus terhadap data Latihan namun bisa generelalisasi permasalahan utama data tersebut.
3. *Hyperparameter Tuning* atau pemilihan model: Penyesuaian *hyperparameter* model terhadap data validasi (*cross validation*) agar bisa di implementasikan di dunia nyata. Untuk memastikan jaringan yang telah dibuat itu siap untuk di latih, maka perlu dijelaskan beberapa Langkah saat melakukan kompilasi, yakni:

1. Fungsi *loss* : Sebuah fungsi untuk mengukur perfoma jaringan pada data pelatihan, sehingga model bisa di sesuaikan pada arah yang diinginkan. Fungsi ini bisa di rumuskan menjadi persamaan (2.4)

$$R_{emp(f,x,y)} = \frac{1}{N} \sum_{n=1}^N l(y_n, \hat{y}_n) \quad (2.4)$$

2. Fungsi *optimizer* : Mekanisme dari jaringan agar setiap layer pada jaringan bisa memperbarui dirinya berdasarkan data yang dilihat dan fungsi *lossnya*.
3. Fungsi evaluasi : Fungsi untuk memantau perfoma dari segi akurasi model untuk memproses data input baik dari data pelatihan atau pengujian.

2.1.3. Model *Machine Learning* untuk *Object Detection*

Permasalahan ML itu bisa digeneralisasikan menjadi 4 cabang besar, yakni:

1. *Supervised Learning* : cabang yang bekerja untuk menemukan relasi dari kumpulan data input yang sebelumnya sudah ada dan sudah diberi outputnya. Contoh aplikasi dari cabang ini adalah rekognisi karakter secara visual, rekognisi suara, klasifikasi gambar dan penerjemahan bahasa, namun ada juga contoh yang lebih tidak umum yakni, *sequence generation* (memberi deskripsi dari sebuah gambar), *syntax tree prediction* (peramalan dekomposisi sintaks dari sebuah kalimat), *object detection* (pemberian dan prediksi koordinat kotak pembatas/*bounding box* pada objek yang di kenali oleh algoritma), dan *image segmentation* (memberi lapisan pixel pada gambar yang di kenal)
2. *Unsupervised Learning* : Cabang ini berfokus pada penemuan pola dari data input yang sebelumnya tidak memiliki label atau tidak ada output dengan tujuan untuk visualisasi data, kompresi data atau *denoising* data untuk semakin memahami korelasi data. Contoh dari aplikasi cabang ini adalah *dimensionality reduction* dan *data clustering*.
3. *Self-supervised Learning*: Cabang yang spesifik dalam *supervised learning* tapi berbeda karena label data yang diberikan bukan dari manusia, bisa diartikan bahwa tidak ada ikut

campur manusia dalam loop pembelajaran. Namun masih menggunakan label yang dihasilkan dari data input biasanya untuk algoritma *heuristic*.

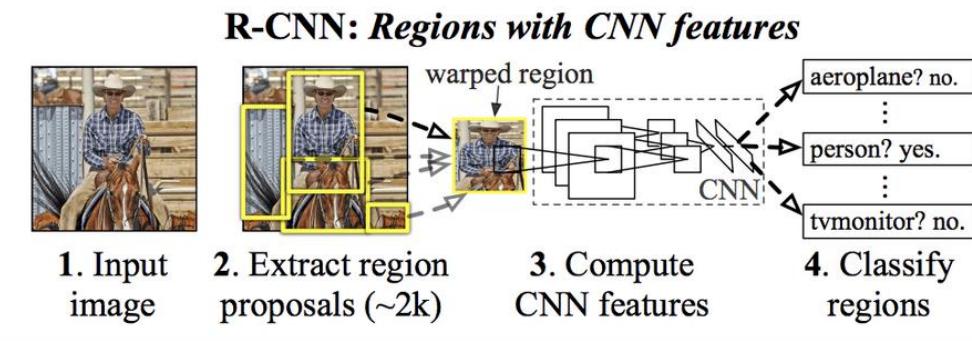
4. *Reinforcement Learning* : Cabang dari ML yang menggunakan sebuah *agent*, bisa di bayangkan sebagai seorang pelajar, yang mendapat informasi dari lingkungannya dan belajar peraturan dari permainan tersebut, dan konsep dari peraturan psikologis, *stick and carrot*, dimana *agent* akan mendapat point jika aksi yang dilakukan itu benar dan mendapat penalty berupa reduksi point jika aksi yang dilakukannya itu salah. Tujuan dari pelatihan *reinforcement learning* itu adalah melatih *agent* untuk mendapat point terbanyak dalam sebuah permainan. Contoh dari cabang ini adalah riset dari program *autopilot* kendaraan *autonomous* level 4 namun sampai saat penulisan laporan ini, cabang dari ML ini masih dalam tahapan yang perlu riset mendalam agar lebih bisa menjamin keamanan dan keselamatan produk.

Selain dari cabang-cabang yang telah dijelaskan, banyak sekali metode yang digunakan dalam cabang-cabang tersebut untuk mencapai hasil yang diinginkan. Metode ini bisa merupakan pengembangan model ML baru atau pengembangan metode pembelajaran/optimalisasi model ML yang sudah ada. Berikut adalah dua kategori model yang umum digunakan:

1. Model dari keluarga R-CNN : Kependekan dari “*Region-Based Convolutional Neural Network*” merupakan keluarga model yang dikembangkan oleh (Girshick et al., 2014). Merupakan model pengaplikasian paling besar dan sukses untuk masalah *object localization, detection, dan segmentation* yang terdiri dari 3 modul, yakni:
 - a. *Region Proposal* : Menghasilkan dan ekstrak kategori dari bagian yang diusulkan (kandidat untuk *bounding box*)
 - b. *Feature Extractor* : Ekstrak fitur dari kandidat bagian dengan menggunakan *Deep Neural Network*
 - c. *Classifier* : Mengklasifikasi fitur dari salah satu kelas yang diketahui dengan model *SVM Classifier*.

Dari 3 modul itu digabung dengan teknik yang disebut dengan *selective search*, yakni mencari sebuah fitur lalu menyamakkannya dari kumpulan target label yang telah disiapkan. Metode ini lambat dibanding versi-versi terbaru. Cara kerja dari model ini bisa dilihat di

Gambar 2.4.

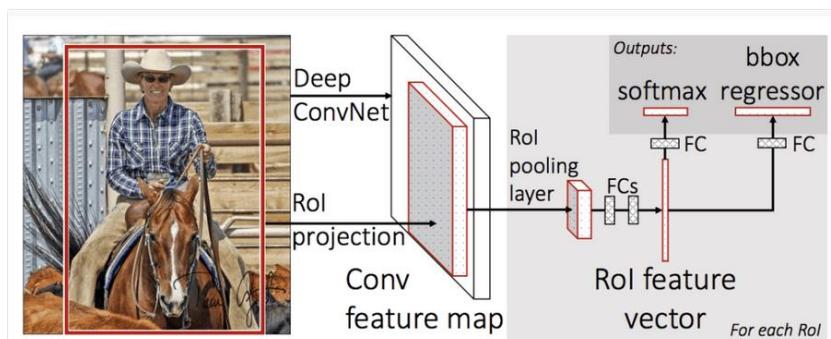


Gambar 2.4 Cara Kerja R-CNN Secara Umum

(Sumber: medium.com)

Dengan sukses dari R-CNN ini maka dikembangkan model yang berdasarkan jenis model ini yakni *Fast R-CNN*, perbedaan pada versi ini adalah dengan penggunaan metode *spatial pyramid pooling* yang mempercepat deteksi dan prediksi model (Girshick, 2015). Jadi cara kerjanya adalah dengan menerima foto dari kumpulan daerah yang diusulkan lalu di masukan dalam lapisan *deep convolutionalp neural network* yang kemudian di masukan dalam model CNN yang telah di latih (seperti VGG-16) untuk ekstraksi fitur dan di akhir dari lapisan itu diberi lapisan khusus yang disebut *region of interest pooling layer* atau *RoI Pooling* yang mengekstraksi fitur spesifik untuk daerah input yang diusulkan. Cara kerjanya bisa dilihat di Gambar 2.5.

Setelah *Fast R-CNN* berhasil menghasilkan deteksi yang lebih cepat, dari pihak riset Microsoft pun mulai mengembangkan dari model yang lama pada 2016 dan menghasilkan model yang lebih cepat dengan basis model yang sama dan di latih dengan kumpulan data COCO, yang kependekan dari *Common Objects in Context*, yang bertujuan untuk memberi usulan dan juga memperhalus daerah yang diusulkan menjadi bagian dari proses pelatihan, yang disebut dengan *Region Proposal Network* (RPN).

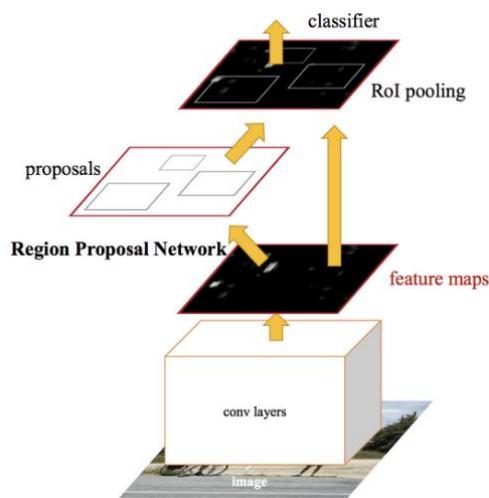


Gambar 2.5 Cara Kerja Fast R-CNN Secara Umum

(Sumber: medium.com)

Daerah ini kemudian di fokuskan lagi dengan *Fast R-CNN* di dalam desain satu model, pengembangan ini mengurangi jumlah daerah yang diusul dan mempercepat waktu operasi pengujian untuk model hampir *real-time* (sudah mendekati waktu realita) dengan perfoma yang sudah *state-of-the-art*. Meskipun disebut memakai satu model yang sudah dipersatukan, desain arsitektur dari model ini terdiri dari 2 modul, yakni:

1. *Region Proposal Network* : Sebuah *Convolutional Neural Network* untuk mengusulkan daerah dan jenis objek yang mungkin ada pada daerah tersebut.
2. *Fast R-CNN* : Sebuah *Convolutional Neural Netwok* untuk ekstrasi fitur dari daerah yang diusul dan mengeluarkan *bounding class* serta *label* dari objek tersebut.



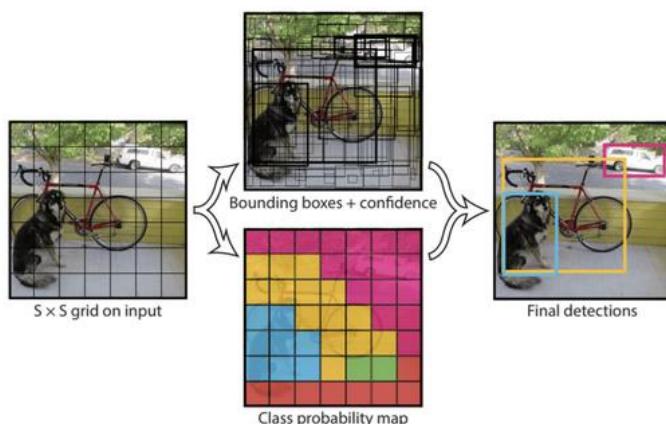
Gambar 2.6 Cara Kerja *Faster R-CNN* Secara Umum

(Sumber: medium.com)

RPN bekerja dengan mengambil output dari *pre-trained deep CNN* dan melewati jaringan kecil di *feature map* dan mengeluarkan usulan beberapa daerah serta prediksi kelasnya untuk tiap daerah tersebut. Daerah usulan itu adalah *bounding boxes* yang berdasarkan *anchor boxes* atau bentuk yang telah ditentukan sebelumnya untuk mempercepat dan mengoptimalkan daerah yang diusul. Prediksi *class* itu berupa binary, berarti menunjukkan apakah ada objek atau tidak di daerah yang disusul tersebut. Prosedur pelatihan alternatif yang bisa dipakai adalah dengan melatih *sub-network* secara bersamaan, meskipun perlu disisipkan. Sehingga memungkinkan parameter dalam detektor fitur *deep CNN* bisa di sesuaikan untuk kedua tugas bersamaan.

3. Model dari keluarga YOLO : YOLO atau kepanjangannya adalah *You Only Look Once* adalah keluarga model ML yang dikembangkan oleh *Facebook AI Research*. Perbedaannya dengan R-CNN adalah model YOLO menggunakan pendekatan yang menggunakan *Neural Network* tunggal dari ujung sampai ujung untuk mengambil goto jadi input dan

memprediksi *bounding boxes* dan kelas *labelnya* untuk tiap *bounding boxes* secara langsung. Teknik ini menghasilkan akurasi prediksi yang rendah (error local yang lebih banyak) namun perfoma komputasinya bisa dari 45 *frame per second* (FPS) hingga 155 FPS untuk model yang dioptimalkan untuk kecepatan. Cara kerja secara umum model ini adalah dengan memecah input gambar menjadi beberapa sel kisi, dimana tiap sel itu bertanggung jawab untuk memprediksi koordinat *bounding boxes* jika titik tengah *bounding boxes* itu berada di dalam daerah sel tersebut, jadi koordinat x,y yang berkoresponden dengan lebar dan tinggi gambar serta tingkat keyakinan untuk prediksi label gambar tersebut. Contohnya, sebuah gambar itu bisa dibagi menjadi kisi 7x7 dan tiap sel di dalam kisi tersebut bisa memprediksi 2 *bounding boxes* yberarti mungkin menghasilkan 94 daerah yang mungkin untuk melakukan prediksi. Dalam kemungkinan tersebut dan *bounding boxes* dengan keyakinan itu digabung menjadi satu set *bounding box* dan labelnya. Cara kerjanya bisa dilihat di Gambar 2.7.



Gambar 2.7 Cara Kerja Model YOLO Secara Umum

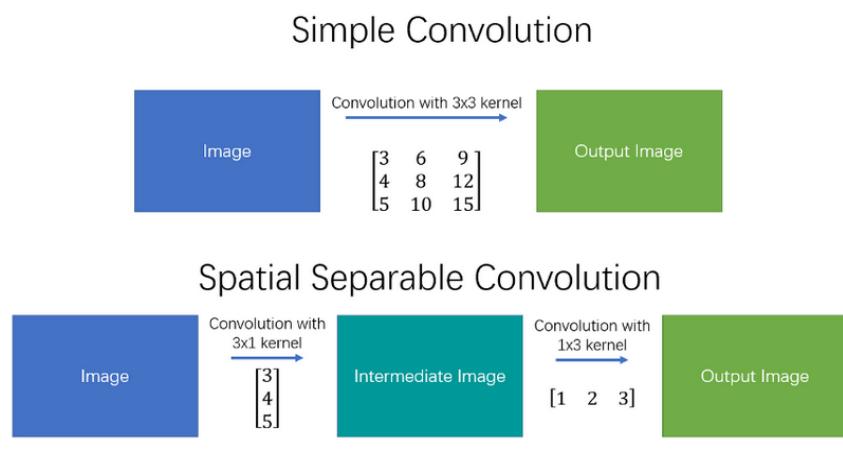
(Sumber: medium.com)

Dengan suksesnya YOLO, maka dikembangkanlah beberapa versi dari model basisnya, yakni YOLOv2 (YOLO9000), sebuah model yang mampu memprediksi 9000 kelas objek yang bekerja mirip dengan *Faster R-CNN*, dan YOLOv3 (Redmon & Farhadi, 2018), model yang merupakan pengembangan dari versi 2 namun hanya pengembangan minor.

2.1.4. MobileNet

MobileNetV1 adalah model ML yang dikembangkan oleh Google yang berfokus pada ukuran yang lebih ringan dan implementasi pada barang yang terhambat oleh perfoma *hardware*. Model ini bekerja dengan prinsip *Depthwise Separable Convolution* (Andrew G.

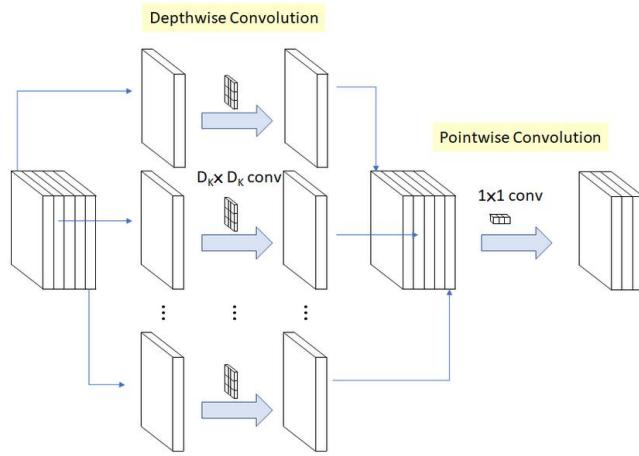
Howard, 2017). Sebelum mengetahui *Depthwise Separable Convolution*, perlu dijelaskan konsep dari *Spatial Separable Convolution*, yaitu sebuah *convolution* yang digunakan untuk operasi dimensi spasial dari sebuah gambar dan kernelnya (lebar dan tinggi, untuk dimensi kedalaman berarti angka pada channel tiap gambar) yang beroperasi dengan membagi kurnel menjadi dua kernel yang lebih kecil (kasus yang paling umum adalah merubah sebuah kernel 3x3 menjadi 1x3 dan 3x1 seperti yang ditunjukan pada Gambar 2.8, jadi daripada operasi *convolution* 9 perkalian, sekarang hanya melakukan operasi *convolution* 3 perkalian sebanyak 2 kali sehingga menaikan perfoma komputasi.



Gambar 2.8 Memisahkan *Kernel* Spasial 3x3

(Sumber: medium.com)

Kekurangan dari metode ini adalah tidak semua kernel itu bisa di pisah menjadi dua kernel lebih kecil dan karena hal ini model hanya bisa pakai bagian kecil yang bisa dipisah menjadi dua itu. Oleh karena itu dari metode itu dikembangkanlah *Depthwise Separable Convolutions* (DSC), metode ini tidak hanya bekerja untuk dimensi spasial tapi juga dimensi kedalaman jadi bisa bekerja untuk kernel yang tadinya tidak bisa dibagi menjadi dua kernel yang lebih kecil. Dengan cara ini, model akan menerima input gambar yang memiliki 3 channel (RGB) lalu setelah beberapa opeasi *convolutions*, *channel* dari gambar akan menjadi lebih banyak sehingga interpretasi dari sebuah gambar itu akan menjadi lebih berarti. Untuk cara kerja metode kurang lebih sama seperti *Separable Convolutions*, jadi sebuah kernel dirubah menjadi *Depthwise Convolutions* dan *Pointwise Convolutions*, seperti yang dintunjukan pada Gambar 2.9.



Gambar 2.9 Cara Kerja *Depthwise Convolutions* pada *MobileNetV1*
 (Sumber: medium.com)

Cara ini menggunakan *DKxDK Spatial Convolution*, jadi apabila terdapat 5 *channel*, maka akan ada 5 *DKxDK Spatial Convolution*, sedangkan untuk *Pointwise Convolution*, hanya terdapat 1x1 untuk merubah dimensinya. Sehingga dengan menggunakan operasi ini, akan didapat biaya operasional sebesar persamaan (2.5)

$$D_k \cdot D_k \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \quad (2.5)$$

Dimana M adalah jumlah *Channel Input*, N adalah jumlah *Channel Output*, D_k adalah ukuran *kernel* dan D_F adalah ukuran *feature map*. Jika menggunakan operasi *convolution* biasa maka rumus empirisnya akan menjadi persamaan (2.6)

$$D_k \cdot D_k \cdot M \cdot D_F \cdot D_F \quad (2.6)$$

Sehingga reduksi komputasi yang didapat apabila menggunakan operasi DSC, sebesar persamaan (2.7)

$$\frac{D_k \cdot D_k \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_k \cdot D_k \cdot M \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_k^2} \quad (2.7)$$

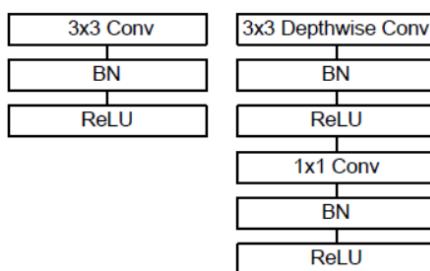
Jadi apabila *DKxDK* itu 3x3, maka akan tercapai komputasi yang 8 sampai 9 kali lebih sedikit dengan reduksi akurasi yang kecil. Arsitektur *layer* model ini bisa dilihat di Tabel 2-2.

Tabel 2-2 Arsitektur *Layer* dari *MobileNet*

Type/Stride	Bentuk Filter	Ukuran Input
Conv / s2	3 x 3 x 3 x 32	224 x 224 x 3
Conv dw /s1	3 x 3 x 3 x 32 dw	112 x 112 x 32
Conv / s1	1 x 1 x 32 x 64	112 x 112 x 32

Type/Stride	Bentuk Filter	Ukuran Input
Conv dw /s2	3 x 3 x 64 dw	112 x 112 x 64
Conv / s1	1 x 1 x 64 x 128	56 x 56 x 64
Conv dw /s1	3 x 3 x 128 dw	56 x 56 x 128
Conv / s1	1 x 1 x 128 x 128	56 x 56 x 128
Conv dw /s2	3 x 3 x 128 dw	56 x 56 x 128
Conv / s1	1 x 1 x 128 x 256	28 x 28 x 128
Conv dw /s1	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 256	28 x 28 x 256
Conv dw /s2	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 512	14 x 14 x 256
5 x Conv dw /s1 + Conv/ s1	3 x 3 x 512 dw 1 x 1 x 512 x 512	14 x 14 x 512 14 x 14 x 512
Conv dw/ s2	3 x 3 x 512 dw	14 x 14 x 512
Conv /s1	1 x 1 x 512 x 1024	7 x 7 x 512
Conv dw / s2	3 x 3 x 1024 dw	7 x 7 x 1024
Conv /s1	1 x 1 x 1024 x 1024	7 x 7 x 1024
Avg Pool/s1	Pool 7 x 7	7 x 7 x 1024
FC / s1	1024 x 1000	1 x 1 x 1024
Softmax / s1	Classifier	1 x 1 x 1000

Perlu dicatat bahwa ada *layer Batch Normalization* (BN) dan ReLU untuk tiap *convolution*, jadi bentuknya seperti di Gambar 2.10. Apabila dilihat dari segi ukuran model, maka model DSC akan memiliki ukuran lebih kecil daripada *Convolution standard*, seperti yang ditunjukkan di Gambar 2.11 dengan reduksi akurasi hanya 1% untuk kumpulan data yang sama.



Gambar 2.10 Perbedaan *Convolution Standard* dengan *Deepwise Separable Convolution* dengan BN dan ReLU

(Sumber: towardsdatascience.com)

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Gambar 2.11 Ukuran Model *MobileNet* dan *Conv MobileNet*

(Sumber: towardsdatascience.com)

2.1.5. Aplikasi *Machine Learning* dalam masalah *Computer Vision*

Computer Vision adalah salah satu masalah terbesar yang dihadapi di dunia transportasi *autonomous*, dimana computer perlu diajari supaya bisa melihat sebuah objek atau jalur seperti manusia agar bisa membuat pilihan berdasarkan data inputan berupa bentuk objek, dimensi objek dan jarak antar objek dan dirinya agar bisa menghindari kecelakaan fatal dan kerusakan bahkan kehilangan nyawa. Dalam dunia ML, salah satu cara untuk menyelesaikan masalah ini adalah dengan menggunakan model jaringan *neural network* berjenis *covnet* yang merupakan kependekan dari *convolutional neural network* (Chollet, 2018). *Covnet* merupakan jenis jaringan yang menerima data berupa gambar dengan format (tinggi_gambar, lebar_gambar, channel_warna_gambar) yang memiliki dimensi yang dapat dikonfigurasi sesuai dengan data input latihan, contoh dari arsitektur *covnet* ini bisa dilihat pada Gambar 2.12.

```
>>> model.summary()
Layer (type)                  Output Shape           Param #
=====
conv2d_1 (Conv2D)             (None, 26, 26, 32)    320
maxpooling2d_1 (MaxPooling2D) (None, 13, 13, 32)    0
conv2d_2 (Conv2D)              (None, 11, 11, 64)   18496
maxpooling2d_2 (MaxPooling2D) (None, 5, 5, 64)     0
conv2d_3 (Conv2D)              (None, 3, 3, 64)    36928
flatten_1 (Flatten)            (None, 576)          0
dense_1 (Dense)                (None, 64)          36928
dense_2 (Dense)                (None, 10)          650
=====
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

Gambar 2.12 Contoh Ringkasan dari Arsitektur *Covnet*

(Sumber: Chollet, 2018)

Covnet menerima data dengan bentuk matrix (28,28,1) berarti sebuah data gambar dengan ukuran tinggi dan lebar 28 dan mempunyai *channel* warna 1 atau gambar hitam/putih yang dialirkan ke dua jenis layer, yakni *Conv2D* dan *MaxPooling2D* yang merupakan layer yang

menerima bentuk *tensor* 3D (Tinggi, Lebar dan *Channel Warna*). Output gambar dari tiap *layer* arsitektur memang cenderung mengecil semakin dalam data di proses dalam jaringan karena di tiap *node* itu mengingat ciri khas dari sebuah gambar tersebut sehingga fiturnya berkurang semakin sering di saringnya gambar. Pada layer terakhir dari arsitektur perlu diingat bahwa jenis data yang dikeluarkan itu berupa data *tensor* 3D sehingga harus di kemas menjadi data 1D sehingga output bisa di terima di *dense layer* terakhir, dalam kasus contoh di Gambar 2.12 adalah klasifikasi gambar dengan 10 label *output*.

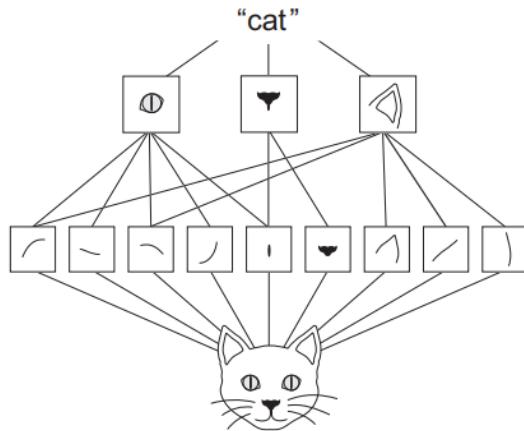


Gambar 2.13 Data Gambar yang Dipecah menjadi Pola Unik

(Sumber: Chollet, 2018)

Perbedaan dari *convolution layer* dengan *connected layer* adalah *dense layer* itu mempelajari pola dari sebuah data secara global, dalam arti dia hanya mempelajari secara garis besar pola yang paling sering ketemu untuk 1 set data yang diterimanya, sedangkan *convolution layer* itu mempelajari pola data secara lokal, bisa dilihat pada Gambar 2.13 dimana sebuah data gambar di pecah menjadi pola yang unik dan di ingat dalam sebuah jendela 2D dengan dimensi tertentu sehingga memberi karakteristik unik dari *covnet* ini menjadi dua point yang menarik:

1. Pola yang dipelajari itu bisa diterjemah secara invariant, artinya apabila jaringan telah mempelajari pola pada salah satu jendela kecil pada Gambar 2.13 maka *covnet* akan mengingat dan mampu mengetahui pola tersebut dimana saja, tidak seperti *dense layer* yang harus mempelajari pola baru jika terdeteksi pada data/lokasi baru. Sehingga membuat *covnet* jauh lebih efisien untuk memproses data berupa gambar dan memerlukan sampel pelatihan yang lebih sedikit untuk mempelajari representasi untuk mendapat kemampuan pengetahuan umum



Gambar 2.14 Representasi dari Hirarki Spasial dari Sebuah Konsep Visual sebuah Kucing

(Sumber: Chollet, 2018)

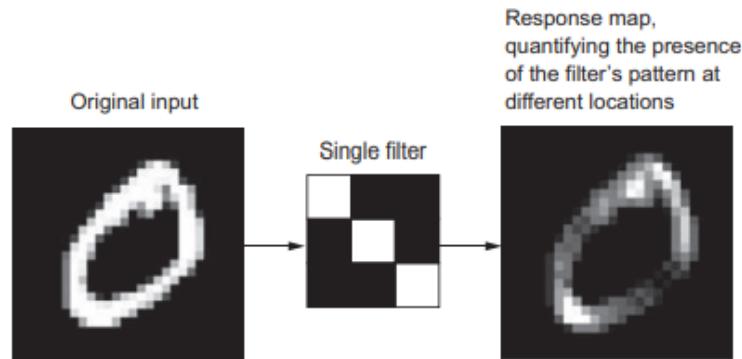
2. *Covnet* bisa belajar pola spasial secara hirarki, artinya lapisan pertama dari arsitektur *covnet* akan mempelajari pola lokal kecil, seperti ujung dari sebuah objek, lalu lapisan kedua akan mempelajari pola yang lebih besar dari fitur dari lapisan pertama dan seterusnya karena hal ini *covnet* bisa mempelajari konsep visual yang lebih kompleks dan abstrak secara efisien. Seperti yang diilustrasikan pada Gambar 2.14.

Sebuah *convolutional* itu bekerja pada *tensor* 3D yang disebut *feature maps*, dengan dua sumbu spasial (tinggi dan lebar) juga sumbu tambahan, yakni sumbu kedalaman (juga disebut sebagai sumbu *channel/warna*). Untuk gambar yang memiliki warna RGB maka channel itu memiliki nilai 3.

Convolution bekerja dengan mengambil bagian dari *input feature maps* dan mengimplementasikan transformasi yang sama terhadap semua bagian tersebut sehingga menghasilkan *output feature map*. Perlu diingat bahwa *output feature map* ini masih berupa tensor 3D, mempunyai lebar dan ketinggian namun kedalaman/channelnya berubah-rubah karena kedalaman output itu parameter dari lapisan dan *channel* yang berbeda dalam sumbu kedalaman tidak lagi merepresentasikan sebuah warna spesifik dalam input RGB, namun merepresentasikan sebuah *filter* yang menyandikan sebuah aspek dari data input.

Dari tiap hasil proses sebuah lapisan filter akan menghasilkan *response map*, sebuah kumpulan respon terhadap pola yang terdeteksi pada lokasi yang berbeda dalam data input seperti yang ditunjukkan pada Gambar 2.15. kumpulan dari seluruh map ini disebut dengan *feature map*, yakni semua dimensi dalam sumbu kedalaman itu berupa fitur atau filter dan

output yang berupa tensor 2D $[::,n]$ itu dinamakan *spatial map* dari respons terhadap filter pada data input.



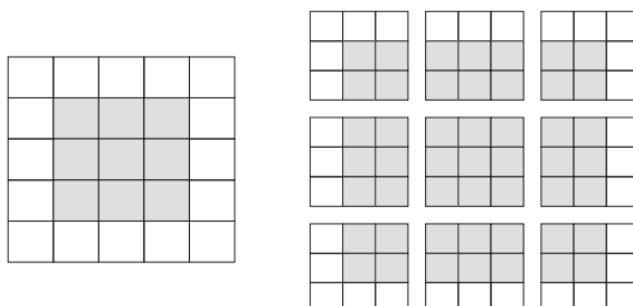
Gambar 2.15 Contoh dari *Response Map*

(Sumber: Chollet, 2018)

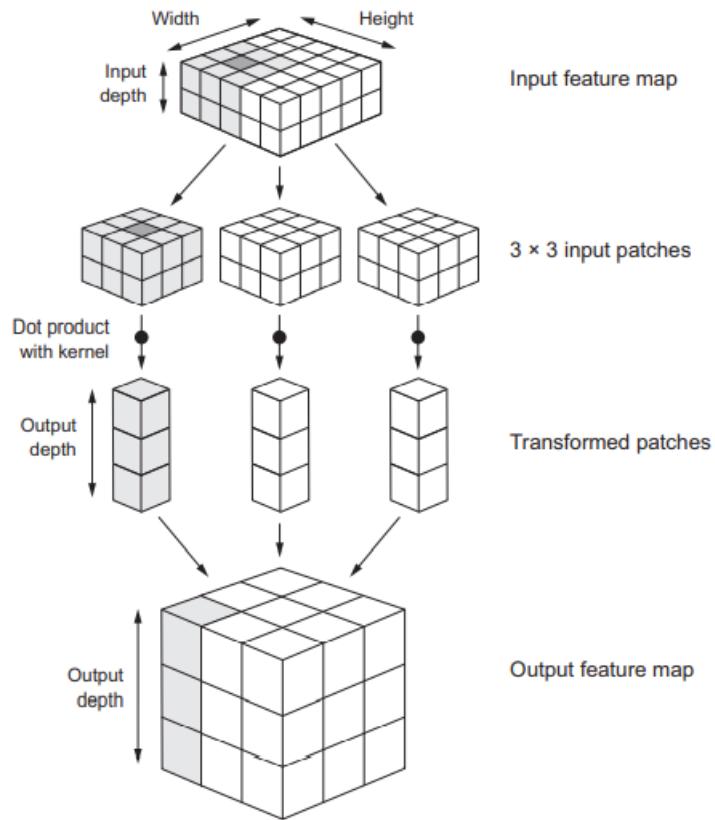
Parameter kunci yang mendefinisikan sebuah *convolution* adalah

1. Ukuran dari area ekstraksi dari data input, biasanya dalam bentuk jendela 3×3 atau 5×5
2. Kedalaman dari output *feature map*, jumlah dari filter yang di komputasi pada *convolution*.

Cara kerja dari kedua parameter itu adalah dengan memindahkan jendela yang berukuran 3×3 atau 5×5 itu terhadap data *input tensor* 3D *feature map* yang kemudian fiturnya akan di ekstraksi (bentuk (tinggi_jendela,lebar_jendela,kedalaman_input)). Lalu untuk tiap tambalan 3D itu di transformasi lewat operasi produk tensor dengan berat atau *weight* dari matrix yang sama, disebut dengan *convolutional kernel* menjadi *vector* 1D dengan bentuk (kedalaman_output,) yang kemudian akan di sambung ulang menjadi *output map* 3D dengan bentuk (tinggi, lebar, kedalaman_output). Untuk tiap lokasi spasial dalam *output feature map* itu berkorelasi pada lokasi yang sama pada *input feature map*, jadi informasi yang ada pada output sebelah bawah kanan itu akan sama seperti informasi yang ada pada data input. Untuk visualisasi kerja dari *convolutional* bisa dilihat pada Gambar 2.17.



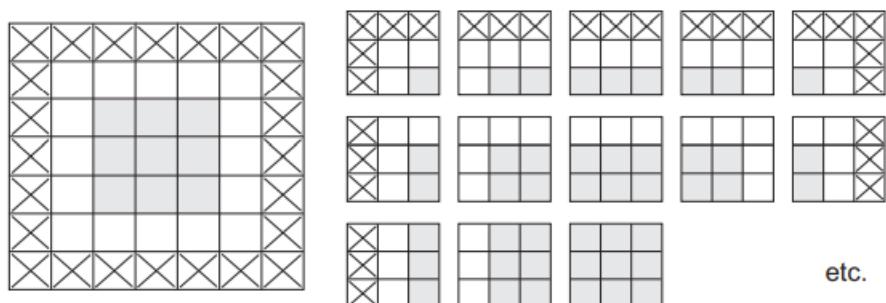
Gambar 2.16 Visualisasi Lokasi dari Tambalan 3×3 pada Input Feature Map Berukuran 5×5
(Sumber: Chollet, 2018)



Gambar 2.17 Cara Kerja *Convolution*

(Sumber: Chollet, 2018)

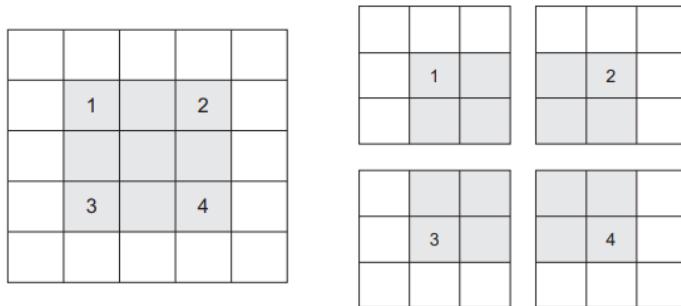
Perlu dicatat bahwa ukuran lebar dan tinggi dari output itu bisa berubah dari lebar dan ketinggian data input, hal ini dikarenakan:



Gambar 2.18 Guna *Padding* pada *Input* 5x5 untuk Ekstrasi 25 Tambal Berukuran 3x3

(Sumber: Chollet, 2018)

1. Efek *border* yang bisa di mitigasi dengan menggunakan *padding* pada *input feature map*. *Padding* adalah cara untuk menambah jumlah kolumn dan baris pada *input feature map* agar mendapat *output feature map* yang mempunyai dimensi spasial yang sama seperti inputnya.



Gambar 2.19 Tambal *Convolution* dengan ukuran 3x3 dan dengan *stride* 2x2

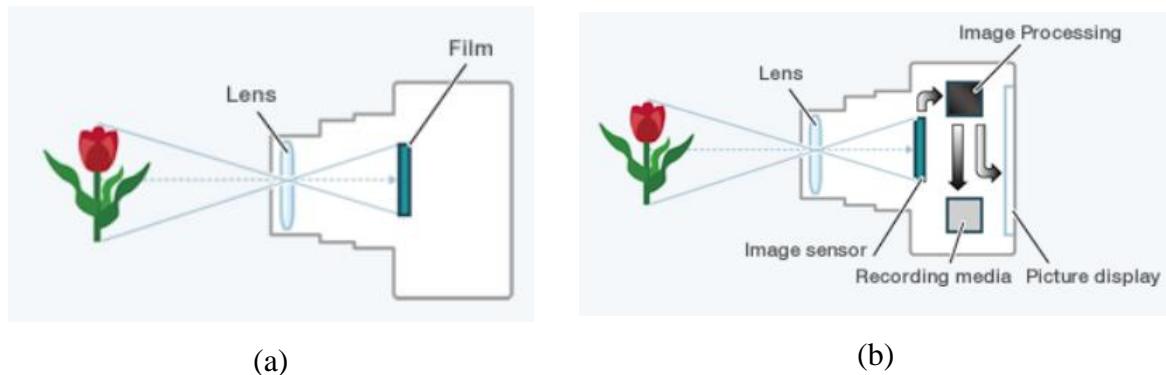
(Sumber: Chollet, 2018)

2. Penggunaan *stride*, jarak antar jendela yang ada pada *convolution*. Dimana biasanya memiliki nilai 1, memungkinkan juga mengubah nilai ini menjadi nilai yang lebih tinggi. Pada Gambar 2.19 adalah contoh dari sebuah tambalan 3x3 dengan *stride* 2 yang memiliki input 5x5 tanpa *padding*. Penggunaan *stride* 2 itu berarti tinggi dan lebar dari *feature map* itu di *downsampled* (di jadikan lebih kecil) dengan faktor 2 (ditambah dengan perubahan yang mungkin terjadi oleh efek border). Meskipun *stride convolution* itu jarang di pake di lapangan, tapi tetap menjadi salah satu metode yang berguna untuk beberapa jenis model. Cara lain untuk melakukan *downsampling* adalah dengan menggunakan operasi *max-pooling*. *Max-pooling* ini bekerja dengan mengekstraksi jendela dari *input feature map* dan mengeluarkan nilai maximal untuk tiap channel. Mirip dengan *convolution* namun operasi ini tidak merubah tambalan lokal dengan transformasi linear, tapi dengan operator *max tensor*. Apabila *max-pooling* tidak dipakai maka proses pembelajaran akan menjadi tidak efisien dan hasil model akan menjadi terlalu besar.

Max-pooling itu bekerja dengan jendela ekstrasi fitur dari *input feature map* dan mengeluarkan nilai terbesar untuk tiap *channel*, sehingga secara konsepsual cara kerjanya mirip dengan *convolution*. Bedanya itu kalau di *convolution* itu ada transformasi lokal pada tambalan via transformasi linear yangh telah di pelajari (*convolution kernel*), *max-pool* itu di transformasi dengan operasi *max tensor*. Apabila tidak menggunakan *max-pooling* maka model akan menjadi tidak kondusif untuk mempelajari fitur secara *spatial hierarchy* dan *output* terakhir dari *feature map* akan menjadi sangat besar karena tidak di *downsample*. Kesimpulannya adalah *downsampling* dengan metode *max-pool* itu digunakan untuk mengurangi ukuran dari koefisien *feature-map* yang akan di proses, dan juga menimbulkan *spatial-filter hierarchies* dengan *convolution* yang beturut-turut yang melihat jendela yang semakin besar. Meskipun *max-pooling* bukan satu-satunya metode untuk dowsampling, bisa juga dengan menggunakan *average pooling* atau dengan menggunakan *layer stride* sebelum lapisan *convolution*, namun

metode *max-pooling* ini biasanya bekerja lebih baik daripada solusi alternatif yang telah dijelaskan. Karena fitur itu cenderung untuk *encode spatial presence* beberapa pola atau konsep terhadap tambalan besar dalam *feature map* dan juga lebih informatif untuk melihat *presence* maksimal yang ada pada fitur yang berbeda dari apa melihat rata-ratanya.

2.1.6. Sensor Kamera



Gambar 2.20 Cara Kerja Kamera Analog/Film (a) dan cara Kerja Kamera Digital (b)

(Sumber: ricoh-imaging.co.jp)

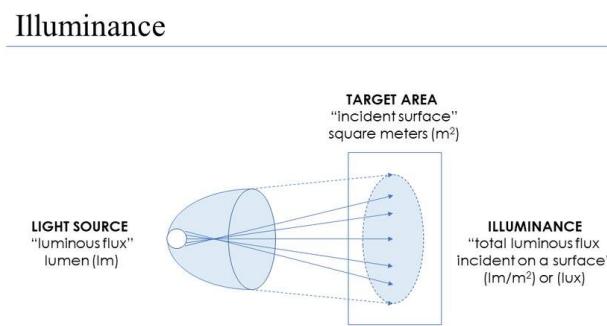
Definisi umum dari kamera merupakan alat yang merekam gambar visual dalam bentuk film, foto atau signal video. Kamera dibedakan menjadi dua kategori besar, yakni kamera digital/modern dan kamera analog/retro. Cara kerja dari kamera secara umum itu kamera akan mengambil cahaya photon, biasanya dalam spektrum yang terlihat pada manusia namun bukan berarti kamera tidak bisa mengambil cahaya dari spektrum elektromagnetik lain, seperti yang bisa dilihat pada Gambar 2.20. Secara global kamera yang ada pada pasar adalah kamera digital yang digunakan untuk dokumentasi foto/video pada spektrum cahaya kasat mata, dengan mekanisme cahaya akan masuk kedalam kotak tertutup melalui lensa konveks dan gambar akan terekam pada medium yang sensitif terhadap cahaya (pada kamera analog biasanya menggunakan film, kalau pada digital menggunakan sensor gambar), sebuah mekanisme *shutter* atau penutup akan mengendalikan lama dari waktu cahaya bisa masuk ke dalam kamera. Hampir semua kamera memiliki alat *viewfinder*, alat yang menunjukkan pemandangan yang akan terekam dan kemampuan untuk mengendalikan focus dan juga *exposure*, banyaknya cahaya per unit area dalam bidang gambar yang disebut *Illuminance* (total insiden fluks cahaya pada permukaan per satuan luas untuk mengukur seberapa banyak cahaya yang terpancar dalam permukaan bisa dihitung menggunakan fungsi (2.8).

$$\Phi_v = 683.002 \frac{lm}{W} \cdot \int_d^{\infty} \bar{y}(\lambda) \Phi_{e\lambda}(\lambda) d\lambda \quad (2.8)$$

Dimana:

- Φ_v : Fluks pencerahan dalam lumens
- $\Phi_{e\lambda}$: Fluks *Spectral Radiant*, dalam watts per nanometer
- $y(\lambda)$: Juga sering diketahui sebagai $V(\lambda)$, itu fungsi *luminosity*, tidak memiliki dimensi
- λ : Panjang gelombang, dalam nanometer.

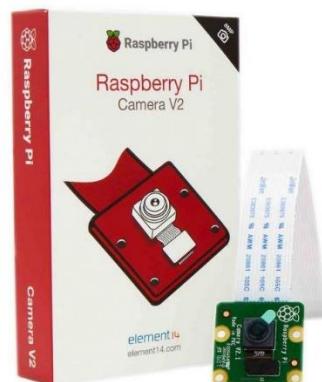
Untuk mengetahui visual dari *Illuminance* ini bisa dilihat dari Gambar 2.21.



Gambar 2.21 Diagram *Illuminance*

(Sumber: hmong.es)

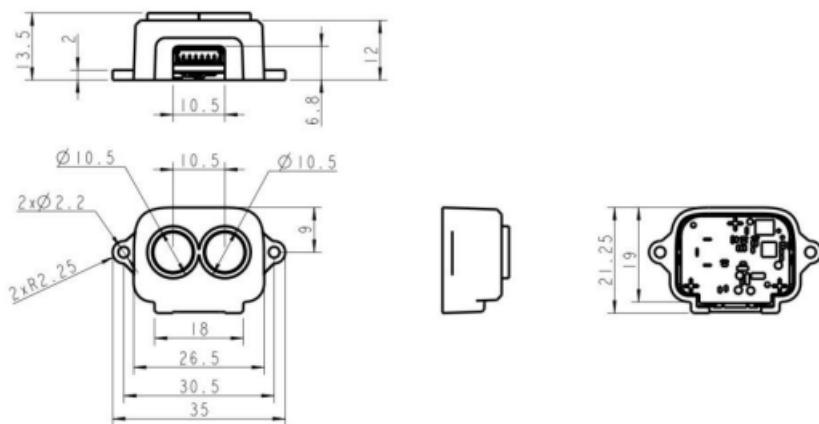
Raspberry Pi menggunakan modul kamera sendiri agar menghubungkan perangkat keras dan perangkat lunaknya gampang. Modul kamera ini bisa dilihat pada Gambar 2.22.



Gambar 2.22 Raspberry Pi Camera Module V2

(Sumber: tokopedia.com)

2.1.7. Sensor LiDAR

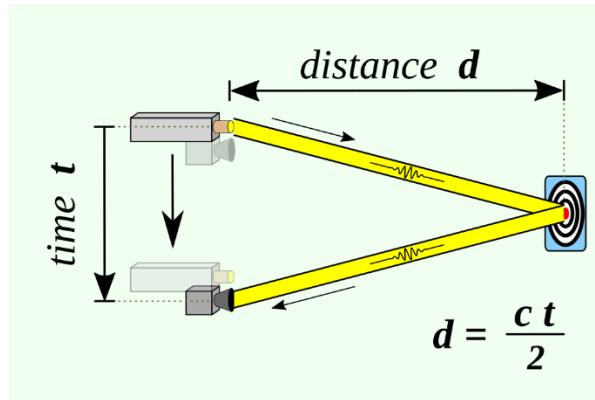


Gambar 2.23 Sensor LiDAR TF-Luna

(Sumber: benewake.com)

Light Detection and Ranging atau yang sering disebut dengan LiDAR ini merupakan sistem sensor aktif, artinya sensor ini menghasilkan energi, dalam kasus ini adalah cahaya, yang digunakan untuk mengukur jarak antar benda dengan menggunakan pantulan dari laser infra merah dan perhitungan *Time of Flight* (ToF). Selain digunakan untuk mengukur, sensor ini juga bisa untuk mencatat pengukuran tersebut dalam bentuk waktu yang diperlukan untuk pantulan cahaya yang di tembak dari LiDAR itu diterima Kembali oleh sensor dan dibagi dengan kecepatan cahaya, karena cahaya memiliki kecepatan yang sangat tinggi jadi bisa dianggap konstan, LiDAR apabila dilihat dari cara merekam datanya itu terbagi menjadi dua jenis, yakni:

1. *Discrete Return LiDAR System*, dimana perekaman titik untuk puncak dari kurva *waveform* itu dilakukan secara individual (*discrete*). sistem ini mengidentifikasi puncak dan mencatat titik untuk tiap lokasi puncak dalam kurva *waveform*. Untuk titik individual itu disebut dengan *returns*. Sistem ini bisa mencatat 1-4 atau bahkan lebih *returns* untuk tiap pulsa laser.
2. *Full Waveform LiDAR System*, di mana sistem mencatat distribusi dari energi cahaya yang Kembali. Data dari sistem ini lebih kompleks namun memiliki kemampuan untuk mencatat lebih banyak informasi dibanding dengan *Discrete Return LiDAR System*.



Gambar 2.24 Visualisasi Pengukuran Sensor LiDAR

(Sumber: ourpcb.com)

Secara umum, kedua jenis LiDAR yang sering ditemukan di pasar itu mengukur jarak dari waktu pantulan menggunakan persamaan (2.9)

$$d = c * t/2 \quad (2.9)$$

dimana d adalah jarak antara sensor dengan target yang ingin diukur, c adalah kecepatan cahaya (konstan = $3*10^8$ m/s atau sekitar $1.079*10^9$ km/jam) dan t adalah waktu yang dibutuhkan sensor untuk menerima cahaya pantulan balik dari objek yang diukur, seperti yang ditunjukkan pada Gambar 2.24. Dalam penggunaan suatu alat, pasti ster dapat keuntungan dan juga kerugian dari alat yang digunakan ini. Berikut adalah keuntungan dan juga kerugian dari penggunaan sensor LiDAR;

1. Keuntungan:

- a. Aman digunakan, sensor ini menggunakan cahaya pada spektrum infra merah yang tidak terlihat oleh mata kasat dan karena sensor menggunakan cahaya, jarak yang diukur tergantung dengan jarak yang bisa ditempuh laser sebelum terdifraksi
- b. Cepat, karena sensor menggunakan media cahaya untuk mengukur maka kecepatan ukuran tergantung dengan kecepatan cahaya
- c. Akurat, sensor LiDAR modern mampu mengukur hingga jarak 2.5 kilometer dengan akurasi 5mm.

2. Kerugian:

- a. Biaya, untuk sensor LiDAR yang digunakan pada mobil *autonomous* itu perlu intensitas dan juga kecepatan serta jangkauan ukur yang sangat besar sehingga biaya untuk sensor akan meningkat secara signifikan.
- b. Satu titik data belum tentu valid, karena pengukuran yang dilakukan satu kali itu belum tentu akurat untuk posisi sesungguhnya

- c. Terlalu banyak data, karena sensor ini bekerja dengan mengirim pulsa untuk tiap kali cahaya diterima kembali oleh sensor maka jumlah pulsa cahaya yang mungkin dikirim untuk satu kali pengukuran yang dilakukan dalam waktu 1 detik itu bisa sampai 20 kali pulsa.

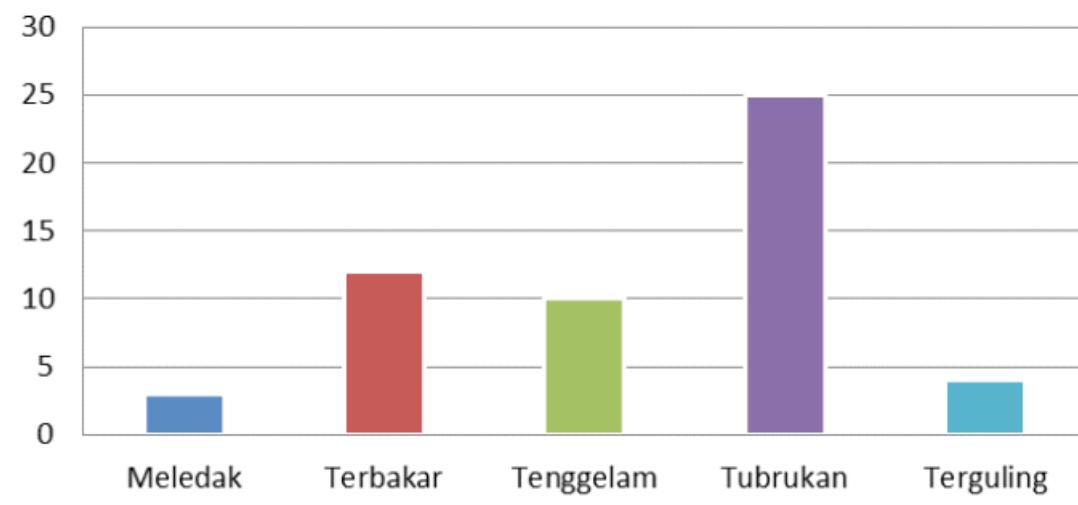
Sensor LiDAR mempunyai banyak sekali aplikasinya mulai dari bidang agrikultur, archeology, atmosphere hingga *autonomous vehicle driving*. Biaya untuk sensor yang digunakan pada kendaraan *autonomous* memang sangat tinggi bahkan hingga ratusan juta rupiah untuk 1 alat sensor LiDAR, namun dengan perkembangan teknologi yang semakin canggih mampu merendahkan biaya untuk sensor ini bahkan untuk ukurannya juga sangat kecil, seperti yang ditunjukkan pada Gambar 2.23 merupakan salah satu contoh sensor LiDAR modern TF-Luna dengan lebar 35 mm dan mampu mengukur jarak hingga 8 meter.

2.2. Tinjauan Pustaka

Dalam subab ini berisi referensi dan/atau hasil penilitian yang telah dilakukan dan relevan digunakan untuk menguraikan teori yang akan digunakan dalam pengerjaan Tugas Akhir ini.

2.2.1. Pemetaan Karakteristik Kecelakaan Kapal di Perairan Indonesia Berdasarkan Investigasi KNKT

Diagram Kecelakaan Kapal berdasarkan Jenis Kecelakaan



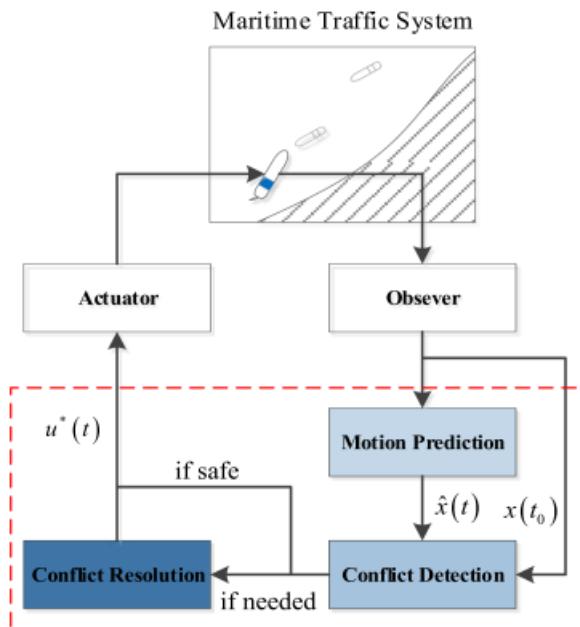
Gambar 2.25 Diagram Kecelakaan Kapal Berdasarkan Kecelakaan Kapal
(Sumber: Hasugian et al., 2018)

Dalam kurun waktu tahun 2007 hingga 2014 terjadi berbagai macam kecelakaan kapal seperti tenggelam, terguling, kandas dan tubrukan (Hasugian et al., 2018). Dari hasil investigasi Komite Nasional Kecelakaan Transportasi (KNKT), dapat disimpulkan dengan analisis

scatterplot data dari laporan KNKT dari tahun 2007 sampai dengan 2014 maka hasilnya menunjukkan bahwa jenis kecelakaan tubrukan kapal adalah jenis kecelakaan yang paling sering terjadi, seperti data yang ditunjukan pada Gambar 2.25, dan cenderung dialami oleh kapal *general cargo* yang berukuran 501 sampai dengan 1500 GT, dimana faktor penyebabnya adalah *watchkeeping* atau pelaksanaan dinas jaga yang belum sesuai prosedur dan peranan nakhoda.

2.2.2. Human and Organisational Factors in Maritime Accidents: Analysis of Collision at Sea using the HFACS

2.2.3. Ship Collision Avoidance Methods: State-of-the-Art



Gambar 2.26 Diagram *Collision Prevention on Board Ship*
(Sumber: Huang et al., 2020)

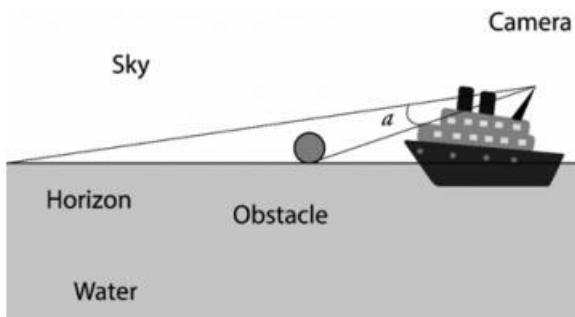
Collision avoidance merupakan topik yang kompleks dan sangat penting dalam dunia pelayaran mengingat lagi dampak tubrukan kapal yang sangat merugikan tidak hanya dari segi ekonomi tapi juga dari segi lingkungan, dimana penyebab utama dari kecelakaan kapal adalah *human error* (Huang et al., 2020). Sehingga banyak peniliti yang sudah mencoba mengembangkan berbagai macam metode, dan dari berbagai metode itu bisa di ambil definisi umum dari *collision avoidance*, adalah sistem yang terdiri dari dua fungsi, yakni *conflict detection* dan juga *conflict resolution*, seperti yang dilihat pada Gambar 2.26. Dimana dalam Tugas Akhir ini akan berfokus dalam fungsi pertama, *conflict detection* dengan menggunakan *Object Detection* dan juga pengukuran jarak aman dengan sensor LiDAR.

2.2.4. Obstacle Avoidance Approaches for Autonomous Navigation of Unmanned Surface Vehicles



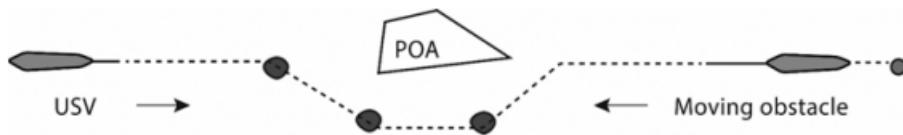
Gambar 2.27 Arsitektur Sistem Autonomous
(Sumber: Polvara et al., 2018)

Untuk membuat sistem *autonomous* perlu beberapa komponen, seperti yang bisa dilihat pada Gambar 2.27, untuk bisa mencapai tingkat otomasi pada kapal. Maka diperlukannya dua komponen utama, yakni:



Gambar 2.28 Estimasi Jarak *Obstacle* dengan Kamera pada USV
(Sumber: Polvara et al., 2018)

1. *Collision Detection* : Untuk bisa merepresentasikan lingkungan secara akurat dengan kombinasi data model secara 3 Dimensional.



Gambar 2.29 Contoh dari *Path Planner*
(Sumber: Polvara et al., 2018)

2. *Path Planner* : Untuk kapal dapat beradaptasi dengan kondisi *existing* yang dihadapi kapal dan mampu membuat pilihan agar bisa menghindari tabrakan dengan beberapa metode, seperti A* Algorithm (Polvara et al., 2018).

2.2.5. Automatic Collision Avoidance System dengan Konsep *Blocking Area*

Blocking Area itu digunakan untuk mengevaluasi resiko tabrakan antar kapal. Dimana *Blocking Area* itu dibagi menjadi 4 kelas yang disebut juga *Collision Area*, *Critical Collision*, *Safety Blocking Area*, dan *Blocking Area with Space* (Furukawa, 2003). Dalam kategori *Blocking Area* tersebut terdapat parameter yang menimbulkan aksi yang harus diambil kapal tersebut, seperti apabila sebuah kapal memasuki *watching area* kapal lain maka kapal yang masuk itu akan menjadi target perhatian dan apabila kapal tersebut melebihi *watching area* maka kapal yang areanya di invasi itu harus membuat pilihan apakah tetap berjalan dalam lajur yang sama atau menghindari kapal tersebut. Untuk bentuk dari *Blocking Area* sendiri itu merupakan kombinasi dari dua garis elips yang didefinisikan dengan R_{bf} , R_{ba} , dan S_b seperti yang ditunjukkan pada Gambar 2.30, dimana untuk R_{bf} dan R_{ba} itu menunjukkan radius

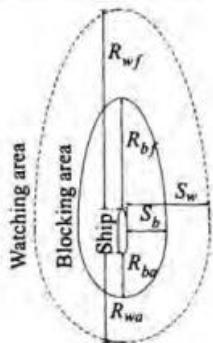
longitudinal dari *blocking area* pada *fore* dan *aft* secara berturutan dan S_b merupakan radius secara transversal untuk kedua area tersebut, jika di modelkan secara matematis maka akan menjadi persamaan (2.10)

$$\begin{cases} R_{bf} = L + (1 + s)T_{90}U \\ R_{ba} = L + T_{90}U \\ S_b = B + (1 + t)D_T \end{cases} \quad (2.10)$$

Dimana untuk L, B dan U adalah panjang, lebar dan kecepatan kapal, T_{90} adalah waktu untuk merubah arah heading menjadi 90° , D_T adalah diameter taktikal dan s&t adalah koefisien yang harus diperhatikan saat kedua kapal tersebut bertemu.

Untuk memprediksi area ini digunakan formula estimasi T_{90} yang dijabarkan secara matematis sebagai fungsi (2.11)

$$T_{90} = \left(\frac{0.67}{U} \right) * \sqrt{AD^2 + \left(\frac{Dr}{2} \right)^2} \quad (2.11)$$



Gambar 2.30 *Blocking Area*

(Sumber: Kijima & Furukawa, 2003)

Dimana A_D adalah kemajuan kapal, untuk nilai A_D dan D_T bisa dicari dengan mengikuti pendekatan rumus (2.12)

$$\begin{cases} \log(A_D / L) = 0.359 \log U_{kt} + 0.0952 \\ \log(D_T / L) = 0.544 \log U_{kt} - 0.0795 \end{cases} \quad (2.12)$$

Dimana U_{kt} adalah kecepatan kapal dalam knots. Parameter yang memberi bentuk dalam *watching area* bisa didapat dari persamaan (2.13)

$$\begin{cases} R_{wf} = L + 2(R_{bf} - L) \\ R_{wa} = L + 2(R_{ba} - L) \\ S_w = B + 2(S_b - B) \end{cases} \quad (2.13)$$

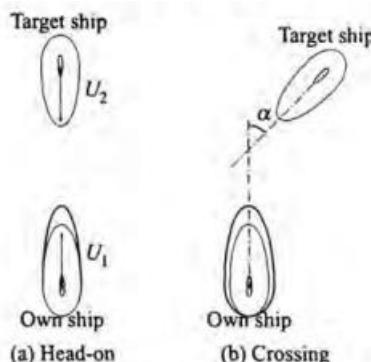
Dimana R_{wf} , R_{wa} , dan S_w adalah radius longitudinal dari *watching area* pada bagian *aft* dan *fore* serta radius transversal bisa dilihat pada Gambar 2.30. Ketika pelaut membuat pilihan untuk

menghindari tabrakan dengan kapal lain maka cara penanggulangannya akan berubah tergantung dengan situasi bertemunya. Dimana jika suasinya normal maka koefisien s&t akan bernilai 1. Arti dari situasi bertemunya kedua kapal itu sesuai yang dijelaskan pada rule 13-15 COLREGs, yakni:

1. *Head-on Situation*: kapal target mendekati kapal sendiri dari depan. Maka dalam situasi ini perlu diperhatikan dari arah *bow* kapal sendiri dengan memperhatikan kecepatan kapal target relatif terhadap kapal sendiri. Perpanjangan radius secara longitudinal direalisasikan dengan parameter s, sedangkan untuk radius transversal bernilai sama dengan S_b dalam kondisi operasional normal. Untuk kondisi ini variable s&t itu dimodelkan dengan persamaan (2.14)

$$S = 2 - \frac{\Delta u}{u}, \quad t = 1 \quad (2.14)$$

Dimana ΔU itu relatif terhadap kecepatan yang diwakili dengan U_1-U_2 dan $U_1 \& U_2$ itu kecepatan kapal sendiri dan kapal target secara beruntun



Gambar 2.31 Perluasan *Blocking Area*

(Sumber: Kijima & Furukawa, 2003)

2. *Crossing Situation*: Nilai s&t berubah tergantung sudut relatif jalur kapal sendiri dengan jalur kapal target, karena diperlukan preventif pada sisi itu penting pada kondisi *crossing* dan tingkat kepentingan itu berbeda tergantung dengan sudut tersebut. Parameter s&t untuk situasi ini mengikuti persamaan (2.15)

$$s = 2 - \alpha/\pi, \quad t = \alpha / \pi \quad (2.15)$$

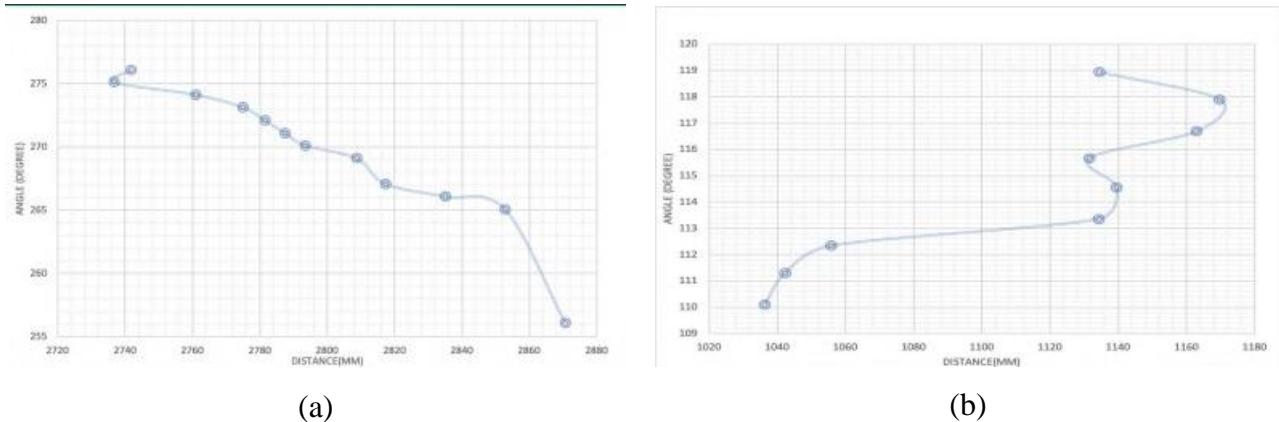
Dimana α merupakan sudut relatif jalur antar kedua kapal tersebut.

3. *Overtaking Situation*: Dalam situasi ini, perhatian diberi pada arah *bow* kapal. Tetapi kecepatan relatif antar kapal kapal yang akan di salip dan kapal yang menyalip

itu kecil dan *blocking area* pada daerah *bow* akan lebih besar dibanding bagian *stern*. Jadi parameter s&t untuk kondisi *overtaking* akan bernilai seperti pada (2.16)

$$s = 1, \quad t = 1 \quad (2.16)$$

2.2.6. Collision Avoidance System for Autonomous Vehicles

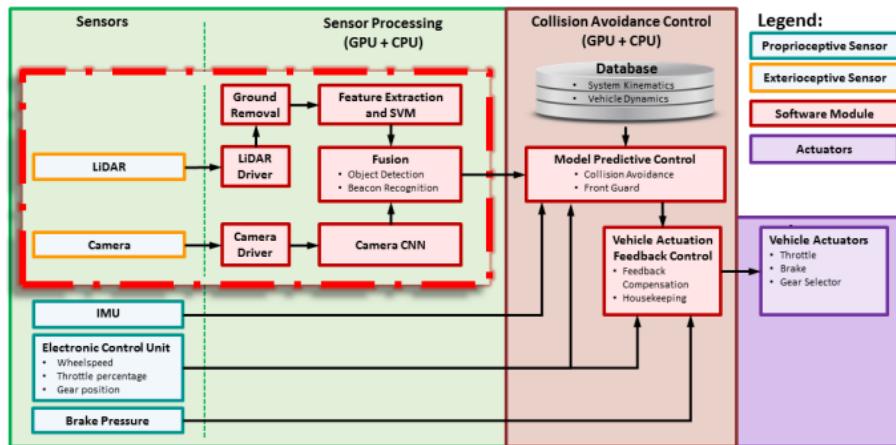


Gambar 2.32 (a) dan (b) Sajian Grafis Data dari LiDAR yang Digunakan pada Percobaan
(Sumber: Aliabady, 2020)

Collision Avoidance System (CAS) atau pendetksi halangan merupakan komponen penting dalam transportasi *autonomous* (Aliabady, 2020). Dalam percobaan ini, prototipe yang dibuat dengan menggunakan sensor LiDAR untuk mendeteksi objek dan untuk pengukuran data berupa jarak, seperti yang di tunjukan pada Gambar 2.32, antar robot dan objek yang perlu dihindari. Hasil dari percobaan menghasilkan data untuk yang menunjukkan keakuratan dari sensor LiDAR yang bisa disimpulkan untuk penggunaan sensor LiDAR pada lingkungan yang tidak bisa di prediksi.

2.2.7. LiDAR and Camera Detection Fusion in a Real-Time Industrial Multi-Sensor Collision Avoidance System

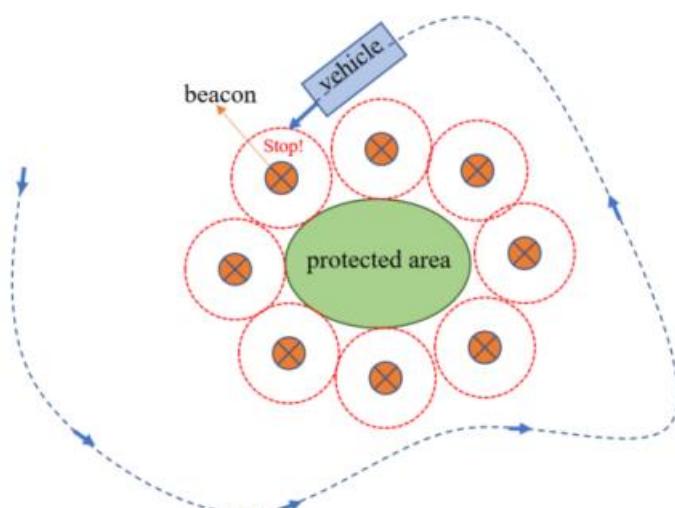
Sistem *Anti Collision* itu sangat diperlukan di seluruh industri baik itu industri manufaktur hingga maritim karena dengan adanya sistem *Anti Collision* yang bersifat otomatis itu bisa mengurangi tidak hanya kebutuhan sumber daya manusia untuk mengoperasikan alat tersebut juga meminimalisir resiko kerusakan alat industri serta kehilangan nyawa. Untuk membuat sebuah sistem *anti collision* yang kokoh maka diperlukannya sistem kombinasi sensor yang bisa bekerja secara harmonis untuk mencapai tujuannya, yakni keamanan dan kesalamatan dalam operasi.



Gambar 2.33 Block Diagram *Fusion System* yang dirancang

(Sumber: Wei, 2018)

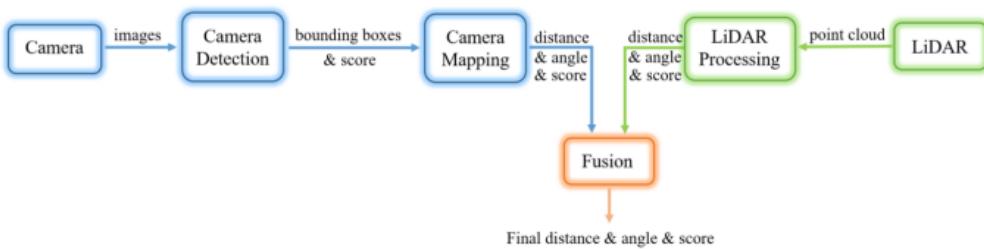
Untuk membuat sistem ini, prototipe itu harus bisa mendeteksi objek berdasarkan warna dan bentuknya, maka diperlukannya kamera RGB. Namun kamera tidak cukup untuk opeasi ini karena sifat dari penghindaran sebuah halangan itu juga harus bisa membaca data jarak antar halangan dengan alat yang di otomasikan dengan pemikiran itu maka diperlukannya sensor pembantu untuk mengukur jarak. Tentu bisa menggunakan sensor lain untuk mengukur jarak namun solusi yang paling sederhana adalah menggunakan sensor LiDAR karena sensor ini tidak membutuhkan daya proses yang tinggi dan juga biaya dari sensor bisa diatur sesuai dengan kebutuhan industri. (Wei, 2018)



Gambar 2.34 Pemakaian *Passive Beacon* sebagai Penanda Daerah yang Dilarang Masuk

(Sumber: Wei, 2018)

Apabila dilihat pada Gambar 2.33, bisa dilihat bahwa kamera dan LiDAR adalah *extroperceptive sensors* yang berarti sensor yang mengambil informasi dari lingkungan robot (seperti jarak, suhu, intensitas cahaya, amplitudo suara dan seterusnya) yang digunakan sebagai data input robot untuk mengambil keputusan dan aksi. Dari kedua sensor itu maka outputnya akan masuk pada bagian permrosesan signal, signal dari LiDAR digunakan untuk mengukur jarak serta mendeteksi barang yang dipasang *beacon* dan menghindari benda lain yang tidak ada *beaconnya* dan kamera digunakan untuk mendeteksi objek dalam bentuk *bounding boxes* dan tingkat kepercayaan deteksi serta klasifikasi, proses ini bisa dilihat pada Gambar 2.35. Penggabungan data dari kedua sensor ini menghasilkan sistem deteksi yang kokoh dalam kondisi industrial, dimana terdapat beberapa daerah yang tidak boleh dilewati transportasi industrial seperti yang bisa dilihat pada Gambar 2.34.



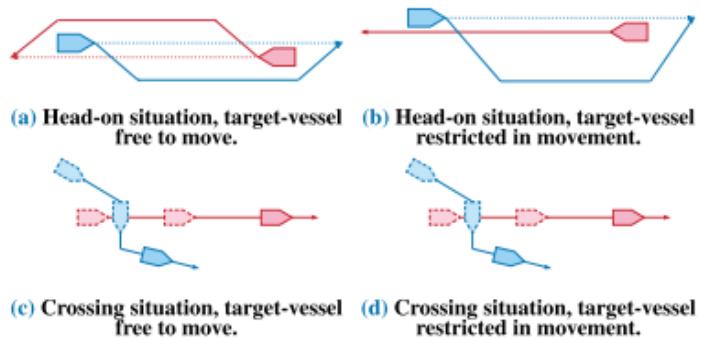
Gambar 2.35 Penjelasan Proses *Fusion System* yang Dirancang

(Sumber: Wei, 2018)

2.2.8. COLREG-Compliant Collision Avoidance for Unmanned Surface Vehicle using Deep Reinforcement Learning

Semua peraturan tentang tubrukan di laut itu taat pada regulasi internasional *Preventing Collision at Sea* (COLREGs). Sebelum kapal *autonomous* dikembangkan, COLREGs dibuat untuk mencegah tabrakan antara dua atau lebih kapal (Meyer, 2020). Intisari dari peraturan-peraturan ini bisa di ambil dua *point* yang penting, yakni:

1. Kapal yang memberi jalan itu harus bertindak awal dan aksinya harus substansial.
2. Kecepatan aman harus dijaga setiap saat, sehingga perubahan alur jalan itu efektif terhadap penghindaran tabrakan dimana terdapat ruang air yang cukup antar kedua kapal tersebut.



Gambar 2.36 Skenario Tabrakan Kapal yang Telah Didefinisikan pada COLREGs

(Sumber: Meyer, 2020)

Juga terdapat point lain yang dijelaskan dalam COLREGs juga membahas perilaku kapal saat menghadapi kapal lain, seperti pada Gambar 2.36:

- *Rule 14 (Head-on Situation)* : Ketika terdapat dua kapal bermesin yang akan bertemu pada arah yang berlawanan atau hampir bertemu sehingga menimbulkan resiko untuk saling bertabrakan maka kedua kapal harus merubah arah ke arah *starboard* sehingga saling melewati dari sisi *portsidenya*.
- *Rule 15 (Crossing Situation)* : Ketika kedua kapal bermesin itu saling menyebrang sehingga menimbulkan resiko untuk tabrakan maka kapal yang melihat kapal lain di sisi *starboardnya* harus menjauh dan, jika perlu, menghindari situasi *crossing* secara keseluruhan
- *Rule 18 (Responsibilities between Vessels)* : Setiap kapal memiliki tanggung jawab sendiri saat berlayar, dimana:
 - a) Kapal bermesin yang sedang berlayar harus menghindari dari kapal lain yang terbatas oleh kemampuannya untuk bermanuver.

Artinya, kapal yang berukuran lebih kecil harus memberi jalan kepada kapal yang berukuran lebih besar, karena kemampuan *maneuvering* kapal yang berukuran lebih kecil akan lebih tinggi daripada kapal yang berukuran besar.

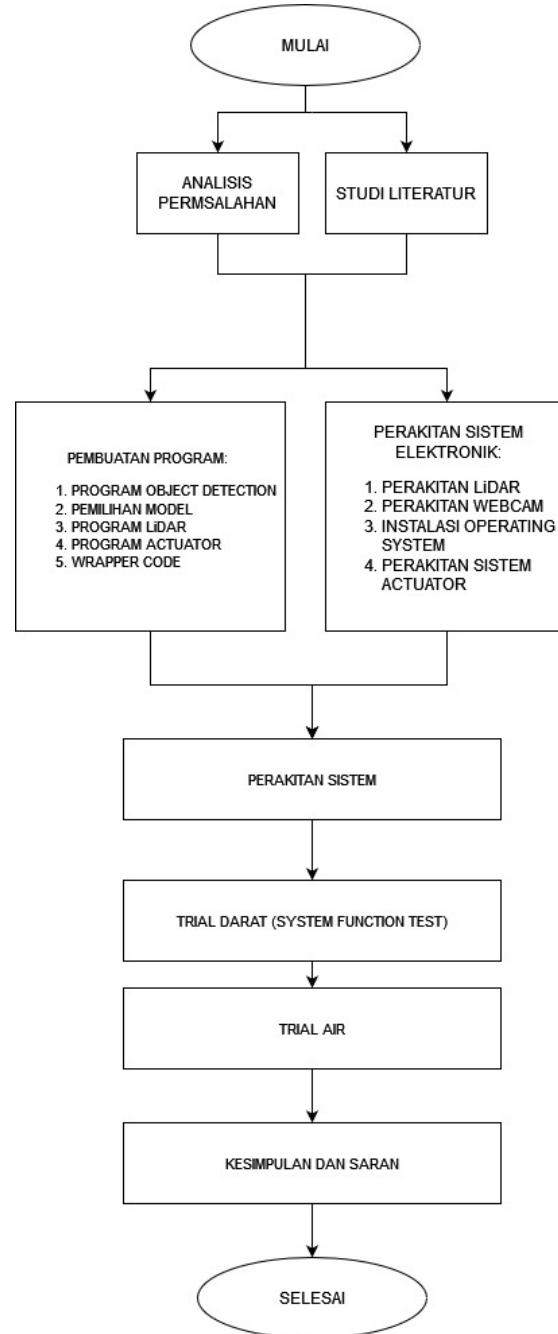
Halaman ini sengaja dikosongkan

BAB 3

METODOLOGI

3.1. Bagan Alir

Secara umum metodologi dalam pengerjaan Tugas Akhir ini bisa di lihat pada Gambar 3.1.



Gambar 3.1 Bagan Alir Pengerjaan Tugas Akhir

3.2. Analisis Permasalahan

Merupakan Langkah awal dalam pengerojan Tugas Akhir, dimana akan dianalisis masalah utama dalam *Collision Avoidance* kapal terutama masalah yang berhubungan dengan *Collision Avoidance*, *object detection*, dan *Fusion Sensor*. Sehingga didapat konsep dari *Collision Avoidance* yang murah dan efektif namun sehingga dapat digunakan di Indonesia.

3.3. Studi Literatur

Pada Tahapan ini meliputi studi untuk Python, matematika untuk *Machine Learning* dan *TensorFlow*. Selain itu studi ini juga meliputi sensor dan perangkat pendukung lainnya, seperti ranging, dan juga aplikasi *object detection*. Pada tahap ini akan juga dilakukannya pertimbangan dalam batasan masalah dan kualitas informasi itu sendiri. Oleh karena itu, beberapa implementasi studi literatur dalam konsep berita diminimalisir selama hal tersebut masih berupa opini.

3.4. Pembuatan Program

Dalam tahap ini, kerangka program untuk sistem *Collision Avoidance* dibuat dengan memecah program besar menjadi program kecil yang memecahkan tiga masalah besar sistem ini, yakni: ↵ benerin

1. Program *Object Detection*
2. Model ML yang digunakan dalam sistem *Object Detection*
3. Program untuk sensor LiDAR
4. Program untuk actuator
5. Program *wrapper* yang menggabung ke empat program tersebut.

3.5. Perakitan Sistem Elektronik

Tahap ini mengerjakan perakitan perangkat keras elektronik untuk sistem *Collision Avoidance* ke dalam model kapal. Penghubungan sensor serta actuator model.

3.6. Perakitan Sistem

Tahap ini dimulai pengujian untuk tiap program kecil untuk memastikan programnya berjalan sebagaimana mestinya dengan *hardware* yang sesuai. ↵ diperjelas

3.7. Trial Darat (*System Function Test*)

Dalam tahap ini, program besar yang sudah dibuat perlu di coba di darat agar memastikan perpaduan dari program-program kecil itu tidak mengganggu kinerja komponen elektronik dan menghasilkan *output* yang diinginkan pada model kapal. Dimana untuk *System Function Test* ini akan diuji untuk pembacaan sensor LiDAR dan juga sudut yang bisa dicapai untuk *rudder* model kapal. Sehingga dari data kedua hal itu bisa, program yang dibuat bisa di kalibrasi sesuai kemampuan model prototipe kapal.

3.8. Trial Air

Tahap pengambilan data perfoma sistem *Collision Avoidance*, dalam tahap ini model akan di coba pada kolam yang terkendali agar meminimalisir hal-hal yang tidak diinginkan. Dimana, kapal perlu dicari juga data sudut *heading* maksimalnya, pengujian kecepatan mesin kapal dan juga perfoma dari *collision avoidance* yang di rancang.

3.9. Kesimpulan dan Saran

Pada tahap terakhir ini, maka perlu diambil kesimpulan dari hasil uji coba yang telah dilakukan untuk menentukan apakah sistem yang di rancang ini efektif digunakan di Indonesia dan apakah perlu dikembangkan lagi menjadi sistem yang lebih kompleks dengan menambah sistem prediksi tubrukannya dengan model ML atau pengendalian actuator lain.

Halaman ini sengaja dikosongkan

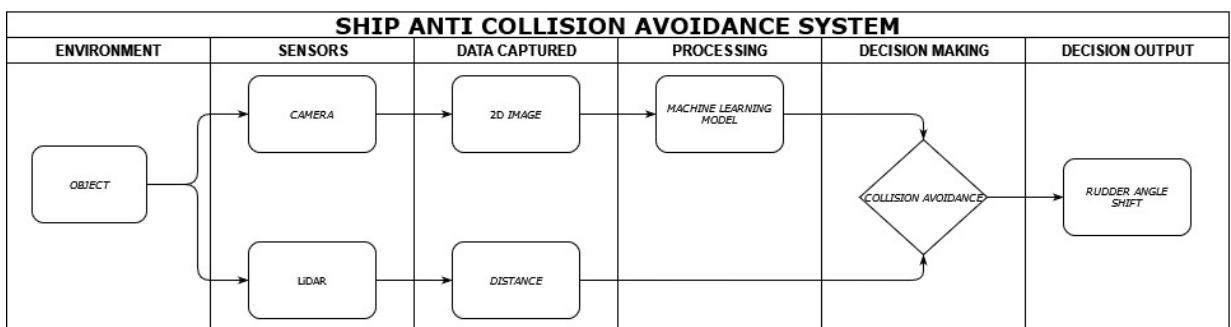
BAB 4

PEMBAHASAN DAN ANALISIS

4.1. Perencanaan Sistem *Collision Avoidance*

Untuk sistem *Collision Avoidance* sendiri yang direncanakan bisa dilihat pada Gambar 4.1. Dimana kapal akan dihadapi oleh sebuah objek, dari objek itu akan diambil dua aspek visual, yakni gambar visual dan jarak yang akan di proses oleh sensor kamera dan LiDAR secara beruntun. Dimana dari data itu akan di proses oleh dua program yang berbeda, gambar visual akan di proses oleh model ML yang akan mendeteksi keberadaan objek tersebut lalu akan merekam kejadian, dan jarak akan di proses oleh *code* dari sensor LiDAR yang menghitung jarak dari waktu pantulan atau *Time of Flight* tembakan laser ke objek tersebut. Dari hasil dua proses itu akan dimasukan ke *decision maker* yang akan menentukan langkah selanjutnya yang harus di ambil kapal agar mampu menghindari tubrukan. Terdapat 3 kondisi yang akan terjadi dalam proses ini, yakni:

1. Kapal akan tetap melaju dengan kecepatan konstan, apabila hanya terdeteksi gambar visual namun jaraknya masih jauh.
2. Kapal akan menggerakan *rudder*, apabila jarak antara kapal dan objek sudah memasuki daerah bahaya, dimana daerah ini sudah ditentukan sebelumnya dari perhitungan *blocking area* (Kijima & Furukawa, 2003).



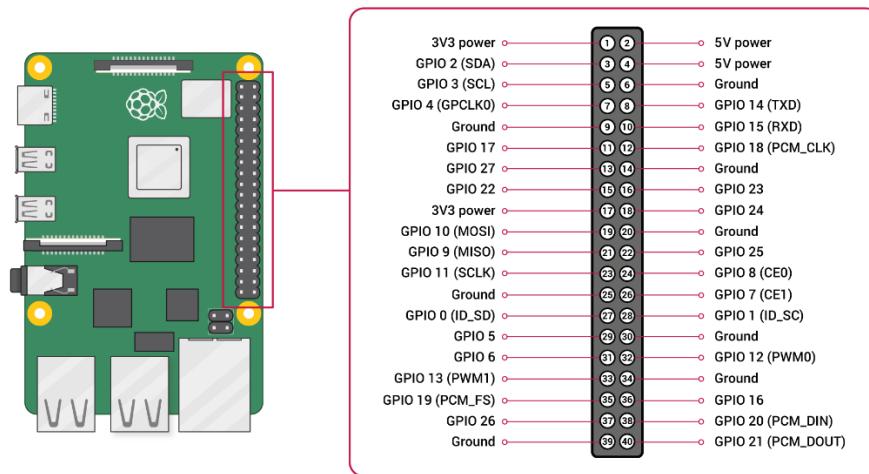
Gambar 4.1 Flowchart Sistem *Collision Avoidance*

4.2. Pemilihan Model *Machine Learning*

Seperti yang dijelaskan di BAB 2, pemilihan model ML yang cocok untuk aplikasi *Object detection* pada *microprocessor* itu tidak cuman langsung ambil dari sumber yang bersifat *open-soourced* namun banyak pertimbangan yang perlu diselesaikan untuk memilih model dan

mengimplementasikannya dalam sebuah sistem. Hal ini dikarenakan sebuah model belum tentu cocok/*compatible* dengan jenis *hardware* dan juga optimal untuk memcahkan masalah yang akan perlu diselesaikannya. Dalam kasus pengerjaan Tugas Akhir ini, Model yang di pilih adalah model MobileNet yang sudah di format dengan *framework TensorFlow Lite* sehingga bisa digunakan pada *microcontroller* dan juga *microprocessor*. Dengan merubah sebuah model untuk kegunaan *hardware* yang terlimitasi oleh perfoma akan mengorbankan akurasi untuk ukuran model yang kecil dan optimal dengan kondisi yang sangat terbatas namun masih memenuhi standar penggunaannya di lapangan. Kelebihan juga dari model satu ini adalah karena dia bersifat *open-source* maka tidak dikenakan biaya penggunaan dan juga bisa di modifikasi sesuai keperluan aplikasi permasalahan kedepannya.

4.3. Perakitan Sistem Elektronik



Gambar 4.2 Diagram Fungsi dari GPIO Port Raspberry Pi

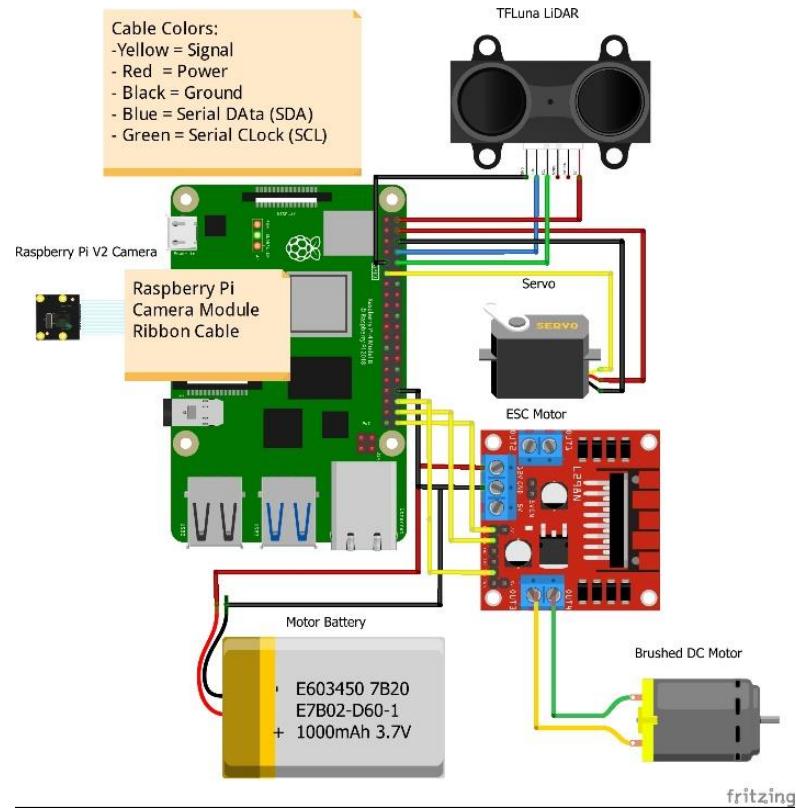
(Sumber: raspberrypi.com)

Setelah model telah didapat, maka perlu dirakit prototipe sistem *Collision Avoidance* dengan *Raspberry Pi* 4 sebagai otak dari sistem, penggunaan *Raspberry Pi* disarankan membaca dahulu dokumentasi dan manual penggunaannya agar bisa mengopersikan *microprocessor* dengan aman. Sensor kamera dan LiDAR harus bisa di sambungkan ke *Raspberry Pi* dengan menggunakan *GPIO jumper cable* atau *Ribbon Cable* agar gampang menghubungkannya dengan *GPIO port* yang sudah tersediakan pada *Raspberry Pi*. Pemasangan kabel sensor ke *Rasoberry Pi* pun tidak boleh asal-asalan, karena *GPIO port* pada *Raspberry Pi* itu di tujuhan untuk fungsi yang spesifik, untuk fungsi tiap *port* bisa dilihat pada Gambar 4.2.

Untuk pemasangan sensor dan *actuator* itu harus terlebih dahulu membaca *user manual/datasheet* dari sensor/*actuator* agar bisa terhubung dengan *Raspberry Pi* dengan aman. Untuk sensor yang digunakan pada sistem ini terdapat:

1. TFLuna LiDAR (Gambar 2.23) ← tambahkan tabel komponen spek dan fungsi
2. *Raspberry Pi Camera Module V2* (Gambar 2.22)

Setelah membaca dan memastikan sensor bisa di hubungkan dengan *Raspberry Pi*, maka sensor boleh dipasang beserta dengan *actuator*nya pada GPIO Pin *Raspberry Pi*. Hasil dari perakitan ini bisa dilihat dengan *wiring diagram* pada Gambar 4.3.



Gambar 4.3 *Wiring Diagram* Sensor dan *Actuator* pada *Raspberry Pi*

Apabila perakitan sistem elektronik telah dilakukan, Langkah selanjutnya adalah untuk memasang sistem rakitan pada model kapal yang sudah disediakan, dengan terlebih dahulu memodifikasi *superstructure* model kapal, seperti pada Gambar 4.4, agar bisa kedap air dan juga menjadi tempat penyimpanan *raspberry pi*. Untuk Tugas Akhir ini, *superstructure* kapal menggunakan tempat makanan yang kedap udara, dimana tempat makanan itu akan diberi lubang pada dasarnya agar kabel bisa dengan mudah diatur dan memberi akses pada mesin dan juga perlengkapan lain yang ada pada lambung kapal. Karena berwarna transparan, tempat makan ini memberi kemudahan memantau apakah *raspberry pi* dan juga komponen elektronik lain bekerja sebagaimana harusnya.



Gambar 4.4 Model Kapal yang Telah Dimodifikasi

Karena keperluan ukuran *superstructure* model kapal cukup besar, untuk sisa ruang yang tidak tertutupi oleh tempat makan, akan diberi lapisan mika transparan tipis., seperti yang bisa dilihat pada Gambar 4.5. Dengan tujuan untuk bisa memantau sistem mekanis kapal, *servo*, motor, *shaft* dan *buse*, dan memberi akses darurat mudah apabila terjadi kesalahan saat pengujian air.



Gambar 4.5 Tampak atas *Superstructure* Model Kapal

4.4. Pembuatan *Code* Sensor dan *Actuator*

Apabila *code* yang telah di kumpulkan itu berfungsi sebagaimana harusnya dengan sensor masing-masing, maka perlu dibuat *code* penggabung atau *wrapper code* yang bisa menghubungkan dan memproses input dari dua sensor tersebut. Jika pemrograman untuk sensor berhasil di proses maka perlu digabungkan dengan *code* untuk *actuator* model, dalam kasus Tugas Akhir ini adalah motor. Dimana akan di lihat perpaduan *code* ini apakah bekerja sebagaimana mestinya atau tidak.

Untuk membuat *code wrapper* ini beberapa fungsi kritis dalam sistem *anti collision*, yakni:

1. Menerima *input* data dari *code Model* ML kamera
2. Menerima *input* data dari *code* pabrik sensor LiDAR
3. Membuat pilihan dari kedua input berdasarkan 1 skenario yang telah ditentukan sebelumnya
4. Menyalurkan hasil pilihan ke *actuator* kapal
5. Sinkronisasi proses yang terjadi selama program berjalan

Dengan pertimbangan tujuan yang telah disebut itu maka *code wrapper* ini akan mengimplementasikan konsep programan yang kompleks, yakni konsep *parallelism*, dimana program yang dibuat akan melakukan 3 proses secara bersamaan secara asikronus dengan menggunakan sumber daya berupa *processor* yang terbatas sehingga terjadi harmonisasi dari kedua proses tersebut dengan memanfaatkan *multicore* pada *processor* modern untuk melakukan *multitasking* atau yang sering disebut dengan *multithreading*.

4.4.1. Penjelasan dan Pengujian *Code* Kamera dan LiDAR

Untuk mengimplementasikan sebuah model *machine learning* dalam sistem ini maka diperlukannya semacam sistem *container* untuk bisa menghubungkan model ke dalam aplikasi yang ingin dikembangkan supaya bisa dipakai untuk memecahkan masalah yang dihadapi. Dalam Tugas Akhir ini, pembuatan sistem tersebut itu memerlukan beberapa komponen yang berupa *python packages*, sekumpulan perintah khusus yang telah dikembangkan oleh pihak lain untuk digunakan bebas oleh komunitas *programmer python*, yang digunakan untuk sistem ini adalah sebagai berikut:

1. OS : Untuk program yang dibuat bisa berinteraksi dengan *file* yang tersimpan di *Operating System* computer, contoh membuat atau hapus *file* atau *directory*

2. Argparse : Untuk program yang dibuat bisa menerima argument-argument tambahan untuk modifikasi program sebelum di eksekusi oleh terminal *Operating System*.
3. OpenCV2/CV2 : Untuk memproses dan memvisualisasikan data gambar
4. Numpy : Untuk memproses dan eksekusi operator matematika yang komplek, contoh differential, integrasi dan regresi dengan efisien
5. Sys : Untuk program bisa berinteraksi dengan *runtime environment*, file yang dieksekusi oleh interpreter *Python*.
6. Time : Untuk memproses dan perintah yang berhubungan dengan waktu, seperti mencatat waktu sistem, kapan program harus sleep dan juga mencatat *runtime* dari sebuah program.
7. Threading : Sekumpulan perintah *python* yang membisakan program untuk eksekusi beberapa bagian/*function* atau *processes* secara *concurrent*, hal ini diperlukan agar menghemat dan mengoptimalkan sumber daya berupa *memory pool* dan *threading* atau *core* yang tersedia pada processor komputer
8. Importlib.util : Untuk *Import library* dari *python* sebelum versi 2.x ke 3.x dengan fungsi yang sama agar bisa di pakai di semua versi *python*.
9. Serial : Kumpulan perintah python untuk menerima dan membaca data yang berupa serial, berarti aliran data dikirim dalam bentuk format bit yang berisikan beberapa informasi (seperti jarak, temperatur, dan kekuatan cahaya)
10. GPIOZero : Kumpulan perintah untuk mengendalikan dan memungkinkan *Raspberry Pi* untuk membaca data yang masuk dari pin GPIO.

Dari sekumpulan komponen yang telah terdaftar itu harus di masukan ke dalam program yang dibuat, cara memasukan komponen tersebut bisa dilihat di Gambar 4.6.

```
# Import packages
import os                                     #To interact with Operating system
import argparse                                #To add arguments when executing file
import cv2                                     #To process Image processing in Object Detection
import numpy as np                             #To process numerical side of Object Detection
import sys                                      #To access system resources
import time                                     #To add time related operation in code
from threading import Thread                    #To enable concurrent code execution
import importlib.util                          #To be able to other library utilities
import RPi.GPIO as GPIO                        #To be able to use Raspberry Pi's GPIO Pins
import serial                                   #To be able to read serial data from LiDARS
```

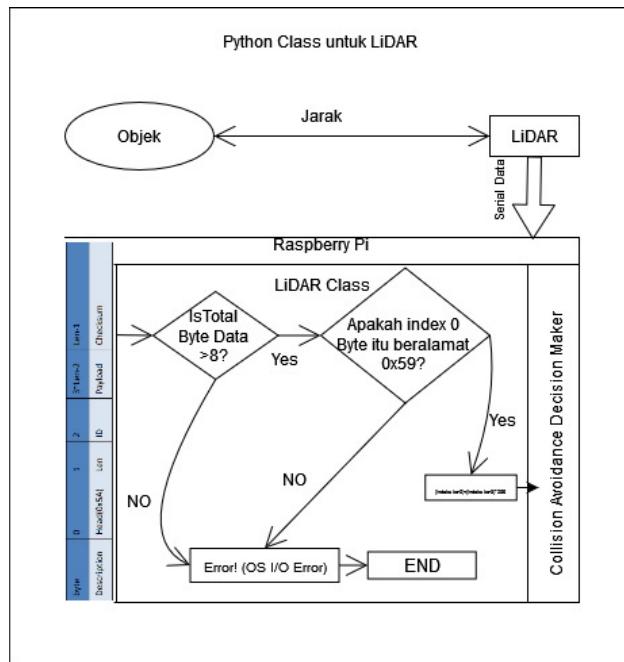
Gambar 4.6 Perintah *Python* untuk *import* dalam Sistem Program

Setelah berhasil proses *import* dari *Python Packages* itu maka perintah yang tersimpan dalam *package* itu bisa digunakan dalam program yang akan dibuat. Hal ini merupakan sifat

dari bahasa pemrograman *Python* yang disebut *Object Oriented Programming* (OOP), dimana kumpulan perintah dari sebuah program itu bisa digunakan oleh program lain apabila program yang dibuat itu bersifat *Class Object*. Sifat dari OOP juga terdapat:

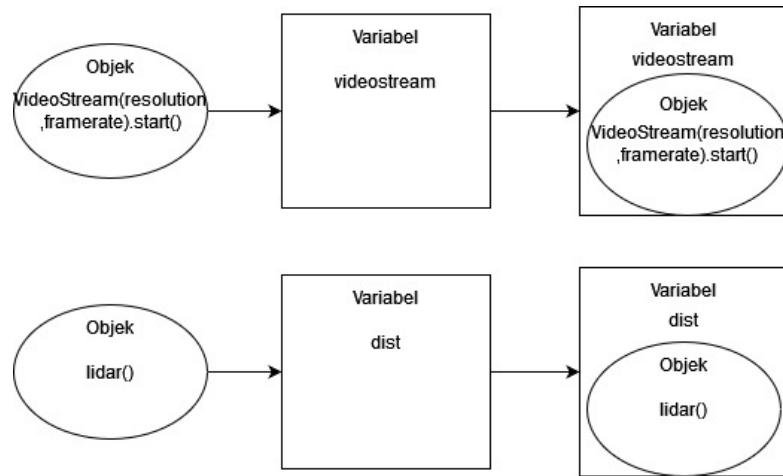
1. *Class* : Kumpulan dari *object*, dimana *object* ini harus memiliki atribut dan juga metode yang ingin dipakai dalam program. Contohnya adalah jika sebuah perusahaan memiliki kumpulan data karyawan maka dari data tersebut akan tersimpan atribut dan metode yang tercantum bersama dengan nama karyawan tersebut, contoh umur, dudukan dalam perusahaan dan tugasnya
2. *Object* : Segala sesuatu yang memiliki sebuah kondisi dan sifat, contoh di dunia nyata adalah bolpen, dimana kondisi bolpen terdapat dua yakni waktu tutup bolpen tertutup dan waktu tutup bolpen terbuka dan sifat dari bolpen itu adalah untuk menulis sesuatu jika kondisi tutup bolpen terbuka
3. *Method* : Sebuah fungsi yang dikaitkan dengan objek, dalam bahasa *Pyhton* sebuah *method* itu tidak unik kepada sebuah kelas maka segala jenis objek bisa memiliki sebuah *method*. Contohnya adalah pekerjaan seorang admin dalam sebuah perusahaan logistik dimana *method* yang diasosiasikan dengan pekerjaan *admin* adalah mengurutkan dan menata produk serta memberi perintah kepada kurir bahwa sebuah produk perlu dikirim ke alamat tertentu.
4. *Inheritance* : Aspek ini menjelaskan bahwa sebuah kelas bisa menurunkan sifat serta atribut dari kelas lain. Contoh di dunia nyata adalah sepasang orang tua yang menurunkan sifat gen kepada turunannya.
5. *Polymorphism* : *Poly* artinya banyak, *morph* itu artinya bentuk. Jadi sifat ini menjelaskan bahwa sebuah sifat itu bisa memiliki banyak bentuk. Contohnya adalah dengan sebuah *class* binatang, semua binatang berbicara tetapi berbicara dengan cara yang berbeda. Dalam contoh ini berbicara adalah sebuah sifat dari sebuah binatang namun aspek *polymorphnya* adalah bagaimana binatang itu berbicara.
6. *Data Abstraction*: Aspek yang dicapai dengan *encapsulation* agar detail dari cara kerja fungsi itu disembunyikan dan hanya menunjukkan fungsionalitasnya saja.
7. *Encapsulation* : Merupakan aspek yang menjelaskan membatasi *method* dan juga *variable* yang digunakan sebuah *object*, dalam *encapsulation* sebuah *code* dan data itu terselubungi menjadi satu unit agar tidak termodifikasi dengan tidak sengaja.

Dari aspek yang telah dijelaskan maka perbedaan sebuah OOP dengan *Procedural Programming* adalah OOP itu menggunakan *Object Class* serta *Method* yang tersimpan dalam *Object Class* itu untuk menyelesaikan sebuah masalah namun *procedural Programming* itu berarti metode penyelesaian masalah yang berisikan daftar instruksi yang perlu dikomputasikan satu per satu jadi lebih panjang, karena tidak ada abstraksi, dan lebih susah di pelihara/*Maintain* karena sifat dari instruksi ini harus sangat detail dan bisa berkembang menjadi instruksi yang sangat kompleks, contoh dari konsep OOP ini bisa dilihat pada Gambar 4.7.



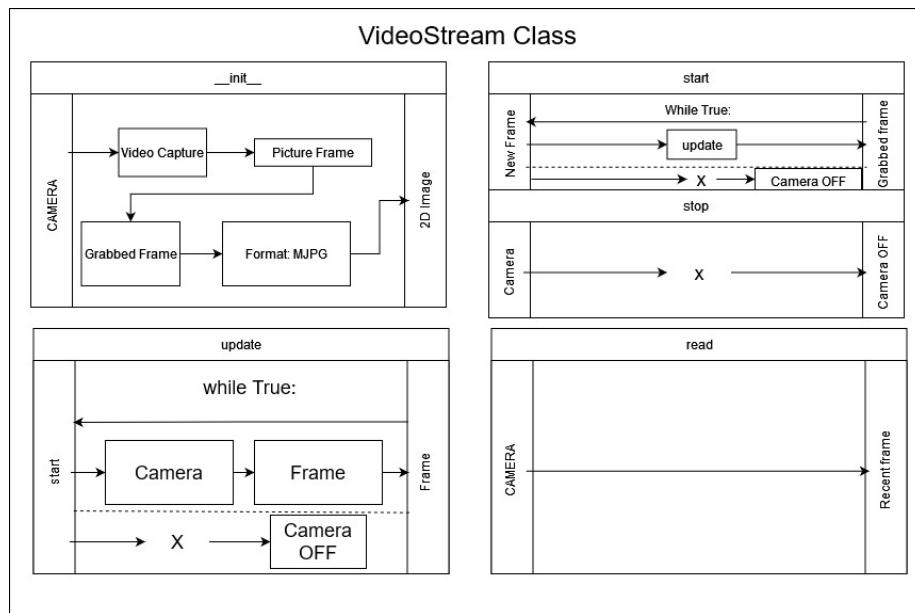
Gambar 4.7 Python Class untuk LiDAR

Setelah komponen di *import* dalam sebuah *environment* atau lingkungan dimana program akan dieksekusi maka selanjutnya adalah membuat sistem dengan menggabungkan konsep dari OOP, seperti yang terlihat pada Gambar 4.8. Dimana untuk mengaktifasi *Object Class* perlu di panggil nama *Classnya* dan menyimpan nama tersebut pada sebuah variable kosong atau bisa juga terdapat agurmen, sesuai dengan *class* yang dibuat sebelumnya, agar gampang digunakan dalam proses pembuatan sistem.



Gambar 4.8 Pemanggilan *Object* pada Sistem *Autonomous*

Setelah membuat objek dari *package* yang diperlukan maka diperlukannya cara untuk sistem menggunakan sensor berupa kamera, oleh karena itu diperlukannya *object class*. Bernama *VideoStream* untuk program dapat menerima video *real time* dari kamera yang terhubung pada *Raspberry Pi*, seperti yang terlihat pada Gambar 4.9 dan *code* pada lampiran.



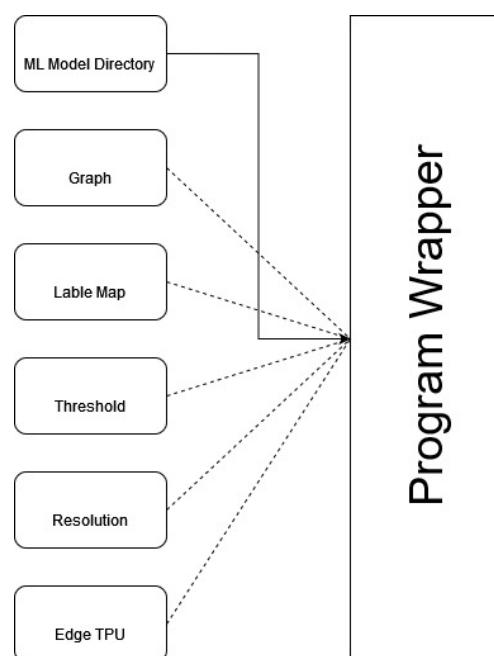
Gambar 4.9 *Object Class VideoStream*

Dalam *object class* *VideoStream* itu terdapat 5 (lima) fungsi agar bisa digunakan dalam sistem, yakni:

1. `__init__(self,resolution=(),framerate=())` : Untuk menginisiasi sensor kamera dan sambungan data gambar dari kamera
2. `Start(self)` : Untuk memulai proses *Threading*, untuk menjalankan beberapa proses secara parallel

3. Update (self) : Untuk menjalankan *infinite loop* sampai proses *threading* berhenti
4. Read(self) : Untuk membaca data *stream* dari kamera
5. Stop(self) : Untuk menghentikan operasi kamera

Karena sistem ini direncanakan untuk menerima model yang telah di latih dari pihak ke-3 dan belum tentu model memiliki nama serta perfoma yang sama, juga perlu diingat bahwa program belum tentu digunakan dengan sensor yang sama resolusinya atau menggunakan alat tambahan berupa *edgetpu*, alat untuk menambah kemampuan *threading* dari *processor Raspberry Pi*, maka program harus bisa menerima beberapa argument tambahan sebelum di eksekusi oleh *user*, seperti yang ditunjukan pada Gambar 4.10.



Gambar 4.10 Program menerima Argumen Tambahan sebelum Dieksekusi

Penggunaan argument tersebut bisa diimplementasikan dalam sistem dengan cara OOP yang disimpan dalam variable kosong yang bisa diakses semua fungsi dalam program yang dibuat. apabila sistem yang sudah disiapkan untuk implementasi model ML, maka Langkah berikutnya adalah untuk *import framerwork* yang akan digunakan untuk mengaplikasikan model yang ingin digunakan, dalam kasus ini adalah *framework TensorFlow Lite* untuk *microprocessor Raspberry Pi* dengan cara yang ditunjukan pada Gambar 4.11.

```

# Import TensorFlow libraries
# If tflite_runtime is installed, import interpreter from tflite_runtime, else
import tensorflow as tf
# If using Coral Edge TPU, import the load_delegate library
pkg = importlib.util.find_spec('tflite_runtime')
if pkg:
    from tflite_runtime.interpreter import Interpreter
    if use_TPU:
        from tflite_runtime.interpreter import load_delegate
else:
    from tensorflow.lite.python.interpreter import Interpreter
    if use_TPU:
        from tensorflow.lite.python.interpreter import load_delegate

# If using Edge TPU, assign filename for Edge TPU model
if use_TPU:
    # If user has specified the name of the .tflite file, use that name,
    # otherwise use default 'edgetpu.tflite'
    if (GRAPH_NAME == 'detect.tflite'):
        GRAPH_NAME = 'edgetpu.tflite'

# Get path to current working directory
CWD_PATH = os.getcwd()

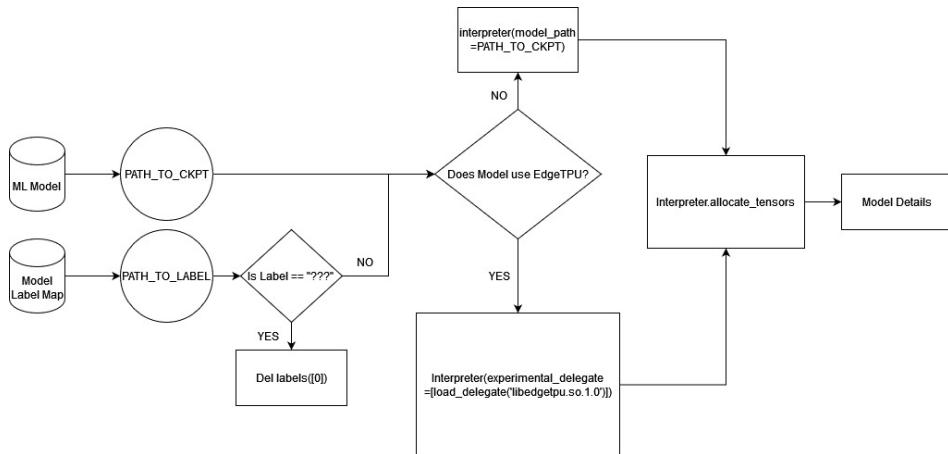
# Path to .tflite file, which contains the model that is used for object
# detection
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)

# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)

```

Gambar 4.11 Cara Setup Framework TensorFlow Lite untuk Membaca Model ML

Setelah *framework* berhasil di *install* pada *Raspberry Pi* maka Langkah berikutnya adalah memasukan model ML yang ingin digunakan dengan cara yang ditunjukan pada Gambar 4.12.



Gambar 4.12 Cara Import Model ML yang Sudah di Pre-Trained

Setelah model ML telah berhasil dimasukkan kedalam sistem maka Langkah berikutnya adalah menguji model dengan menyalakan sensor kamera dengan sistem *Object Detection* dengan perintah yang bisa dilihat pada Gambar 4.13.

```

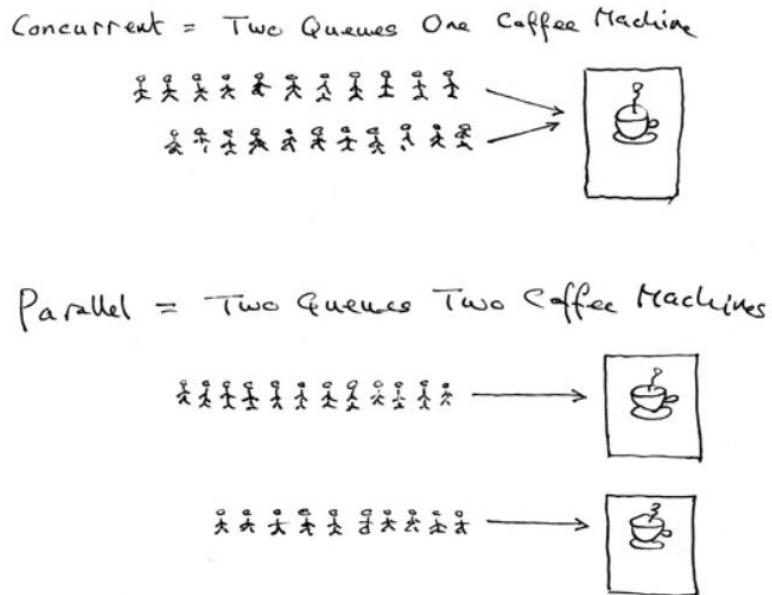
# Initialize video stream
videostream = VideoStream(resolution=(imW,imH),framerate=30).start()
time.sleep(1)

```

Gambar 4.13 Perintah untuk Menyalakan Kamera dalam Program Sistem Object Detection

Agar fungsi kamera dan LiDAR bisa diintergrasikan, maka perlu diimplementasikan konsep *concurrent processing*. *Concurrent processing* adalah konsep pemrograman yang

berarti program akan menjalankan kedua fungsi bersamaan, namun berbeda dengan parallelisme. Perbedaan kedua proses tersebut adalah penggunaan sumber daya CPU, dimana proses untuk proses parallel itu memerlukan beberapa *core* pada *processor* untuk eksekusi fungsinya, tetapi untuk proses *concurrent* itu bisa dilakukan dengan 1 *core processor* untuk mencapai hasil yang sama, untuk melihat proses dari proses ini bisa dilihat pada Gambar 4.14.



Gambar 4.14 Perbedaan Proses *Concurrency* dan *Parallel*

(Sumber: stackoverflow.com)

Dimana, maksud dari Gambar 4.14 itu berarti dengan proses *concurrent*. Sumber daya yang digunakan untuk menjalankan proses bisa dikurangi sehingga beban yang diterima pada CPU *raspberry pi* bisa berkurang dan program bisa dijalankan dengan menjadi lebih efisien dan sensor kamera serta LiDAR bisa terintegrasi, serta dengan penambahan jedah pembacaan sensor LiDAR selama 0.005 detik, seperti yang bisa dilihat pada Gambar 4.15, untuk menghindari antrian proses yang bertabrakan maka program mampu berjalan dengan sedikit dan hampir tidak ada gangguan pada *background process Operating System*.

```
#Continously update read data from LiDAR
distance = dist.read_tfluna_data()
time.sleep(0.005) #give time for concurrent process
```

Gambar 4.15 Fungsi Jedah Pembacaan LiDAR

Untuk *user* bisa melihat apakah sistem ini berjalan sesuai dengan yang diharapkan maka perlu dibuatnya *graphical user interface* (GUI) untuk bisa memvisualisasikan jalan program, perintah untuk membuat GUI ini bisa dilihat pada Gambar 4.16.

```
# Create window
cv2.namedWindow('Object detector', cv2.WINDOW_NORMAL)
```

Gambar 4.16 Perintah untuk Membuat GUI Sederhana

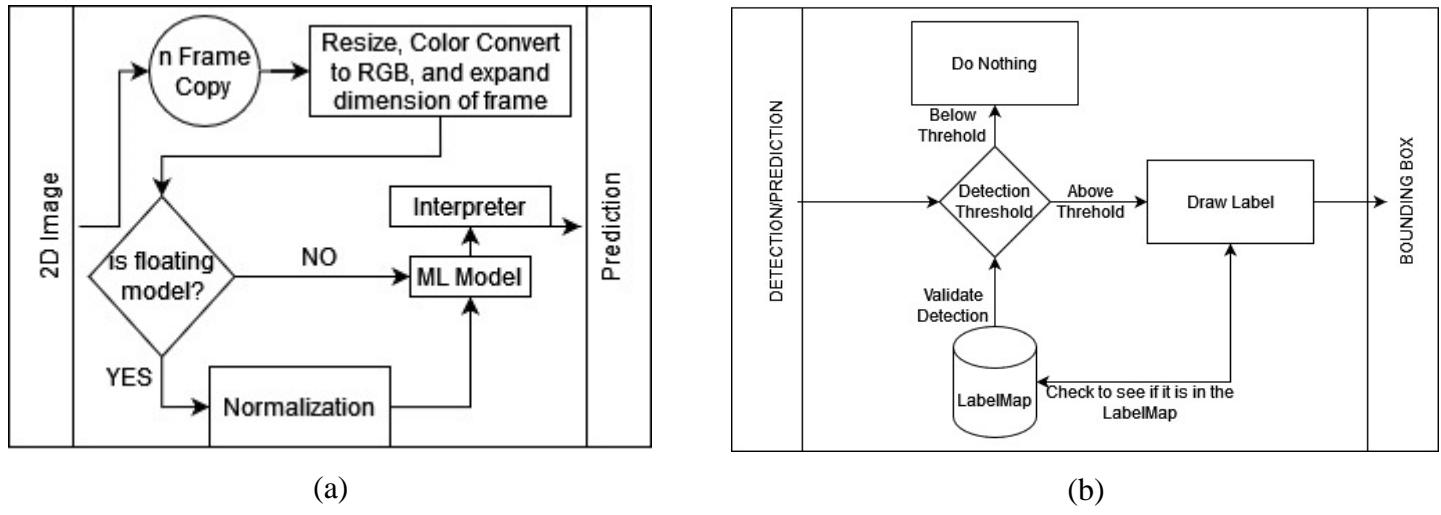
Apabila GUI sederhana telah dibuat maka tampilannya akan seperti pada Gambar 4.17.



Gambar 4.17 Graphical User Interface dari Object Detector

Setelah memastikan program bisa menampilkan gambar dari kamera dan pembacaan LiDAR tanpa masalah, maka berikutnya perlu dilakukan *looping* agar data berupa video yang masuk dari kamera bisa di prosess oleh model ML dan prediksi *Object Detection* bisa dilakukan secara *real-time* menggunakan program *looping* yang bisa dilihat pada Gambar 4.18.

Dimana program tersebut akan di *loop* secara *while True* atau akan *looping* hingga program diberhentikan oleh *user*, kerja program akan menerima gambar 2D dimana untuk tiap *frame* gamabr 2D itu akan dilakukan prediksi, dimana untuk tiap prediksi akan diukur dengan tingkat keyakinan atau *confidence level* dalam bentuk persentase (%) dan jika tingkat keyakinan itu melebihi batas yang telah disesuaikan untuk model maka objek tersebut akan terdeteksi, untuk mengetahui apakah ada objek yang terdapat pada *labelmap* model, lalu jika ada maka program akan mengeluarkan *label* dan juga *bounding box* pada area yang dianggap objek oleh model tersebut.



Gambar 4.18 (a)Proses *Looping* untuk Menampilkan Hasil Prediksi Model ML (b) Perintah untuk Menggambarkan *Boundix Box* dari Objek Prediksi

Di dalam loop ini juga dimana sistem *collision avoidance* dengan memasukan dan memproses hasil nilai dari LiDAR seperti yang ditunjukan pada Gambar 4.19.

```

#Funct to read lidar data
def read_tfluna_data(self):
    while True:
        counter = ser.in_waiting # count the number of bytes of the serial
        port
        if counter > 8:
            bytes_serial = ser.read(9) # read 9 bytes
            ser.reset_input_buffer() # reset buffer
            if bytes_serial[0] == 0x59 and bytes_serial[1] == 0x59: # check
                first two bytes
                distance = bytes_serial[2] + bytes_serial[3]*256 # distance
                in next two bytes
            return distance
  
```

(a)

```

while True:
    # Start timer (for calculating frame rate)
    t1 = cv2.getTickCount()

    #Continuously update read data from LiDAR
    distance = dist.read_tfluna_data()
    time.sleep(0.005) #give time for concurrent process
  
```

(b)

Gambar 4.19 (a) Fungsi untuk Membaca *Serial* Data LiDAR dan (b) Untuk Menampilkan Data tersebut secara Terus Menerus

Untuk perintah *decision makingnya* itu sesuai dengan kondisi yang sudah ditentukan di awal menggunakan metode *Blocking Area Method* berdasarkan perhitungan yang bisa dilihat hasilnya pada Gambar 4.20.

Model Kapal: (asumsi 100 kali dari skala				Log(Ad)	Log(Dt)	Ad/L	Dt/L
Loa	60	m	skala	0.54	0.59	1.71	1.80
Bm	30	m	100	0.61	0.70	1.84	2.02
H	14	m		0.65	0.75	1.91	2.12
T	12	m					

Kapal Nyata

speed (knot)	Speed (m/s)	Ad	Dt	T90	Rbf (m)	Rba (m)	S (m)	Note	Froude number kapal	Fn Model
17.010109	8.75	102.67	108.24	8.89	293.28	137.76	138.24	Arus Tidak Stabil	0.360659955	0.36
27.216174	14	110.48	120.95	6.03	313.15	144.38	150.95	Arus Tidak Stabil	0.577055928	0.58
34.020218	17.5	114.39	127.50	5.01	323.21	147.74	157.50	Arus Stabil	0.72131991	0.72

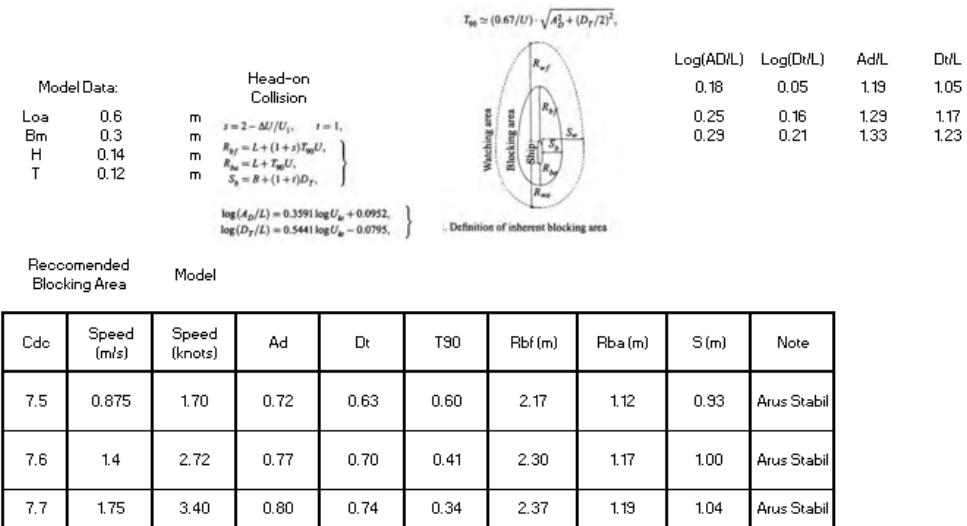
Gambar 4.20 Perhitungan *Blocking Area* untuk Kapal Nyata

Model kapal yang akan diuji akan menjadi representasi kapal yang memiliki ukuran utama seperti yang ditunjukkan pada Tabel 4-1. Dimana dari contoh kapal itu akan di skala 1:100 menjadi dimensi model agar bisa di uji coba, dimana hasil perhitungan *blocking area* untuk model bisa dilihat pada Gambar 4.21.

Tabel 4-1 *Main Dimension Kapal Contoh*

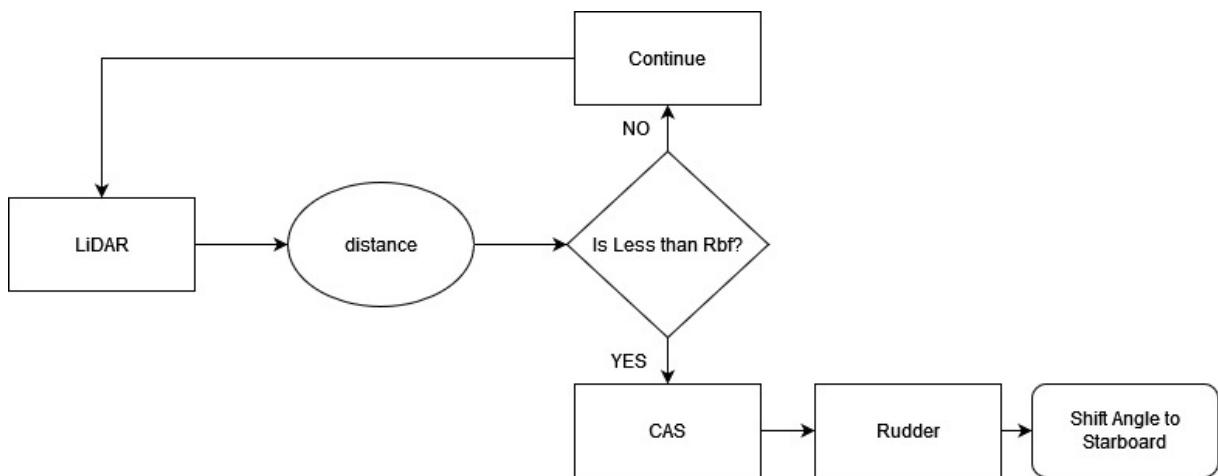
Main Dimension Kapal Contoh		
Loa	60	m
Bm	30	m
H	14	m
T	12	m

Sehingga dari perhitungan skala contoh kapal asli menjadi ukuran model, bisa ditentukan variasi kecepatan yang akan diuji. Yakni *Froude Number* (Fn) 0.36, 0.58 dan 0.72.



Gambar 4.21 Perhitungan *Blocking Area* untuk Model

Dimana dari perhitungan estimasi itu akan dijadikan perintah kondisional dengan faktor keamanan 40% untuk antisipasi faktor lingkungan, seperti angin kencang, hambatan kapal kecil/besar, manuver kapal terganggu karena faktor lain dan seterusnya. Sehingga bentuk perintahnya bisa dilihat di Gambar 4.22. Dimana selama jarak yang dibaca melebihi 1.2 m atau 120 cm, maka kapal akan tetap melaju. Jika kondisi terpenuhi maka kapal akan melakukan manuver penghindaran sesuai yang dijelaskan pada COLREGs *rule 14*.



Gambar 4.22 Sistem Kondisi *Collision Avoidance*

4.5. Trial Darat

Setelah *Wrapper Code/ Decision Maker* itu telah dibuat maka perlu di uji coba terlebih dahulu sistemnya di darat, apakah sistem yang dirancang ini bekerja sebagaimana mestinya atau perlu di modifikasi lagi sebelum pengujian di air Bersama dengan model kapal yang telah

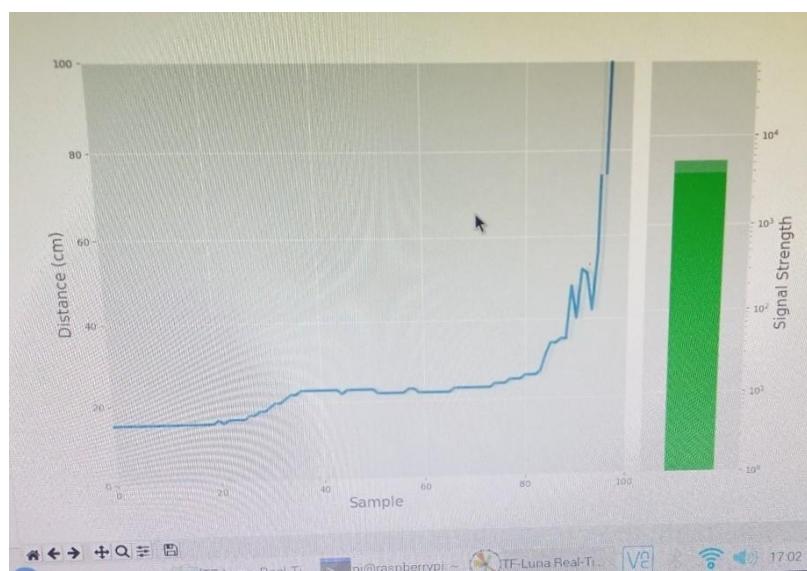
disiapkan. Dimana pada tahap ini akan di ujicoba program dengan 2 variasi uji coba, dengan kompleksitas berikut:

1. Jarak kapal dengan benda (jarak penuh dan kurang dari penuh)
2. Kecepatan kapal dengan merubah nilai CDC pada ESC

Dari 2 variasi pengujian itu akan dilakukan sebanyak minimal 20 kali dimana akan menghasilkan data di darat sebanyak minimal 60 data kasus yang perlu di proses dengan metode statistika. Dengan mengetahui batasan kapabilitas dari program yang dibuat di darat maka pengujian di air bisa disesuaikan dengan hasil uji darat.

4.5.1. Pengujian LiDAR

Dalam percobaan darat, sistem yang diuji adalah fungsi dari sensor, berupa LiDAR dan *onboard Camera*. Dimana dalam pengujian sensor LiDAR akan diuji dengan mengukur jarak benda dengan jarak interval yang akan di validasikan dengan pengukuran menggunakan meteran yang satuan ukurnya menggunakan standar internasional, seperti yang bisa dilihat pada Gambar 4.24. Untuk mengetahui apakah sensor mengukur dengan program yang dibuat pada Subab **Error! Reference source not found.** maka perlu di visualisasikan data pengukuran dalam bentuk grafik pengukuran di dalam *microprocessor* seperti yang ada pada Pengujian ini menghasilkan data dari LiDARseperti yang ditunjukan pada Gambar 4.25.

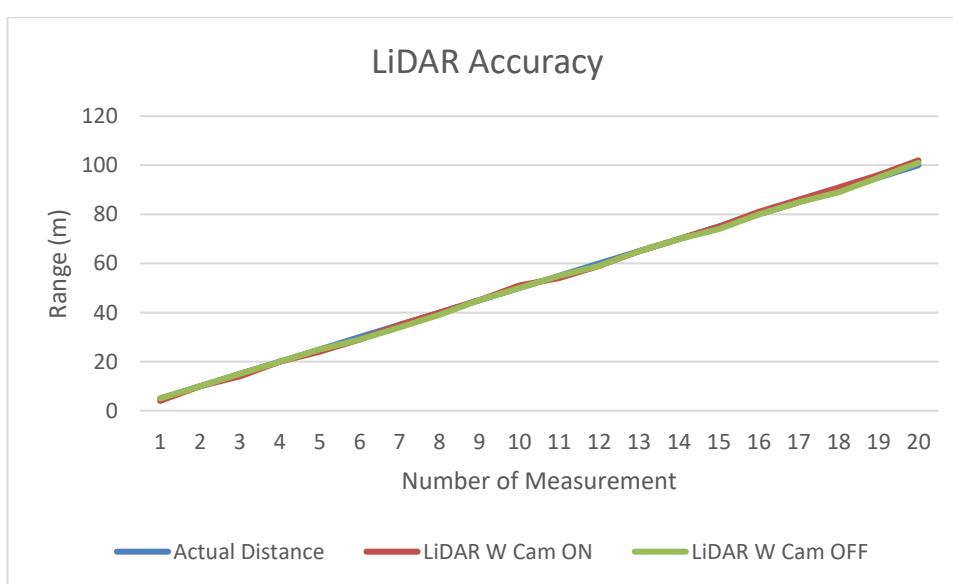


Gambar 4.23 GUI Pengukuran dengan LiDAR pada *Raspberry Pi*



Gambar 4.24 Uji Pengukuran Sensor LiDAR dengan Meteran

Diketahui bahwa dari pengambilan data LiDAR menghasilkan perbedaan pengukuran LiDAR sebesar ± 0.45 cm atau *margin of error* sebesar 2.67% dari posisi asli benda yang ingin diukur. Salah satu faktor mengapa terjadinya kesalahan pengukuran dengan LiDAR adalah faktor lingkungan yang mengganggu pemancaran dan penerimaan cahaya infra merah dari LiDAR, karena untuk mengukur jarak LiDAR memiliki komponen yang sensitif terhadap cahaya untuk menerima pantulan balik cahaya inframerah dan karena komponen ini sensitif terhadap cahaya maka ada kemungkinan dikarenakan cahaya dari matahari ditangkap oleh LiDAR sebagai data *noise* sehingga mengganggu pengukuran sensor.



Gambar 4.25 Akurasi Pembacaan LiDAR

Dari 20 kali pengukuran dengan interval 5 cm itu diketahui bahwa untuk jarak yang lebih dari 1 m sensor masih bisa digunakan karena pada jarak tersebut pengukuran LiDAR pada jarak tersebut <5% apabila sensor digunakan bersamaan dengan sistem kamera ML. Seperti yang bisa dilihat pada Tabel 4-2.

Tabel 4-2 Pemeriksaan Akurasi LiDAR

Actual Distance (cm)	LiDAR Reading (cm)		Difference		Margin of error Camera ON	Margin of error Camera OFF
	w camera ON	w camera OFF	w camera ON	w camera OFF		
5	4	5	1	0	20.00%	0.00%
10	10	10	0	0	0.00%	0.00%
15	14	15	1	0	6.67%	0.00%
20	20	20	0	0	0.00%	0.00%
25	24	25	1	0	4.00%	0.00%
30	29	29	1	1	3.33%	3.33%
35	35	34	0	1	0.00%	2.86%
40	40	39	0	1	0.00%	2.50%
45	45	45	0	0	0.00%	0.00%
50	51	50	1	0	2.00%	0.00%
55	54	55	1	0	1.82%	0.00%
60	59	59	1	1	1.67%	1.67%
65	65	65	0	0	0.00%	0.00%
70	70	70	0	0	0.00%	0.00%
75	75	74	0	1	0.00%	1.33%
80	81	80	1	0	1.25%	0.00%
85	86	85	1	0	1.18%	0.00%
90	91	89	1	1	1.11%	1.11%
95	96	95	1	0	1.05%	0.00%
100	102	101	2	1	2.00%	1.00%
Mean difference (cm):		0.5	0.375			
	Reading error (%):	4.25%	1.09%			
	Accuracy:	95.75%	98.91%			

4.5.2. Pengujian Servo

Selain pengujian sensor, perlu juga dilakukan pengujian sistem *actuator* sebelum di model di uji di air untuk mencari konfigurasi elektronik yang tepat di dalam program, terutama untuk konfigurasi *ChangeDutyCycle* (CDC) nya *servo* untuk mengendalikan *rudder* kapal. Untuk menguji servo, maka perlu di buat program penguji dimana akan divariasikan nilai dari CDCnya dan untuk melihat perubahan sudut dari *rudder* akan diperlukan busur dan tusuk gigi. Dimana poros *rudder* akan di pasang tusuk gigi yang ujungnya ditandai dengan tinta dari spidol

yang berwarna, seperti yang bisa dilihat pada Gambar 4.26, dan untuk setiap variasi nilai CDC akan dilihat perubahan sudut pada poros *rudder*.



Gambar 4.26 Pengujian Konfigurasi *Rudder*

Dari pengujian, maka didapat nilai variasi yang bisa dilihat pada Tabel 4-3. Dimana diketahui, bahwa untuk merubah arah *rudder* kapal, diperlukan nilai CDC minimal 7 untuk menghasilkan sudut 80° ke arah *port*, seperti yang bisa dilihat pada , untuk mengarahkan *rudder* pada posisi netral 90° diperlukan nilai CDC 8, dan dari pengujian *actuator* ini diketahui juga untuk menggerakkan *rudder* ke arah *starboard* sebesar 10° , seperti yang bisa dilihat pada Gambar 4.29, memperlukan nilai CDC 9.



Gambar 4.27 Perubahan *Rudder* ke Arah *Ports*

Tabel 4-3 Konfigurasi Servo

Rudder Function Test		
No.	Change Duty Cycle	Rudder Angle (Degrees)
1	7	80
2	7.1	81
3	7.2	82
4	7.3	83
5	7.4	84
6	7.5	85
7	7.6	86
8	7.7	87
9	7.8	88
10	7.9	89
11	8	90
12	8.1	91
13	8.2	92
14	8.3	93
15	8.4	94
16	8.5	95
17	8.6	96
18	8.7	97
19	8.8	98
20	8.9	99
21	9	100

<-- Rudder Minimal Angle

<-- Rudder Neutral Angle

<-- Rudder Maximal Angle

Apabila diperhatikan pada Tabel 4-3, maka akan diketahui bahwa untuk merubah *rudder* sejauh 1° , seperti yang bisa dilihat pada dari posisi awalnya maka diperlukan penambahan atau pengurangan nilai CDC sebanyak 0.1.



Gambar 4.28 Perubahan Sudut Rudder Sebanyak 1 Derajat



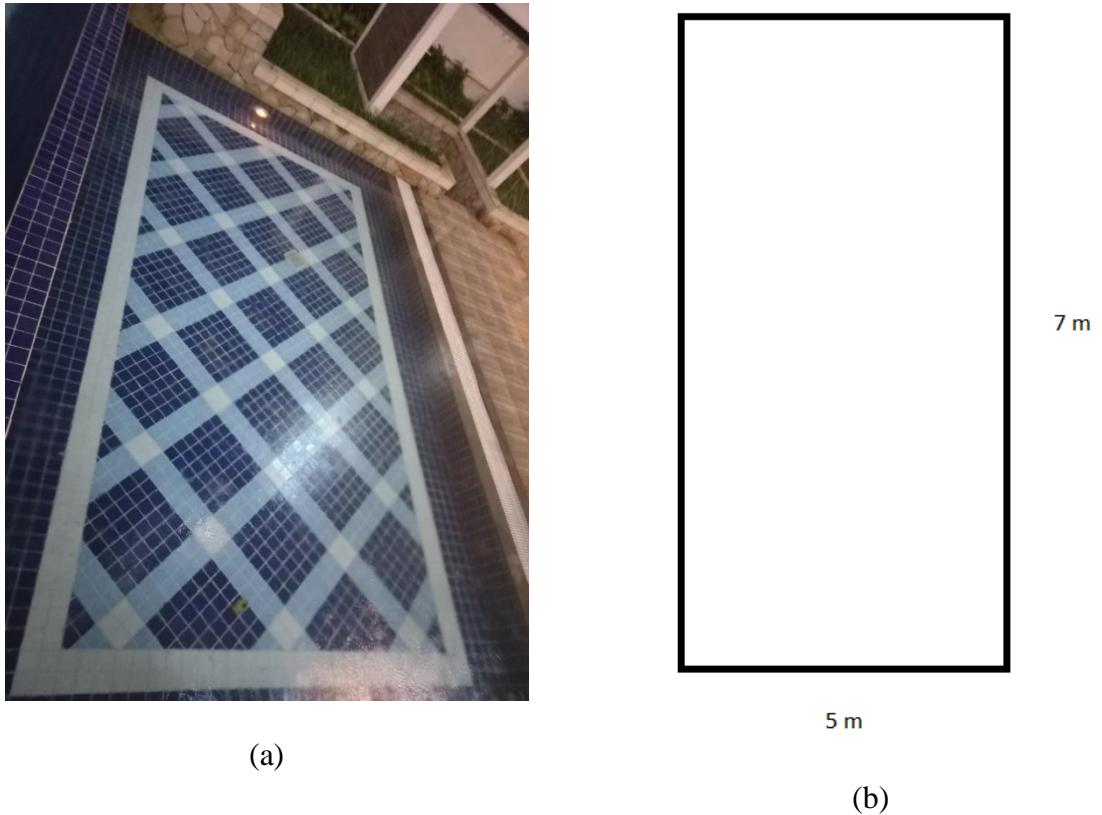
Gambar 4.29 Perubahan Rudder ke Arah Starboard

4.6. Trial Air

Apabila scenario yang diinginkan telah ditentukan maka kapal akan di uji coba dengan Prototipe kapal lain yang akan digunakan sebagai simulasi di lapangan, dari pengujian itu akan dilakukan sebanyak minimal 10 kali dimana akan menghasilkan data di air sebanyak minimal 10 data per variasi yang perlu di proses dengan metode statistika. Dalam proses percobaan di air ini akan dilakukan pada kolam sehingga lingkungan percobaan terkendali. Hasil dari percobaan air ini akan didapat beberapa data, yakni:

1. Data perfoma dari prototipe model kapal yang meliputi:
 - a. Kecepatan prototipe
 - b. Kemampuan manuverabilitas prototipe
2. *Success Rate* dari sistem itu yang berati berapa kalinya prototipe berhasil menghindari objek yang telah ditentukan, dan tingkat akurasi sistem deteksi objek sistem.
3. *Trajectory* dan juga sudut *heading* kapal

Untuk menguji perfoma, perfoma dalam arti kecepatan model kapal, maka kapal akan diuji di kolam yang sudah diukur dimensinya, seperti yang bisa dilihat pada Gambar 2.32.



Gambar 4.30 (a) Kolam Uji Kecepatan (b) Ilustrasi Ukuran Kolam Uji

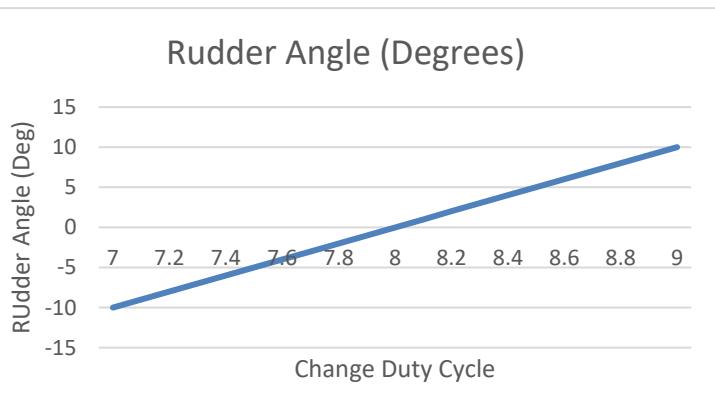
Apabila dimensi kolam uji sudah terukur, maka langkah selanjutnya adalah untuk menguji kapal dengan variasi nilai CDC, seperti pengujian *servo*, dimana kapal akan diberi input CDC dan akan diukur waktu yang diperlukan kapal untuk bergerak dari ujung ke ujung kolam uji dan dari waktu itu akan dibagi dengan panjang kolam uji sehingga akan didapat kecepatan kapal tersebut dan dicatat untuk tiap variasi nilai CDCnya dalam bentuk data tabular, seperti yang bisa dilihat pada Tabel 4-4.

Tabel 4-4 Data Kecepatan Kapal dengan Nilai CDC

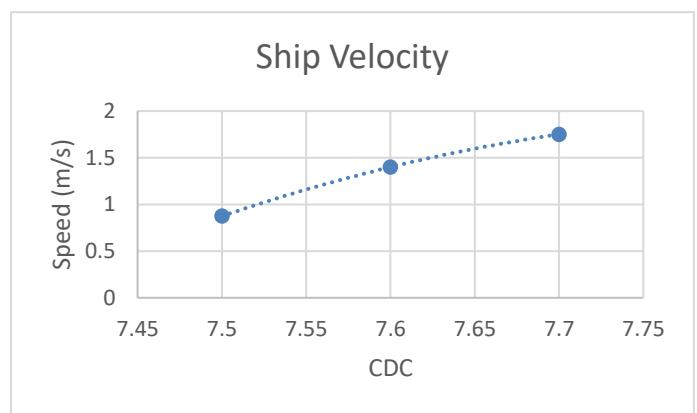
Engine Function Test				Panjang Kolam = 7m
No.	Cdc	Speed (m/s)	Speed (Knots)	Time (s)
1	7.5	0.875	1.70086	8
2	7.6	1.4	2.721376	5
3	7.7	1.75	3.40172	4

Setelah percobaan air itu maka bisa diambil kesimpulan dari percobaan apabila kesuksesan dari sistem itu melebihi 95% namun perlu pengembangan ulang untuk aplikasi di lingkungan yang dinamis, seperti di laut terbuka atau air yang tidak tenang.

Kapal yang di uji coba di kolam, pertama perlu di dapatkan kemampuan actuator yang *onboard*, sudut Servo dan juga *power* dari Motor, karena *datasheet* yang tersedia untuk *actuator* itu diuji di darat sehingga tidak terhitungkan gaya hambatan viskositas dari air. Data yang didapat untuk *actuator* ini divariasikan terhadap inputan perintah berupa *ChangeDutyCycle* (CDC) untuk mengendalikan gerakan dari *actuator*. CDC bekerja dengan mengubah lebar dari gelombang signal listrik yang dikirim dari *microprocessor* ke elektronik, metode ini disebut juga dengan *Pulse Width Modulation* (PWM). Data yang didapat di sajikan dalam bentuk grafik garis, seperti yang di lihat pada Gambar 4.31.



(a)

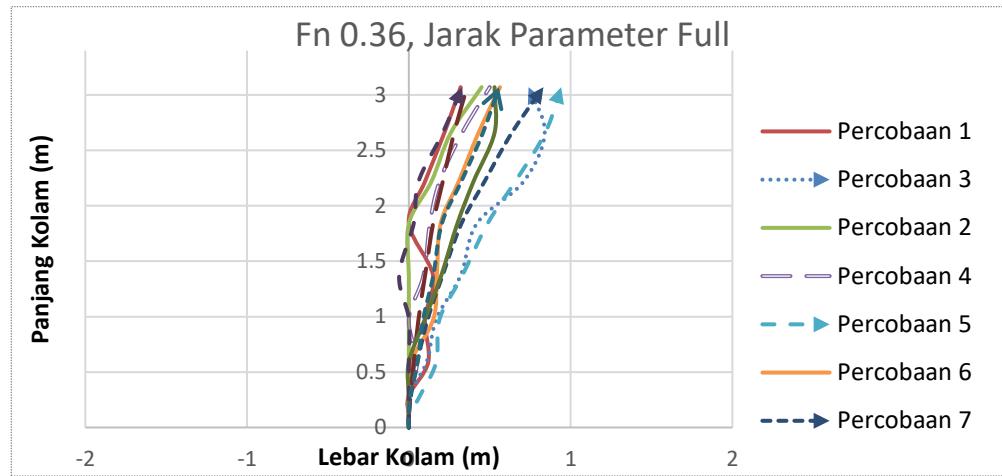


(b)

Gambar 4.31 (a) Grafik Sudut *Rudder* dari Gerakan Servo. (b) Grafik Kecepatan Kapal Terhadap Perubahan CDC.

4.6.1. Variasi Fn 0.36 dengan Parameter Jarak Penuh

Dengan sudut konstan maksimal rudder ke arah starboard sejauh 10^0 maka kapal akan membentuk lintasan penghindaran elips seperti yang dilihat pada Gambar 4.32 untuk melihat kecendrungan gerakan kapal pada variasi *forude number* 0.36.



Gambar 4.32 Bentuk Lintasan Variasi Fn 0.36- Jarak Parameter Penuh

Dari 10 kali percobaan, maka didapat lintasan kapal yang ditunjukkan pada Gambar 4.32. Dilihat bahwa eksperimen *collision avoidance* dengan perhitungan *blocking area*, dimana untuk variasi Fn 0.36, akan diperlukan jarak +/-2.17 meter dari kapal. Sehingga untuk menguji kesuksesan dan validasi perhitungan akan diukur terlebih dahulu batas acuan R_{bf} 2.17 meter dari objek yang akan dihindar, lalu kapal akan diuji pada jarak yang lebih dari perhitungan R_{bf} sehingga dapat dipantau kinerja sistem dan mendapat sudut *heading* kapal, untuk variasi ini bisa dilihat pada Tabel 4-5. Dimana disimpulkan bahwa rata- rata sudut *heading* kapal adalah 12.6°

Tabel 4-5 Sudut *Heading* Kapal untuk Fn 0.36 dengan Jarak Parameter Penuh

Variasi Fn 0.36 -1		
n	Sudut Heading	AVG (Deg)
1	10.8	12.6
2	9.5	
3	17.9	
4	9.5	
5	19.8	
6	12.6	
7	18.8	
8	8.2	
9	7.9	
10	10.8	

Visualisasi grafis pada Gambar 4.32, didapat dari hasil proses data berupa video mentah yang di ambil dari kamera *handphone* yang lalu dimasukkan dalam *software blender* dan *fspy*, dimana hasil penangkapan video itu akan di rubah menjadi model 3D oleh *fspy* dan kemudian akan di cari koordinat-koordinat tiap satuan waktu dengan menggunakan *software blender*.

Contoh dari hasil pencatatan data dalam bentuk koordinat bisa dilihat pada Tabel 4-6 untuk hasil data 5 percobaan pertama dan Tabel 4-7 untuk 5 percobaan terakhir.

Tabel 4-6 Hasil Data Koordinat dari Video Trial Air Fn 0.36-Jarak Parameter Penuh

f	1		2		3		4		5	
	x	y	x	y	x	y	x	y	x	y
10	0	0	0	0	0	0	0	0	0	0
20	0	0.01	0	0.01	0	0.01	0	0.01	0	0.01
30	0	0.1	0	0.1	0	0.1	0	0.1	0	0.1
40	0	0.29	0	0.29	0	0.29	0	0.29	0.07	0.29
50	0.12	0.59	0	0.59	0.11	0.59	0.05	0.59	0.168995	0.59
60	0.1	0.98	0	0.98	0.17	0.98	0	0.98	0.190239	0.98
70	0.15	1.37	0	1.37	0.32	1.37	0.09	1.37	0.330239	1.37
80	0	1.84	0	1.84	0.42	1.84	0.13	1.84	0.486763	1.84
90	0.1	2.22	0.14	2.22	0.71	2.22	0.19	2.22	0.658228	2.22
100	0.22	2.66	0.26	2.66	0.84	2.66	0.33	2.66	0.84343	2.66

Tabel 4-7 Hasil Data Koordinat dari Video Trial Air Fn 0.36-Jarak Parameter 75%

f	6		7		8		9		10	
	x	y	x	y	x	y	x	y	x	y
10	0	0	0	0	0	0	0	0	0	0
20	0	0.01	0	0.01	0	0.01	0	0.01	0	0.01
30	0	0.1	0	0.1	0	0.1	0	0.1	0	0.1
40	0	0.29	0.013	0.29	0.01	0.29	0	0.29	0	0.29
50	0.012	0.59	0.052	0.59	0.03	0.59	0	0.59	0	0.59
60	0.152	0.98	0.117	0.98	0.06	0.98	0.1	0.98	0	0.98
70	0.172785	1.37	0.208	1.37	0.1	1.37	0.2	1.37	-0.06	1.37
80	0.196785	1.84	0.325	1.84	0.15	1.84	0.3	1.84	0.03	1.84
90	0.308588	2.22	0.468	2.22	0.21	2.22	0.4	2.22	0.06	2.22
100	0.431062	2.66	0.637	2.66	0.28	2.66	0.53	2.66	0.21	2.66

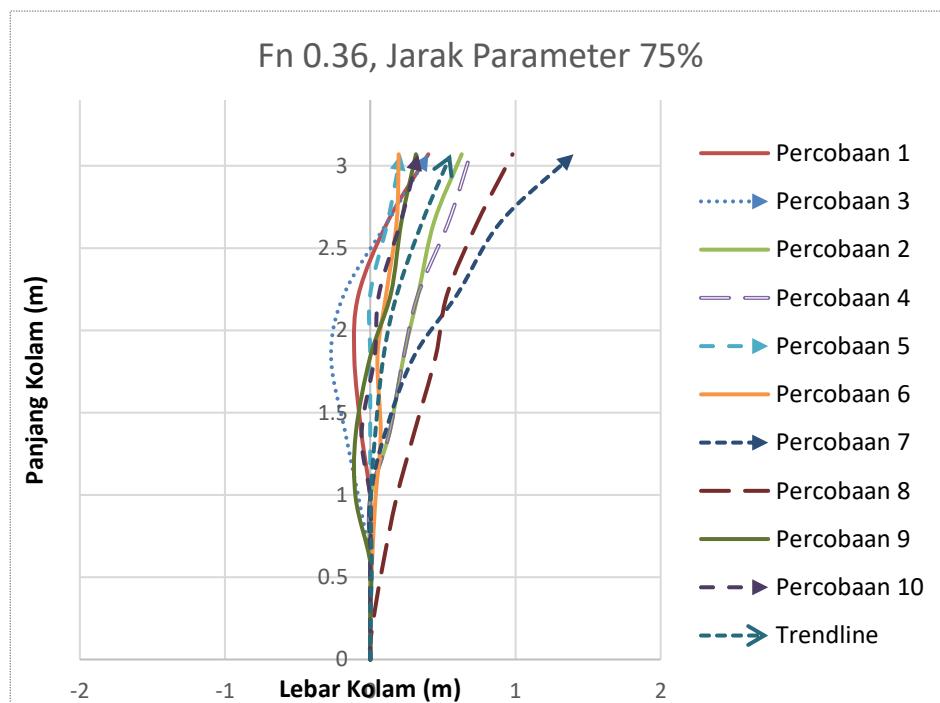
Dari Gambar 4.32, bisa dilihat bahwa kapal akan mulai berbelok pada jarak +/- 1.35 m. hal ini dikarenakan jarak awal antar kapal dan benda itu +/- 3.6 m, sehingga jika di setting parameter R_{bf} menjadi 2.17 m maka secara otomatis kapal akan bereaksi terhadap deteksi objek yang memasuki area kurang dari 2.17 m ini. Sehingga dari jarak awal 3.6 m itu kapal seharusnya bereaksi sekitar +/- 1.43 m, namun realitanya kapal bereaksi pada jarak +/- 1.35 m hal ini dikarenakan beberapa faktor yang mungkin mempengaruhi pengukuran dan telatnya reaksi sistem. Yakni:

1. Gangguan cahaya matahari, LiDAR berkerja dengan menembak cahaya infra merah dan mengukur jarak dari selang waktu pantulan cahaya itu kembali ke sensor. Cara pengukuran menggunakan cahaya ini memiliki kekurangan, yakni tidak cocok untuk cuaca buruk, seperti hujan dan kabut, karena cahaya dari LiDAR bisa terdeviasi oleh partikel air.

2. Karena cara kerja sensor TF-Luna mengirim data dalam bentuk *serial*, maka perlu waktu yang sedikit lebih lama untuk membaca pengukuran yang dijalankan bersamaan dengan proses deteksi objek dari model ML.

4.6.2. Variasi Fn 0.36 dengan 75% Parameter Jarak

Dengan sudut konstan maksimal rudder ke arah starboard sejauh 10° maka kapal akan membentuk lintasan penghindaran elips seperti yang dilihat pada Gambar 4.33 untuk melihat kecendrungan gerakan kapal pada variasi *forude number* 0.36.



Gambar 4.33 Bentuk Lintasan Fn 0.36-75% Jarak Parameter

Dari 10 kali percobaan, maka didapat lintasan kapal yang ditunjukkan pada Gambar 4.33. Dilihat bahwa eksperimen *collision avoidance* dengan perhitungan *blocking area*, dimana untuk variasi Fn 0.36, akan diperlukan jarak ± 1.63 meter dari kapal. Sehingga untuk menguji kesuksesan dan validasi perhitungan akan diukur terlebih dahulu batas acuan R_{bf} 1.63 meter dari objek yang akan dihindar, lalu kapal akan diuji pada jarak yang lebih dari perhitungan R_{bf} sehingga dapat dipantau kinerja sistem dan mendapat sudut *heading* kapal, untuk variasi ini bisa dilihat pada Tabel 4-8. Dimana disimpulkan bahwa rata-rata sudut *heading* kapal adalah 14.5°

Tabel 4-8 Sudut Heading Kapal untuk Fn 0.36 dengan 75% Jarak Parameter

Variasi 0.36-0.75		
n	Sudut Heading	AVG (Deg)
1	14.8	14.5
2	14.8	
3	15.1	
4	12.3	
5	7.9	
6	6.0	
7	32.2	
8	19.5	
9	11.6	
10	10.8	

Visualisasi grafis pada Gambar 4.33, didapat dari hasil proses data berupa video mentah yang di ambil dari kamera *handphone* yang lalu dimasukkan dalam *software blender* dan *fspy*, dimana hasil penangkapan video itu akan di rubah menjadi model 3D oleh *fspy* dan kemudian akan di cari koordinat-koordinat tiap satuan waktu dengan menggunakan *software blender*. Contoh dari hasil pencatatan data dalam bentuk koordinat bisa dilihat pada Tabel 4-9 untuk hasil data 5 percobaan pertama dan Tabel 4-10 untuk 5 percobaan terakhir.

Tabel 4-9 Hasil Data Koordinat dari Video Trial Air Fn 0.36-75% Jarak Parameter Penuh

f	1		2		3		4		5	
	x	y	x	y	x	y	x	y	x	y
10	0	0	0	0	0	0	0	0	0	0
20	0	0.01	0	0.01	0	0.01	0	0.01	0	0.01
30	0	0.1	0	0.1	0	0.1	0	0.1	0	0.1
40	0	0.29	0	0.29	0	0.29	0	0.29	0	0.29
50	0	0.59	0	0.59	0	0.59	0	0.59	0	0.59
60	0	0.98	0	0.98	-0.08	0.98	0	0.98	0	0.98
70	-0.06	1.37	0.13	1.37	-0.17	1.37	0.13	1.37	0	1.37
80	-0.11	1.84	0.23	1.84	-0.27	1.84	0.23	1.84	0	1.84
90	-0.08	2.22	0.33	2.22	-0.18	2.22	0.33	2.22	0	2.22
100	0.12	2.66	0.44	2.66	0.12	2.66	0.54	2.66	0.12	2.66

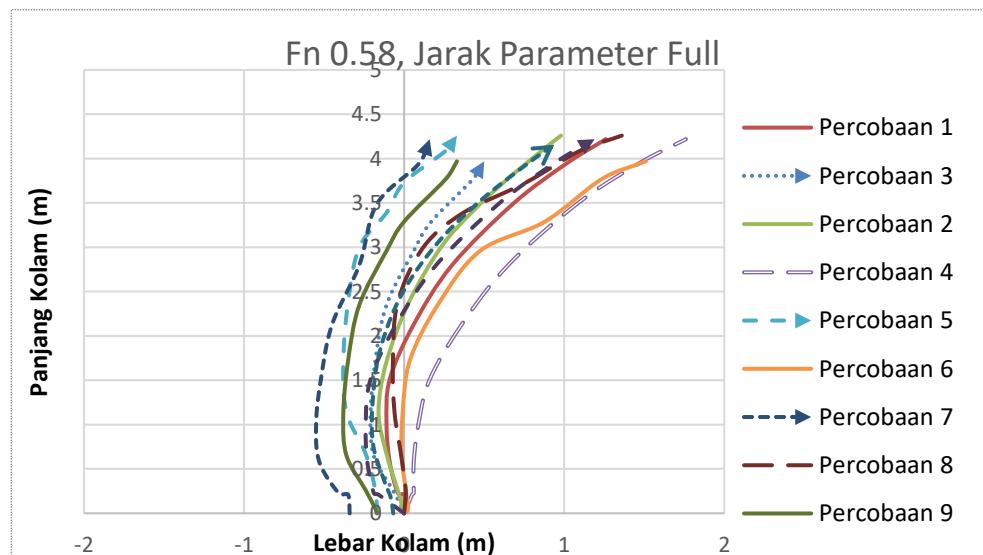
Tabel 4-10 Hasil Data Koordinat dari Video *Trial Air Fn 0.36-75% Jarak Parameter*

f	6		7		8		9		10	
	x	y	x	y	x	y	x	y	x	y
10	0	0	0	0	0	0	0	0	0	0
20	0	0.01	0	0.01	0	0.01	0	0.01	0	0.01
30	0	0.1	0	0.1	0	0.1	0	0.1	0	0.1
40	0	0.29	0	0.29	0.03	0.29	0	0.29	0	0.29
50	0.012	0.59	0	0.59	0.09	0.59	0	0.59	0	0.59
60	0.036	0.98	0	0.98	0.18	0.98	-0.1	0.98	0	0.98
70	0.072	1.37	0.1	1.37	0.3	1.37	-0.1	1.37	-0.06	1.37
80	0.05	1.84	0.3	1.84	0.45	1.84	0	1.84	0.03	1.84
90	0.11	2.22	0.6	2.22	0.53	2.22	0.141421	2.22	0.06	2.22
100	0.182	2.66	0.9	2.66	0.74	2.66	0.214626	2.66	0.21	2.66

Dari Gambar 4.33 bisa dilihat bahwa kapal akan mulai berbelok pada jarak ± 2 m. hal ini dikarenakan jarak awal antar kapal dan benda itu ± 3.6 m, sehingga jika di setting parameter R_{bf} menjadi 1.63 m maka secara otomatis kapal akan bereaksi terhadap deteksi objek yang memasuki area kurang dari 2.17 m ini. Sehingga dari jarak awal 3.6 m itu kapal seharusnya bereaksi sekitar ± 1.97 m, namun realitanya kapal bereaksi pada jarak ± 2 m.

4.6.3. Variasi Fn 0.58 dengan Parameter Jarak Penuh

Dengan sudut konstan maksimal rudder ke arah starboard sejauh 10^0 maka kapal akan membentuk lintasan penghindaran elips seperti yang dilihat pada Gambar 4.34 untuk melihat kecendrungan gerakan kapal pada variasi *forude number* 0.58.



Gambar 4.34 Bentuk Lintasan Fn 0.58-Jarak Parameter Penuh

Dari 10 kali percobaan, maka didapat lintasan kapal yang ditunjukkan pada Gambar 4.34. Dilihat bahwa eksperimen *collision avoidance* dengan perhitungan *blocking area*, dimana untuk variasi Fn 0.58, akan diperlukan jarak ± 2.17 meter dari kapal. Sehingga untuk menguji

kesuksesan dan validasi perhitungan akan diukur terlebih dahulu batas acuan R_{bf} 2.17 meter dari objek yang akan dihindar, lalu kapal akan diuji pada jarak yang lebih dari perhitungan R_{bf} sehingga dapat dipantau kinerja sistem dan mendapat sudut *heading* kapal, untuk variasi ini bisa dilihat pada Tabel 4-11. Dimana disimpulkan bahwa rata-rata sudut *heading* kapal adalah 13.4°

Tabel 4-11 Sudut Heading Kapal untuk Fn 0.58-Dengan Jarak Parameter Penuh

Fn 0.58-1		
n	Sudut Heading	AVG (Deg)
1	16.6	13.4
2	13.0	
3	7.2	
4	22.6	
5	6.7	
6	20.8	
7	6.8	
8	17.7	
9	7.2	
10	15.7	

Visualisasi grafis pada Gambar 4.34, didapat dari hasil proses data berupa video mentah yang di ambil dari kamera *handphone* yang lalu dimasukkan dalam *software blender* dan *fspy*, dimana hasil penangkapan video itu akan di rubah menjadi model 3D oleh *fspy* dan kemudian akan di cari koordinat-koordinat tiap satuan waktu dengan menggunakan *software blender*. Contoh dari hasil pencatatan data dalam bentuk koordinat bisa dilihat pada Tabel 4-12 untuk hasil data 5 percobaan pertama dan Tabel 4-13 untuk 5 percobaan terakhir.

Tabel 4-12 Hasil Data Koordinat dari Video *Trial Air* Fn 0.58-Jarak Parameter Penuh

f	1		2		3		4		5	
	x	y	x	y	x	y	x	y	x	y
10	0	0	0	0	0	0	0	0	-0.17	0
20	0	0.21	-0.04	0.26	-0.01	0.05	0.05	0.21	-0.18	0.26
30	-0.04	0.23	-0.11	0.67	-0.07	0.26	0.06	0.23	-0.24	0.67
40	-0.09	0.58	-0.16	1.14	-0.19	0.65	0.06	0.58	-0.36	1.14
50	-0.11	1.01	-0.11	1.71	-0.21	1.11	0.09	1.01	-0.38	1.71
60	-0.09	1.51	0.03	2.38	-0.18	1.72	0.16	1.51	-0.35	2.38
70	0.06	2.13	0.23	3	-0.11	2.35	0.36	2.13	-0.28	3
80	0.3	2.8	0.42	3.39	0.06	2.96	0.65	2.8	-0.11	3.39
90	0.68	3.49	0.62	3.73	0.17	3.29	1.08	3.49	0	3.73
100	1.01	3.94	0.86	4.09	0.43	3.77	1.46	3.94	0.26	4.09
110	1.26	4.22	0.98	4.26	0.5	3.97	1.76	4.22	0.33	4.26

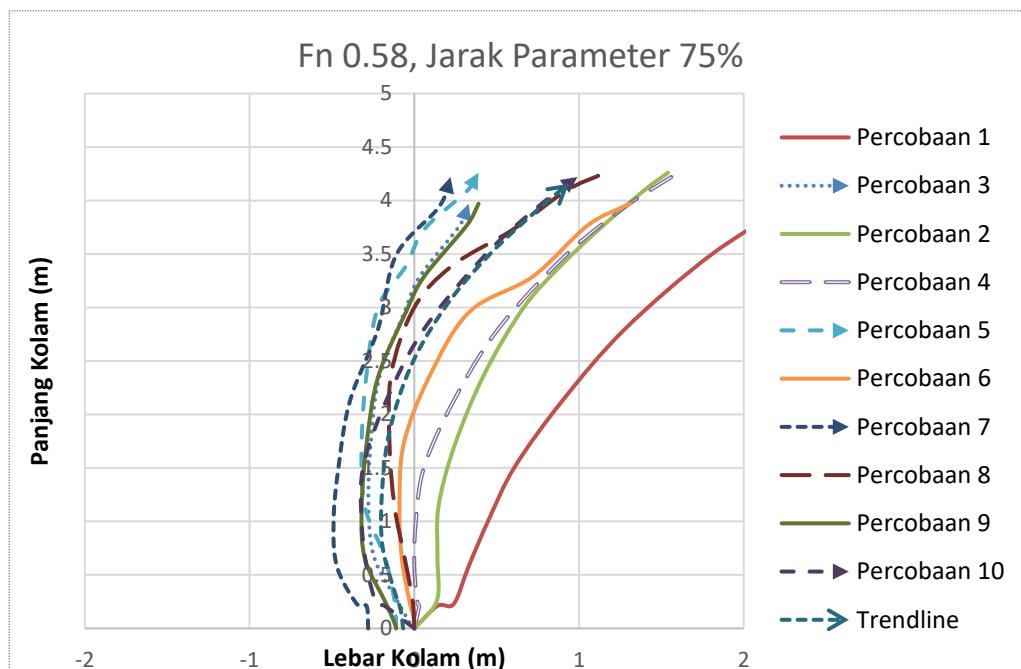
Tabel 4-13 Hasil Data Koordinat dari Video *Trial Air Fn 0.58-Jarak Parameter Penuh*

f	6		7		8		9		10	
	x	y	x	y	x	y	x	y	x	y
10	0	0	-0.34	0	0	0	-0.17	0	0	0
20	0.025	0.05	-0.35	0.21	0.01	0.26	-0.18	0.05	-0.16	0.21
30	0.01	0.26	-0.41	0.23	-0.02	0.67	-0.24	0.26	-0.19	0.23
40	-0.015	0.65	-0.53	0.58	-0.06	1.14	-0.36	0.65	-0.23	0.58
50	-0.01	1.11	-0.55	1.01	-0.07	1.71	-0.38	1.11	-0.24	1.01
60	0.035	1.72	-0.52	1.51	-0.04	2.38	-0.35	1.72	-0.21	1.51
70	0.21	2.35	-0.45	2.13	0.12	3	-0.28	2.35	-0.05	2.13
80	0.475	2.96	-0.28	2.8	0.37	3.39	-0.11	2.96	0.2	2.8
90	0.88	3.29	-0.17	3.49	0.76	3.73	0	3.29	0.59	3.49
100	1.235	3.77	0.09	3.94	1.1	4.09	0.26	3.77	0.93	3.94

Dari Gambar 4.34, bisa dilihat bahwa kapal akan mulai berbelok pada jarak $+/- 1.5$ m. hal ini dikarenakan jarak awal antar kapal dan benda itu $+/- 3.6$ m, sehingga jika di setting parameter R_{bf} menjadi 2.3 m maka secara otomatis kapal akan bereaksi terhadap deteksi objek yang memasuki area kurang dari 2.3 m ini. Sehingga dari jarak awal 3.6 m itu kapal seharusnya bereaksi sekitar $+/- 1.3$ m, namun realitanya kapal bereaksi pada jarak $+/- 1.5$ m

4.6.4. Variasi Fn 0.58 dengan 75% Parameter Jarak

Dengan sudut konstan maksimal rudder ke arah starboard sejauh 10^0 maka kapal akan membentuk lintasan penghindaran elips seperti yang dilihat pada Gambar 4.32 untuk melihat kecendrungan gerakan kapal pada variasi *forude number* 0.58.



Gambar 4.35 Bentuk Lintasan Fn 0.58-75% Jarak Parameter

Dari 10 kali percobaan, maka didapat lintasan kapal yang ditunjukkan pada Gambar 4.35. Dilihat bahwa eksperimen *collision avoidance* dengan perhitungan *blocking area*, dimana

untuk variasi Fn 0.58, akan diperlukan jarak +/-1.17 meter dari kapal. Sehingga untuk menguji kesuksesan dan validasi perhitungan akan diukur terlebih dahulu batas acuan R_{bf} 1.17 meter dari objek yang akan dihindar, lalu kapal akan diuji pada jarak yang lebih dari perhitungan R_{bf} sehingga dapat dipantau kinerja sistem dan mendapat sudut *heading* kapal, untuk variasi ini bisa dilihat pada Tabel 4-14. Dimana disimpulkan bahwa rata-rata sudut *heading* kapal adalah 14.4°

Tabel 4-14 Sudut *Heading* Kapal untuk Fn 0.58 dengan 75% Jarak Parameter

Fn 0.57-0.75		
n	Sudut Heading	AVG (Deg)
1	32.2	14.4
2	19.9	
3	4.8	
4	20.3	
5	6.7	
6	18.3	
7	6.8	
8	15.2	
9	7.2	
10	13.2	

Visualisasi grafis pada Gambar 4.35, didapat dari hasil proses data berupa video mentah yang di ambil dari kamera *handphone* yang lalu dimasukkan dalam *software blender* dan *fspy*, dimana hasil penangkapan video itu akan di rubah menjadi model 3D oleh *fspy* dan kemudian akan di cari koordinat-koordinat tiap satuan waktu dengan menggunakan *software blender*. Contoh dari hasil pencatatan data dalam bentuk koordinat bisa dilihat pada Tabel 4-15 untuk hasil data 5 percobaan pertama dan Tabel 4-16 untuk 5 percobaan terakhir.

Tabel 4-15 Hasil Data Koordinat dari Video *Trial Air Fn 0.57-75%* Jarak Parameter

f	1		2		3		4		5	
	x	y	x	y	x	y	x	y	x	y
10	0	0	0	0	0	0	0	0	-0.11	0
20	0.14	0.21	0.137245	0.26	-0.027	0.05	0.03	0.21	-0.12	0.26
30	0.24	0.23	0.140663	0.67	-0.104	0.26	0.02	0.23	-0.18	0.67
40	0.33	0.58	0.146998	1.14	-0.241	0.65	1.39E-16	0.58	-0.3	1.14
50	0.45	1.01	0.244491	1.71	-0.278	1.11	0.01	1.01	-0.32	1.71
60	0.61	1.51	0.426333	2.38	-0.265	1.72	0.06	1.51	-0.29	2.38
70	0.9	2.13	0.664161	3	-0.212	2.35	0.24	2.13	-0.22	3
80	1.28	2.8	0.888947	3.39	-0.059	2.96	0.51	2.8	-0.05	3.39
90	1.8	3.49	1.121326	3.73	0.034	3.29	0.92	3.49	0.06	3.73
100	2.27	3.94	1.391736	4.09	0.277	3.77	1.28	3.94	0.32	4.09

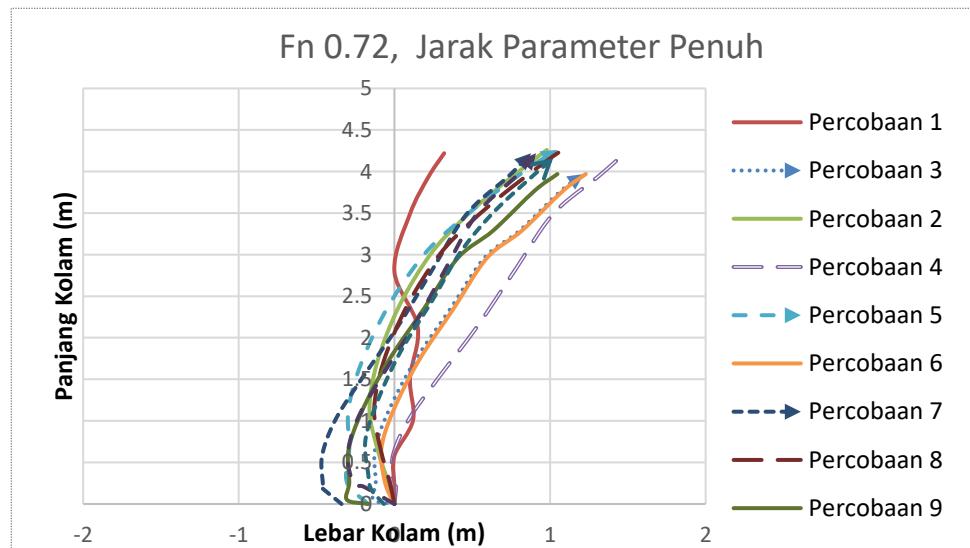
Tabel 4-16 Hasil Data Koordinat dari Video *Trial Air Fn 0.57-75% Jarak Parameter*

f	6		7		8		9		10	
	x	y	x	y	x	y	x	y	x	y
10	0	0	-0.28	0	0	0	-0.11	0	0	0
20	0.005	0.05	-0.29	0.21	-0.01	0.26	-0.12	0.05	-0.18	0.21
30	-0.03	0.26	-0.35	0.23	-0.06	0.67	-0.18	0.26	-0.23	0.23
40	-0.075	0.65	-0.47	0.58	-0.12	1.14	-0.3	0.65	-0.29	0.58
50	-0.09	1.11	-0.49	1.01	-0.15	1.71	-0.32	1.11	-0.32	1.01
60	-0.065	1.72	-0.46	1.51	-0.14	2.38	-0.29	1.72	-0.31	1.51
70	0.09	2.35	-0.39	2.13	5E-16	3	-0.22	2.35	-0.17	2.13
80	0.335	2.96	-0.22	2.8	0.23	3.39	-0.05	2.96	0.06	2.8
90	0.72	3.29	-0.11	3.49	0.6	3.73	0.06	3.29	0.43	3.49
100	1.055	3.77	0.15	3.94	0.92	4.09	0.32	3.77	0.75	3.94

Dari Gambar 4.35, bisa dilihat bahwa kapal akan mulai berbelok pada jarak $+/- 2.5\text{m}$. hal ini dikarenakan jarak awal antar kapal dan benda itu $+/- 3.6\text{ m}$, sehingga jika di setting parameter R_{bf} menjadi 1.17 m maka secara otomatis kapal akan bereaksi terhadap deteksi objek yang memasuki area kurang dari 1.17 m ini. Sehingga dari jarak awal 3.6 m itu kapal seharusnya bereaksi sekitar $+/- 2.43\text{ m}$, namun realitanya kapal bereaksi pada jarak $+/- 2.5\text{ m}$.

4.6.5. Variasi Fn 0.72 dengan Parameter Jarak Penuh

Dengan sudut konstan maksimal rudder ke arah starboard sejauh 10° maka kapal akan membentuk lintasan penghindaran elips seperti yang dilihat pada Gambar 4.32 untuk melihat kecendrungan gerakan kapal pada variasi *forude number* 0.72.



Gambar 4.36 Bentuk Lintasan Fn 0.72-Jarak Parameter Penuh

Dari 10 kali percobaan, maka didapat lintasan kapal yang ditunjukkan pada Gambar 4.36. Dilihat bahwa eksperimen *collision avoidance* dengan perhitungan *blocking area*, dimana

untuk variasi Fn 0.72, akan diperlukan jarak +/-2.4 meter dari kapal. Sehingga untuk menguji kesuksesan dan validasi perhitungan akan diukur terlebih dahulu batas acuan R_{bf} 2.4 meter dari objek yang akan dihindar, lalu kapal akan diuji pada jarak yang lebih dari perhitungan R_{bf} sehingga dapat dipantau kinerja sistem dan mendapat sudut *heading* kapal, untuk variasi ini bisa dilihat pada Tabel 4-17.

Tabel 4-17 Sudut *Heading* Kapal untuk Fn 0.72 dengan Jarak Parameter Penuh

Fn 0.72-1		
n	Sudut Heading	AVG (Deg)
1	10.4	13.9
2	11.7	
3	12.2	
4	17.0	
5	11.7	
6	14.3	
7	11.8	
8	18.9	
9	12.2	
10	19.0	

Visualisasi grafis pada Gambar 4.36, didapat dari hasil proses data berupa video mentah yang di ambil dari kamera *handphone* yang lalu dimasukkan dalam *software blender* dan *fspy*, dimana hasil penangkapan video itu akan di rubah menjadi model 3D oleh *fspy* dan kemudian akan di cari koordinat-koordinat tiap satuan waktu dengan menggunakan *software blender*. Contoh dari hasil pencatatan data dalam bentuk koordinat bisa dilihat pada Tabel 4-18 untuk hasil data 5 percobaan pertama dan Tabel 4-19 untuk 5 percobaan terakhir.

Tabel 4-18 Hasil Data Koordinat dari Video *Trial Air Fn 0.72-Jarak Parameter Penuh*

f	1		2		3		4		5	
	x	y	x	y	x	y	x	y	x	y
10	0	0	-0.17	0	0	0	0	0	-0.17	0
20	0	0.21	-0.18	0.26	-0.01	0.05	0.05	0.21	-0.18	0.26
30	0	0.23	-0.24	0.67	-0.07	0.26	0.1	0.23	-0.24	0.67
40	0	0.58	-0.36	1.14	-0.19	0.65	0.15	0.58	-0.36	1.14
50	0	1.01	-0.38	1.71	-0.21	1.11	0.2	1.01	-0.38	1.71
60	0	1.51	-0.35	2.38	-0.18	1.72	0.25	1.51	-0.35	2.38
70	-0.06	2.13	-0.28	3	-0.11	2.35	0.24	2.13	-0.28	3
80	-0.11	2.8	-0.11	3.39	0.06	2.96	0.24	2.8	-0.11	3.39
90	-0.08	3.49	0	3.73	0.17	3.29	0.32	3.49	0	3.73
100	0.12	3.94	0.26	4.09	0.43	3.77	0.57	3.94	0.26	4.09

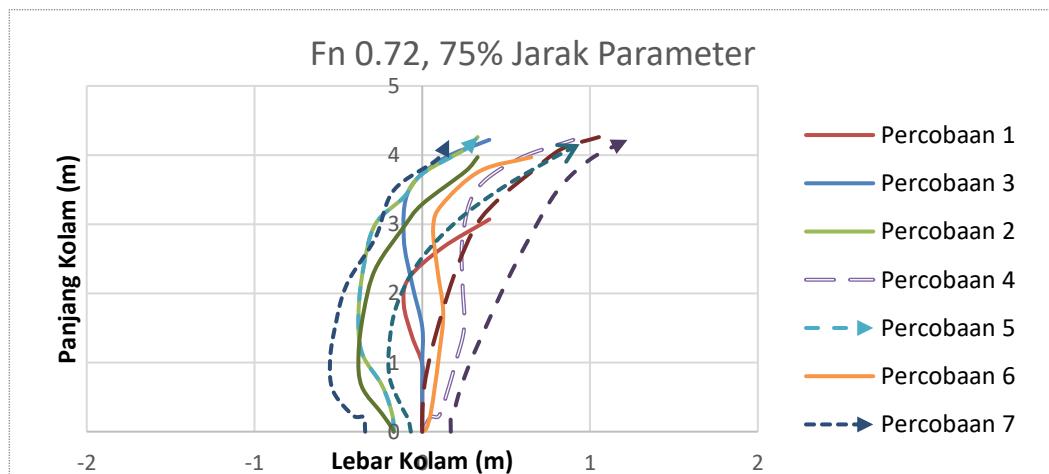
Tabel 4-19 Hasil Data Koordinat dari Video *Trial Air Fn 0.72-Jarak Parameter Penuh*

f	6		7		8		9		10	
	x	y	x	y	x	y	x	y	x	y
10	0	0	-0.34	0	0	0	-0.17	0	0.17	0
20	0.025	0.05	-0.35	0.21	0	0.26	-0.18	0.05	0.17	0.21
30	0.05	0.26	-0.41	0.23	0.013	0.67	-0.24	0.26	0.183	0.23
40	0.075	0.65	-0.53	0.58	0.052	1.14	-0.36	0.65	0.222	0.58
50	0.1	1.11	-0.55	1.01	0.117	1.71	-0.38	1.11	0.287	1.01
60	0.125	1.72	-0.52	1.51	0.208	2.38	-0.35	1.72	0.378	1.51
70	0.09	2.35	-0.45	2.13	0.325	3	-0.28	2.35	0.495	2.13
80	0.065	2.96	-0.28	2.8	0.468	3.39	-0.11	2.96	0.638	2.8
90	0.12	3.29	-0.17	3.49	0.637	3.73	0	3.29	0.807	3.49
100	0.345	3.77	0.09	3.94	0.832	4.09	0.26	3.77	1.002	3.94

Dari Gambar 4.36, bisa dilihat bahwa kapal akan mulai berbelok pada jarak $+/- 1.5$ m. hal ini dikarenakan jarak awal antar kapal dan benda itu $+/- 3.6$ m, sehingga jika di setting parameter R_{bf} menjadi 2.4 m maka secara otomatis kapal akan bereaksi terhadap deteksi objek yang memasuki area kurang dari 2.4 m ini. Sehingga dari jarak awal 3.6 m itu kapal seharusnya bereaksi sekitar $+/- 1.2$ m, namun realitanya kapal bereaksi pada jarak $+/- 1.5$ m.

4.6.6. Variasi Fn 0.72 dengan 75% Parameter Jarak

Dengan sudut konstan maksimal rudder ke arah starboard sejauh 10^0 maka kapal akan membentuk lintasan penghindaran elips seperti yang dilihat pada Gambar 4.32 untuk melihat kecendrungan gerakan kapal pada variasi *forude number* 0.72.



Gambar 4.37 Bentuk Lintasan Fn 0.72-75% Jarak Parameter

Dari 10 kali percobaan, maka didapat lintasan kapal yang ditunjukkan pada Gambar 4.37. Dilihat bahwa eksperimen *collision avoidance* dengan perhitungan *blocking area*, dimana untuk variasi Fn 0.72, akan diperlukan jarak $+/- 1.8$ meter dari kapal. Sehingga untuk menguji kesuksesan dan validasi perhitungan akan diukur terlebih dahulu batas acuan R_{bf} 1.8 meter dari

objek yang akan dihindar, lalu kapal akan diuji pada jarak yang lebih dari perhitungan R_{bf} sehingga dapat dipantau kinerja sistem dan mendapat sudut *heading* kapal, untuk variasi ini bisa dilihat pada Tabel 4-20.

Tabel 4-20 Sudut *Heading* Kapal untuk Fn 0.72 dengan 75% Jarak Parameter

Fn 0.72-0.75		
n	Sudut Heading	AVG (Deg)
1	4.3	14.6
2	13.0	
3	17.1	
4	19.3	
5	16.0	
6	17.2	
7	16.1	
8	14.2	
9	17.1	
10	12.2	

Visualisasi grafis pada Gambar 4.37, didapat dari hasil proses data berupa video mentah yang di ambil dari kamera *handphone* yang lalu dimasukkan dalam *software blender* dan *fspy*, dimana hasil penangkapan video itu akan di rubah menjadi model 3D oleh *fspy* dan kemudian akan di cari koordinat-koordinat tiap satuan waktu dengan menggunakan *software blender*. Contoh dari hasil pencatatan data dalam bentuk koordinat bisa dilihat pada Tabel 4-21 untuk hasil data 5 percobaan pertama dan Tabel 4-22 untuk 5 percobaan terakhir.

Tabel 4-21 Hasil Data Koordinat dari Video *Trial Air Fn 0.72-75%* Jarak Parameter

f	1		2		3		4		5	
	x	y	x	y	x	y	x	y	x	y
10	0	0	0	0	0	0	0	0	-0.17	0
20	0	0.21	-0.04	0.26	-0.135	0.05	0.01	0.21	-0.305	0.26
30	0	0.23	-0.11	0.67	-0.12	0.26	-0.01	0.23	-0.29	0.67
40	0	0.58	-0.16	1.14	-0.125	0.65	-0.01	0.58	-0.295	1.14
50	0.12	1.01	-0.11	1.71	-0.04	1.11	0.09	1.01	-0.21	1.71
60	0.1	1.51	0.03	2.38	0.131	1.72	0.28	1.51	-0.039	2.38
70	0.15	2.13	0.23	3	0.36	2.35	0.53	2.13	0.19	3
80	0	2.8	0.42	3.39	0.577857	2.96	0.77	2.8	0.407857	3.39
90	0.1	3.49	0.62	3.73	0.805	3.29	1.02	3.49	0.635	3.73
100	0.22	3.94	0.86	4.09	1.071667	3.77	1.31	3.94	0.901667	4.09

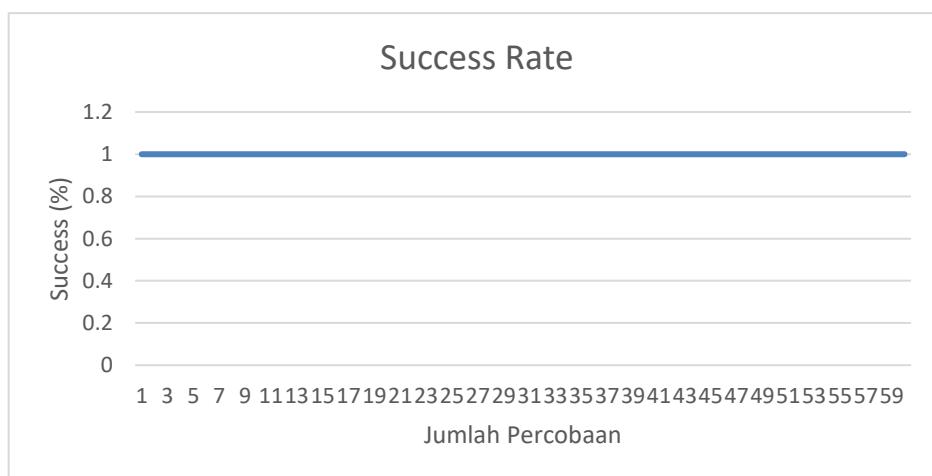
Tabel 4-22 Hasil Data Koordinat dari Video *Trial* Air Fn 0.72-75% Jarak Parameter

	6		7		8		9		10	
f	x	y	x	y	x	y	x	y	x	y
10	0	0	-0.34	0	0	0	-0.17	0	0	0
20	-0.015	0.05	-0.475	0.21	-0.03	0.26	-0.305	0.05	-0.2	0.21
30	-0.06	0.26	-0.46	0.23	-0.09	0.67	-0.29	0.26	-0.26	0.23
40	-0.085	0.65	-0.465	0.58	-0.13	1.14	-0.295	0.65	-0.3	0.58
50	-0.01	1.11	-0.38	1.01	-0.07	1.71	-0.21	1.11	-0.24	1.01
60	0.155	1.72	-0.209	1.51	0.08	2.38	-0.039	1.72	-0.09	1.51
70	0.38	2.35	0.02	2.13	0.29	3	0.19	2.35	0.12	2.13
80	0.595	2.96	0.237857	2.8	0.49	3.39	0.407857	2.96	0.32	2.8
90	0.82	3.29	0.465	3.49	0.7	3.73	0.635	3.29	0.53	3.49
100	1.085	3.77	0.731667	3.94	0.95	4.09	0.901667	3.77	0.78	3.94

Dari Gambar 4.37, bisa dilihat bahwa kapal akan mulai berbelok pada jarak +/- 1.7 m. hal ini dikarenakan jarak awal antar kapal dan benda itu +/- 3.6 m, sehingga jika di setting parameter R_{bf} menjadi 1.8 m maka secara otomatis kapal akan bereaksi terhadap deteksi objek yang memasuki area kurang dari 1.8 m ini. Sehingga dari jarak awal 3.6 m itu kapal seharusnya bereaksi sekitar +/- 1.8 m, namun realitanya kapal bereaksi pada jarak +/- 1.7 m.

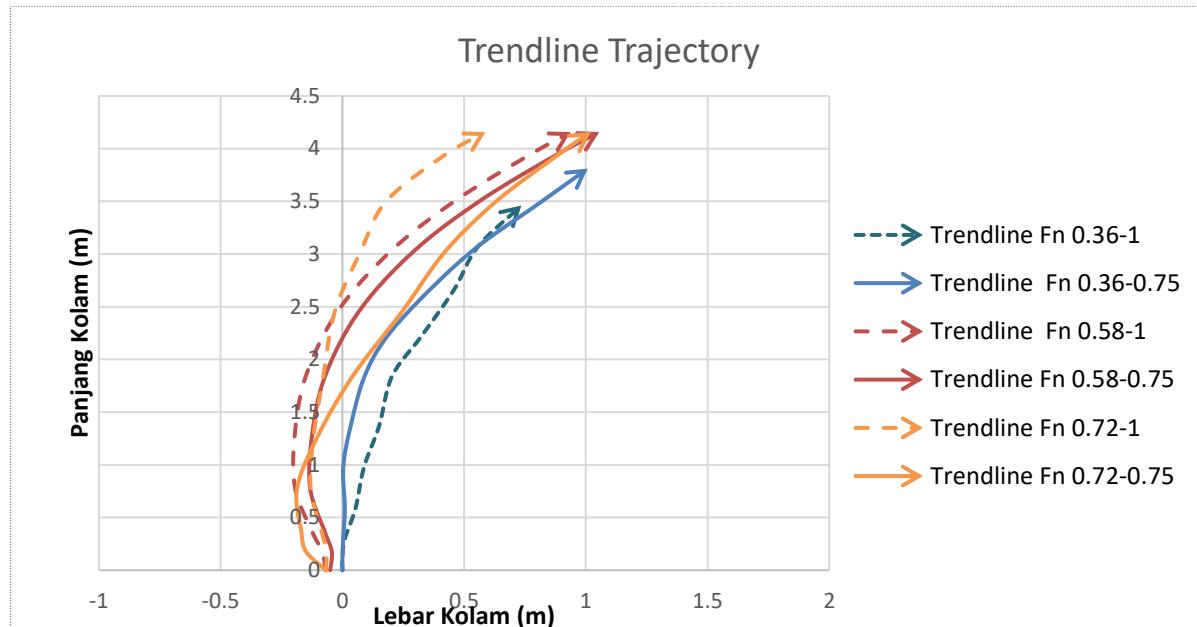
4.6.7. Evaluasi Kinerja Sistem

Setelah mendapat data performa gerakan kapal di air, maka dilakukannya *trial* sistem *Anti Collision* yang menggunakan 1 skenario, yakni *Head-On Collision*. Dengan 2 variasi untuk menguji kemampuan *Object Detection* dari kamera dan pengukuran jarak dengan LiDAR untuk sistem *anti collision* yang kokoh digunakan pada kapal. Dengan variasi jarak dan kecepatan untuk menentukan . Hasil dari 60 kali pengujian bisa dilihat pada Gambar 4.38. Bisa dilihat bahwa sistem yang dibuat telah berhasil 60 kali menghindari objek dengan variasi jarak dan kecepatan yang berbeda-beda.



Gambar 4.38 Plot Success Rate System Collision Avoidance

Selain *success rate* yang dilihat, setelah 60 pengujian juga bisa dilihat *trajectory* rata-rata untuk tiap variasi kecepatan dan jarak sistem, seperti yang ditunjukkan pada Gambar 4.39. Dimana sistem akan mulai mendeteksi dan bergerak saat +/- 2m dan tidak membentuk lintasan yang baik, hal ini dikarenakan faktor angin dan juga *jitter* dari *servo*.



Gambar 4.39 Plotting Trendline tiap Variasi Kecepatan dan Jarak

Dari bentuk lintasan, juga bisa di periksa sudut *heading* rata-rata untuk sistem. Dimana, apabila dilihat pada Tabel 4-23, variasi kecepatan yang mempunyai nilai *froude number* lebih tinggi akan menghasilkan sudut *heading* lebih tinggi dan apabila jarak parameter *blocking area* dikurangi, maka sudut *heading* yang akan dihasilkan sistem akan menjadi lebih besar juga karena perlu manuver yang lebih besar untuk menghindari benda yang lebih dekat.

Tabel 4-23 Rata-Rata Sudut *Heading* Kapal

Average Heading (Deg)	
Variasi 0.36-1	12.6
Variasi 0.36-0.75	14.5
Variasi 0.58-1	13.4
Variasi 0.58-0.75	14.4
Variasi 0.72-1	13.9
Variasi 0.72-0.75	14.6

BAB 5

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Setelah melakukan percobaan dan penelitian, maka kesimpulan yang dapat diperoleh dari Tugas Akhir ini adalah sebagai berikut:

1. Sistem yang dibuat mampu melakukan *object detection* dan pengukuran jarak dengan kamera dan sensor LiDAR.
2. Setelah pengujian di darat untuk memastikan sistem sensor dan aktuator bekerja, bisa disimpulkan bahwa:
 - a. Kamera *object detection* berhasil mendeteksi objek
 - b. Akurasi LiDAR mengukur jarak dengan kondisi program *object detection* aktif memiliki akurasi 95.75% dan kalau *object detection* tidak aktif, akurasi tersebut meningkat hingga 98.91%
 - c. *Servo* mampu dikendalikan dengan merubah nilai *ChangeDutyCycle* (CDC) dengan perubahan sudut sejauh 10°, dimana sudut minimal *rudder* adalah 80° yang berarti kapal berbelok arah *port*, dan maksimal sudut *rudder* adalah 10° yang berarti kapal berbelok arah *starboard*.
3. Setelah 60 kali pengujian di air dengan variasi *froude number* antara 0.36 hingga 0.72, sistem berhasil 60 kali dari 60 kali pengujian. Berarti sistem mempunyai *success rate* 100% untuk skenario *head-on collision* menggunakan methode *blocking area* (Kijima & Furukawa, 2003). Dimana untuk tiap variasi memiliki sudut *heading*:
 - a. Fn 0.36: memiliki rata-rata sudut *heading* 12.6° untuk jarak parameter penuh, dan sudut *heading* rata-rata 14.5° untuk 75% jarak parameter.
 - b. Fn 0.57: memiliki rata-rata sudut *heading* 13.4° untuk jarak parameter penuh, dan sudut *heading* rata-rata 14.4° untuk 75% jarak parameter.
 - c. Fn 0.72: memiliki rata-rata sudut *heading* 8.9° untuk jarak parameter penuh, dan sudut *heading* rata-rata 14.6° untuk 75% jarak parameter.

Namun, pembacaan dari LiDAR masih cenderung salah karena LiDAR merupakan sensor yang sensitif terhadap cahaya sehingga saat pengujian di lapangan terbuka yang disinari cahaya matahari, sensor LiDAR salah membaca dan terjadi *margin of error* yang cukup signifikan, dimana sensor membaca jarak menjadi lebih dekat dari jarak aslinya.

4. Sistem bisa diaplikasikan di kapal, namun perlu pengembangan untuk meningkatkan kinerja dan juga fungsi otomasi sistem menjadi *advance guidance system* untuk mencapai *Maritime Autonomous Surface Ship* (MASS) yang kompleks.

5.2. Saran

Setelah melakukan percobaan dan penilitian maka saran yang bisa diberi untuk Tugas Akhir ini adalah sebagai berikut:

1. Pembuatan *hardware* untuk sistem. Diharapkan menggunakan *microprocessor* NVIDIA Jetson atau penambahan *Coral Edge TPU*.
2. Pembuatan *software* untuk sistem. Diharapkan untuk dikembangkan agar menjadi lebih sederhana.
3. Sensor Jarak. Diharapkan untuk selanjutnya bisa menggunakan sensor yang lebih sederhana, seperti *ultrasonic* sensor.

DAFTAR PUSTAKA

- Aliabady, A. E. (2020). *Collision Avoidance System For Autonomous Vehicles*. California : California State University.
- Andrew G. Howard, M. Z. (2017, 04 17). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. Diambil kembali dari arXiv.net: <https://arxiv.org/pdf/1704.04861.pdf>
- Eivind Meyer, A. H. (2020). *COLREG-Compliant Collision Avoidance for Unmanned Surface Vehicle Using Deep Reinforcement Learning*. Trondheim: IEEE.
- Akdağ, M., Solnør, P., & Johansen, T. A. (2022). Collaborative collision avoidance for Maritime Autonomous Surface Ships: A review. *Ocean Engineering*, 250(February), 110920. <https://doi.org/10.1016/j.oceaneng.2022.110920>
- Catapang, A. N., & Ramos, M. (2017). Obstacle detection using a 2D LIDAR system for an Autonomous Vehicle. *Proceedings - 6th IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2016*, 441–445. <https://doi.org/10.1109/ICCSCE.2016.7893614>
- FRANÇOIS CHOLLET. (2021). Deep Learning with Python. In *Deep Learning with Python*. <https://doi.org/10.1007/978-1-4842-5364-9>
- Girshick, R. (2015). Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision, 2015 Inter*, 1440–1448. <https://doi.org/10.1109/ICCV.2015.169>
- Girshick, R., Donahue, J., Darrell, T., Malik, J., Barduhn, A. J., & Handley, M. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Journal of Chemical and Engineering Data*, 27(3), 306–308. <https://doi.org/10.1021/je00029a022>
- Hasugian, S., Sri Wahyuni, A. A. I., Rahmawati, M., & Arleiny, A. (2018). Pemetaan Karakteristik Kecelakaan Kapal di Perairan Indonesia Berdasarkan Investigasi KNKT. *Warta Penelitian Perhubungan*, 29(2), 229–240. <https://doi.org/10.25104/warlit.v29i2.521>
- Huang, Y., Chen, L., Chen, P., Negenborn, R. R., & van Gelder, P. H. A. J. M. (2020). Ship collision avoidance methods: State-of-the-art. *Safety Science*, 121(April 2019), 451–473. <https://doi.org/10.1016/j.ssci.2019.09.018>
- Kijima, K., & Furukawa, Y. (2003). Automatic collision avoidance system using the concept of blocking area. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 36(21), 223–228. [https://doi.org/10.1016/S1474-6670\(17\)37811-4](https://doi.org/10.1016/S1474-6670(17)37811-4)
- Lagisetty, R., Philip, N. K., Padhi, R., & Bhat, M. S. (2013). Object detection and obstacle avoidance for mobile robot using stereo camera. *Proceedings of the IEEE International Conference on Control Applications, August*, 605–610. <https://doi.org/10.1109/CCA.2013.6662816>
- Liu, Z., Zhang, Y., Yu, X., & Yuan, C. (2016). Unmanned surface vehicles: An overview of developments and challenges. *Annual Reviews in Control*, 41, 71–93. <https://doi.org/10.1016/J.ARCONTROL.2016.04.018>
- Polvara, R., Sharma, S., Wan, J., Manning, A., & Sutton, R. (2018). Obstacle Avoidance Approaches for Autonomous Navigation of Unmanned Surface Vehicles. *Journal of Navigation*, 71(1), 241–256. <https://doi.org/10.1017/S0373463317000753>
- Redmon, J., & Farhadi, A. (2018). YOLO v.3. *Tech Report*, 1–6. <https://pjreddie.com/media/files/papers/YOLOv3.pdf>
- Schiaretti, M., Chen, L., & Negenborn, R. R. (2017). Survey on autonomous surface vessels: Part II - Categorization of 60 prototypes and future applications. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and*

Lecture Notes in Bioinformatics), 10572 LNCS, 234–252. https://doi.org/10.1007/978-3-319-68496-3_16

Zhou, H., Ren, Z., Marley, M., & Skjetne, R. (2021). *A guidance and maneuvering control system design with anti-collision using stream functions with vortex flows for autonomous marine vessels.* 1–15. <http://arxiv.org/abs/2106.02405>

Halaman ini sengaja dikosongkan

LAMPIRAN

Lampiran A *Code Collision Avoidance*

Halaman ini sengaja dikosongkan

LAMPIRAN A

CODE COLLISION AVOIDANCE

```

# Import packages
import os                                     #To interact with Operating system
import argparse                                #To add arguments when executing file
import cv2                                     #To process Image processing in Object Detection
import numpy as np                            #To process numerical side of Object Detection
import sys                                     #To access system resources
import time                                    #To add time related operation in code
from threading import Thread                   #To enable concurrent code execution
import importlib.util                         #To be able to other library utilities
import RPi.GPIO as GPIO                        #To be able to use Raspberry Pi's GPIO Pins
import serial                                  #To be able to read serial data from LiDARS

#
#####
# LiDAR
#####
#LiDAR Setup
ser = serial.Serial("/dev/serial0", 115200,timeout=0) # mini UART serial device
# - - - - - - - - - - - - - - - - - - - - - -
#####
# read ToF data from TF-Luna
#####
#
if ser.isOpen() == False:
    ser.open() # open serial port if not open

#Creating LiDAR class so Threading is possible
class lidar:
    #Funct to read lidar data
    def read_tfluna_data(self):
        while True:
            counter = ser.in_waiting # count the number of bytes of the serial
port
            if counter > 8:
                bytes_serial = ser.read(9) # read 9 bytes
                ser.reset_input_buffer() # reset buffer
                if bytes_serial[0] == 0x59 and bytes_serial[1] == 0x59: #
check first two bytes
                    distance = bytes_serial[2] + bytes_serial[3]*256 # distance in next two bytes
                    return distance
    #Funct to initialize class and start Threading process
    def __init__(self):
        distance = Thread(target=self.read_tfluna_data,args=())
        distance.start()

```

```

#
#####
# Actuator
#####
#Actuator Setup
GPIO.setwarnings(False) #Disable warning of past GPIO Processes
GPIO.setmode(GPIO.BOARD) #To tell python that Raspberry Pi GPIO is read as
Board GPIO not BCM
GPIO.setup(37, GPIO.OUT) #set pin 37 as signal output
GPIO.setup(13, GPIO.OUT) #set pin 13 as signal output

#Assigning GPIO Output to Actuator
servo = GPIO.PWM(37, 50) #set pin 37 as servo output signal of 50 hz
motor = GPIO.PWM(13, 50) #set pin 13 as motor output signal of 50 hz
motor.start(0) #starting motor at 0 volt
servo.start(0) #starting servo at 0 volt

cdc_motor = 6.5 # 5 as in 5% of the total 100% of full PWM
cdc_servo = 5 # Servo is set so that the rudder angle will be at MAX star-
board angle

motor.ChangeDutyCycle(cdc_motor) #To set the ChangeDutyCycle of motor
servo.ChangeDutyCycle(7.3) #Neutral position for servo

#
#####
# Object Detection
#####
#
# Source - Adrian Rosebrock, PyImageSearch:
https://www.pyimagesearch.com/2015/12/28/increasing-raspberry-pi-fps-with-python-and-opencv/
# Define VideoStream class to handle streaming of video from webcam in sepa-
rate processing thread
class VideoStream:
    """Camera object that controls video streaming from the Picamera"""
    def __init__(self, resolution=(640,480), framerate=60):
        # Initialize the PiCamera and the camera image stream
        self.stream = cv2.VideoCapture(0) #Start Video Capture on Raspberry Pi
camera module
        ret = self.stream.set(cv2.CAP_PROP_FOURCC, cv2.Vide-
owriter_fourcc(*'MJPG'))
        ret = self.stream.set(3,resolution[0])
        ret = self.stream.set(4,resolution[1])

        # Read first frame from the stream
        (self.grabbed, self.frame) = self.stream.read()

```

```

# Variable to control when the camera is stopped
    self.stopped = False

def start(self):
    # Start the thread that reads frames from the video stream
    Thread(target=self.update,args=()).start()
    return self

def update(self):
    # Keep looping indefinitely until the thread is stopped
    while True:
        # If the camera is stopped, stop the thread
        if self.stopped:
            # Close camera resources
            self.stream.release()
            return

        # Otherwise, grab the next frame from the stream
        (self.grabbed, self.frame) = self.stream.read()

def read(self):
    # Return the most recent frame
    return self.frame

def stop(self):
    # Indicate that the camera and thread should be stopped
    self.stopped = True

# Define and parse input arguments
parser = argparse.ArgumentParser()
parser.add_argument('--modeldir', help='Folder the .tflite file is located in',
                    required=True)
parser.add_argument('--graph', help='Name of the .tflite file, if different than detect.tflite',
                    default='detect.tflite')
parser.add_argument('--labels', help='Name of the labelmap file, if different than labelmap.txt',
                    default='labelmap.txt')
parser.add_argument('--threshold', help='Minimum confidence threshold for displaying detected objects',
                    default=0.5)
parser.add_argument('--resolution', help='Desired webcam resolution in WxH. If the webcam does not support the resolution entered, errors may occur.',
                    default='1280x720')
parser.add_argument('--edgetpu', help='Use Coral Edge TPU Accelerator to speed up detection',
                    action='store_true')

```

```

args = parser.parse_args()

MODEL_NAME = args.modeldir
GRAPH_NAME = args.graph
LABELMAP_NAME = args.labels
min_conf_threshold = float(args.threshold)
resW, resH = args.resolution.split('x')
imW, imH = int(resW), int(resH)
use_TPU = args.edgetpu

# Import TensorFlow libraries
# If tflite_runtime is installed, import interpreter from tflite_runtime, else
import tensorflow as tf
# If using Coral Edge TPU, import the load_delegate library
pkg = importlib.util.find_spec('tflite_runtime')
if pkg:
    from tflite_runtime.interpreter import Interpreter
    if use_TPU:
        from tflite_runtime.interpreter import load_delegate
else:
    from tensorflow.lite.python.interpreter import Interpreter
    if use_TPU:
        from tensorflow.lite.python.interpreter import load_delegate

# If using Edge TPU, assign filename for Edge TPU model
if use_TPU:
    # If user has specified the name of the .tflite file, use that name, otherwise
    # use default 'edgetpu.tflite'
    if (GRAPH_NAME == 'detect.tflite'):
        GRAPH_NAME = 'edgetpu.tflite'

# Get path to current working directory
CWD_PATH = os.getcwd()

# Path to .tflite file, which contains the model that is used for object detection
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)

# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)

# Load the label map
with open(PATH_TO_LABELS, 'r') as f:
    labels = [line.strip() for line in f.readlines()]

# Have to do a weird fix for label map if using the COCO "starter model" from
# https://www.tensorflow.org/lite/models/object_detection/overview
# First label is '???', which has to be removed.

```

```

if labels[0] == '???':
    del(labels[0])

# Load the Tensorflow Lite model.
# If using Edge TPU, use special load_delegate argument
if use_TPU:
    interpreter = Interpreter(model_path=PATH_TO_CKPT,
                               experimental_delegates=[load_delegate('libedg-
etpu.so.1.0')])
    print(PATH_TO_CKPT)
else:
    interpreter = Interpreter(model_path=PATH_TO_CKPT)

interpreter.allocate_tensors()

# Get model details
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
height = input_details[0]['shape'][1]
width = input_details[0]['shape'][2]

floating_model = (input_details[0]['dtype'] == np.float32)

input_mean = 127.5
input_std = 127.5

# Initialize frame rate calculation
frame_rate_calc = 1
freq = cv2.getTickFrequency()

# Check output layer name to determine if this model was created with TF2 or
TF1,
# because outputs are ordered differently for TF2 and TF1 models
outname = output_details[0]['name']

if ('StatefulPartitionedCall' in outname): # This is a TF2 model
    boxes_idx, classes_idx, scores_idx = 1, 3, 0
else: # This is a TF1 model
    boxes_idx, classes_idx, scores_idx = 0, 1, 2

# Initialize video stream
videostream = VideoStream(resolution=(imW,imH),framerate=60).start()
time.sleep(1)

#initialize lidar
dist = lidar()
condition = "Forward"

```

```

#for frame1 in camera.capture_continuous(rawCapture, for-
mat="bgr",use_video_port=True):
while True:

    # Start timer (for calculating frame rate)
    t1 = cv2.getTickCount()

    #Continously update read data from LiDAR
    distance = dist.read_tfluna_data()
    time.sleep(0.005) #give time for concurrent process

    #Collision Avoidance system based on Blocking Area Method
    if distance<120:
        condition="Starboard" #Based on COLREGs rule 14
        servo.ChangeDutyCycle(cdc_servo)
        motor.ChangeDutyCycle(cdc_motor)
        time.sleep(5)
        break

    # Grab frame from video stream
    frame1 = videotostream.read()

    # Acquire frame and resize to expected shape [1xHxWx3]
    frame = frame1.copy()
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame_resized = cv2.resize(frame_rgb, (width, height))
    input_data = np.expand_dims(frame_resized, axis=0)

    # Normalize pixel values if using a floating model (i.e. if model is non-
    quantized)
    if floating_model:
        input_data = (np.float32(input_data) - input_mean) / input_std

    # Perform the actual detection by running the model with the image as in-
    put
    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()

    # Retrieve detection results
    boxes = interpreter.get_tensor(output_details[boxes_idx]['index'])[0] #
    Bounding box coordinates of detected objects
    classes = interpreter.get_tensor(output_details[classes_idx]['index'])[0]
    # Class index of detected objects
    scores = interpreter.get_tensor(output_details[scores_idx]['index'])[0] #
    Confidence of detected objects

    # Loop over all detections and draw detection box if confidence is above
    minimum threshold

```

```

for i in range(len(scores)):
    if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):

        # Get bounding box coordinates and draw box
        # Interpreter can return coordinates that are outside of image
        dimensions, need to force them to be within image using max() and min()
        ymin = int(max(1,(boxes[i][0] * imH)))
        xmin = int(max(1,(boxes[i][1] * imW)))
        ymax = int(min(imH,(boxes[i][2] * imH)))
        xmax = int(min(imW,(boxes[i][3] * imW)))

        cv2.rectangle(frame, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)
        # Draw label
        object_name = labels[int(classes[i])] # Look up object name from
        "labels" array using class index
        label = '%s: %d%' % (object_name, int(scores[i]*100)) # Example:
        'person: 72%'
        labelSize, baseLine = cv2.getTextSize(label,
        cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2) # Get font size
        label_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw
        label too close to top of window
        cv2.rectangle(frame, (xmin, label_ymin-labelSize[1]-10),
        (xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED) # Draw white box to put label text in
        cv2.putText(frame, label, (xmin, label_ymin-7),
        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2) # Draw label text
        # Draw framerate in corner of frame
        cv2.putText(frame, 'FPS: {:.2f}'.format(frame_rate_calc), (30,
        50),cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 0), 2, cv2.LINE_AA)
        #Draw LiDAR Distance in corner of frame
        cv2.putText(frame,'Distance: {:.2f} cm'.format(distance),(900,
        700),cv2.FONT_HERSHEY_SIMPLEX,1,(255,0,0),2, cv2.LINE_AA)
        #Draw Rudder in corner of frame
        cv2.putText(frame,'Rudder dir: %s' %condition,(50,
        700),cv2.FONT_HERSHEY_SIMPLEX,1,(255,0,0),2, cv2.LINE_AA)
        # All the results have been drawn on the frame, so it's time to display
        it.
        cv2.imshow('Object detector', frame)
        # Press 'q' to quit
        if cv2.waitKey(1) == ord('q'):
            break
    # Clean up
motor.stop()
servo.stop()
GPIO.cleanup()
sys.exit()
cv2.destroyAllWindows()
videostream.stop()

```

BIODATA PENULIS



Anthony Suryajaya dilahirkan di Surabaya pada 30 Januari 2000. Penulis adalah anak keempat dalam keluarganya. Penulis menempuh pendidikan formal tingkat dasar di TK Gloria 1, kemudian SD Anugerah Pekerti, SMPK Santa Maria Surabaya dan SMAK St. Louis 1 Surabaya. Setelah lulus SMA, penulis diterima di Departemen Teknik Perkapalan FTK ITS pada tahun 2018 melalui jalur SBMPTN Tulis.

Di Departemen Teknik Perkapalan Penulis mengambil Bidang Studi Teknologi Kapal Digital. Selama masa studi di ITS, selain kuliah Penulis juga pernah menjadi Kepala Divisi Keprofesian Departemen Riset dan Keprofesian HIMATEKPAL ITS 2021/2022, *staff* Hubungan Luar TPKB 2019/2020, Asisten Lab Teknologi Kapal Digital 2021/2022 serta menjadi *staff* Departemen *Bussiness to Bussiness* (B2B) dari AIESEC in Surabaya.

Penulis tercatat pernah menjadi asisten lab Teknologi Kapal Digital (TKD).

Email: anthony_suryajaya@outlook.com