



TUGAS AKHIR - IF184802

Autonomous Surface Vehicle Pencari Korban Kecelakaan Laut Berbasis Computer Vision

ACHMAD ZIDAN AKBAR
NRP 0511184000005

Dosen Pembimbing 1

Dr. Eng. Chastine Fatichah, S.Kom., M.Kom.

NIP 197512202001122002

Dosen Pembimbing 2

Dr. Rudy Dikairono, S.T., M.T., M.Sc.

NIP 198103252005011002

PROGRAM STUDI S-1 TEKNIK INFORMATIKA

Departemen Teknik Informatika

Fakultas Teknologi Elektro Dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2022



TUGAS AKHIR - 184802

Autonomous Surface Vehicle Pencari Korban Kecelakaan Laut Berbasis Computer Vision

ACHMAD ZIDAN AKBAR

NRP 05111840000005

Dosen Pembimbing 1

Dr. Eng. Chastine Fatichah, S.Kom., M.Kom.

NIP 197512202001122002

Dosen Pembimbing 2

Dr. Rudy Dikairono, S.T., M.T., M.Sc.

NIP 198103252005011002

Program Studi S-1 TEKNIK INFORMATIKA

Departemen Teknik Informatika

Fakultas Teknologi Elektro Dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2022



FINAL PROJECT - IF184802

Computer Vision based Autonomous Surface Vehicle for Search of Marine Casualty

ACHMAD ZIDAN AKBAR

NRP 0511184000005

Advisor I

Dr. Eng. Chastine Fatichah, S.Kom., M.Kom.

NIP 197512202001122002

Advisor II

Dr. Rudy Dikairono, S.T., M.T., M.Sc.

NIP 198103252005011002

Study Program S-1 INFORMATICS

Department of Informatics

Faculty of Intelligent Electrical and Informatics Technology

Institut Teknologi Sepuluh Nopember

Surabaya

2022

LEMBAR PENGESAHAN

Autonomous Surface Vehicle Pencari Korban Kecelakaan Laut Berbasis Computer Vision TUGAS AKHIR

Diajukan untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Komputer pada
Program Studi S-1 Teknik Informatika

Departemen Teknik Informatika
Fakultas Teknologi Elektro Dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh : **ACHMAD ZIDAN AKBAR**
NRP. 05111840000005

Disetujui oleh Tim Penguji Tugas Akhir :

1. Dr. Eng. Chastine Faticahah, S.Kom., M.Kom. Pembimbing
2. Dr. Rudy Dikairono, S.T., M.T., M.Sc. Ko-pembimbing
3. Nama dan gelar penguji Penguji
4. Nama dan gelar penguji Penguji
5. Nama dan gelar penguji Penguji

SURABAYA
Bulan, 2022

PERNYATAAN ORISINALITAS

Yang bertanda tangan di bawah ini:

Nama mahasiswa / NRP : Achmad Zidan Akbar / 05111840000005
Program Studi : S-1 Teknik Informatika
Dosen Pembimbing 1 / : Dr. Eng. Chastine Faticahah, S.Kom., M.Kom. /
NIP : 197512202001122002
Dosen Pembimbing 2 / : Dr. Rudy Dikairono, S.T., M.T., M.Sc. /
NIP : 198103252005011002

dengan ini menyatakan bahwa Tugas Akhir dengan judul "**Autonomous Surface Vehicle Pencari Korban Kecelakaan Laut Berbasis Computer Vision**" adalah hasil karya sendiri, bersifat orisinal, dan ditulis dengan mengikuti kaidah penulisan ilmiah.

Bilamana di kemudian hari ditemukan ketidaksesuaian dengan pernyataan ini, maka saya bersedia menerima sanksi sesuai dengan ketentuan yang berlaku di Institut Teknologi Sepuluh Nopember.

Surabaya, Tanggal Bulan 2022
Mahasiswa



Achmad Zidan Akbar
NRP. 05111840000005

Mengetahui,

Dosen Pembimbing 1

Dosen Pembimbing 2



Dr. Eng. Chastine Faticahah, S.Kom.,
M.Kom.
NIP. 197512202001122002

Dr. Rudy Dikairono, S.T., M.T., M.Sc.
NIP. 198103252005011002

ABSTRAK

AUTONOMOUS SURFACE VEHICLE PENCARI KORBAN KECELAKAAN LAUT BERBASIS COMPUTER VISION

Nama Mahasiswa / NRP	: Achmad Zidan Akbar / 0511184000005
Departemen	: Teknik Informatika, FTEIC - ITS
Dosen Pembimbing 1	: Dr. Eng. Chastine Fatichah, S.Kom., M.Kom.
Dosen Pembimbing 2	: Dr. Rudy Dikairono, S.T., M.T., M.Sc.

Abstrak

Berdasarkan laporan hasil investigasi Komite Nasional Keselamatan Transportasi (KNKT) di wilayah perairan Indonesia pada kurun waktu tahun 2010-2016, terjadi 54 kecelakaan di laut dengan berbagai jenis kejadian seperti tenggelam, terguling, kandas, dan tabrakan. Dalam proses pencarian dan evakuasi korban oleh Tim *Search and Resue* (SAR) kerap dijumpai hambatan seperti cuaca buruk, lokasi kecelakaan, keterbatasan peralatan, dan keterbatasan tenaga tim SAR. Hambatan tersebut tentu akan mengurangi probabilitas korban kecelakaan untuk selamat.

Diusulkan sebuah *Autonomous Surface Vehicle* (ASV) yang dikembangkan untuk membantu proses penyelamatan korban kecelakaan laut. Jenis lambung kapal katamaran diimplementasikan dalam ASV ini karena memiliki tingkat stabilitas yang baik. ASV ini menggunakan sistem propulsi elektrik *thruster* T200 yang disematkan pada bagian belakang robot menggunakan sistem *fix mounting*. *Robot Operating System* (ROS) diimplementasikan sebagai sistem arsitektur perangkat lunak utama. ASV dilengkapi sensor navigasi seperti *Global Positioning System* (GPS), kompas, *Inertial Measuring Unit* (IMU), dan *gyroscope*. Sensor navigasi tersebut digunakan sebagai referensi program kontrol untuk melakukan navigasi *autonomous*. ASV juga menggunakan sensor *ultrasonic* untuk mendukung kemampuan *obstacle avoidance*. Untuk mendukung fungsionalitas ke aktuator, digunakan STM32F4 sebagai *microcontroller*. *Computer Vision* ditambahkan untuk meningkatkan kemampuan dari ASV untuk mendeteksi korban kecelakaan laut menggunakan kamera. Arsitektur YOLOv4 CNN digunakan sebagai pendekripsi orang tenggelam. TensorRT juga digunakan sebagai *optimizer* model dengan perangkat *graphics processing unit* (GPU) Nvidia.

Dengan menggunakan sistem navigasi yang dikembangkan, ASV mampu melakukan navigasi melewati perairan dalam operasi penyelamatan dengan baik. Sistem navigasi mampu memberikan output kontrol yang baik meskipun terdapat *noise* atau gangguan pada sensor kompas. ASV mampu menghindari halangan dengan cukup baik pada kecepatan rendah. Anotasi dataset dilakukan secara manual, dengan data gambar yang diambil di Danau 8 ITS. YOLOv4 sebagai arsitektur yang dipilih menghasilkan evaluasi mAP@IoU=0.5 pada data *testing* sebesar 0.840203. Terdapat peningkatan performa *inference*, model darknet YOLO yang diubah menjadi model TensorRT, dari 27 FPS menjadi 85 FPS. Dengan naiknya performa *inference time*, mAP@IoU=0.5 dari model tensorRT turun menjadi 0.776.

Kata kunci: *Autonomous Surface Vehicle, Computer Vision, YOLO, Search and Rescue, Global Navigation Control, TensorRT*.

ABSTRACT

COMPUTER VISION BASED AUTONOMOUS SURFACE VEHICLE FOR SEARCH OF MARINE CASUALTY

Student Name / NRP	: Achmad Zidan Akbar / 0511184000005
Department	: Informatics, F-ELECTICS ITS
Advisor I	: Dr. Eng. Chastine Fatichah, S.Kom., M.Kom.
Advisor II	: Dr. Rudy Dikairono, S.T., M.T., M.Sc.

Abstract

Based on the investigation report of the National Transportation Safety Committee (NTSC) in Indonesian waters in the period 2010-2016, there were 54 accidents at sea with various types of events such as drowning, overturning, running aground, and collisions. During the search and evacuation process for victims, the Search and Rescue (SAR) Team often encounters obstacles such as bad weather, accident locations, limited equipment, and limited staff for the SAR team. These obstacles will certainly reduce the possibility of accident victims to survive.

An Autonomous Surface Vehicle (ASV) is proposed to be developed to assist the evacuation process of victims of marine accidents. Catamaran hull type was implemented in this ASV because it had a good stability. This ASV used the T200 thruster electronic propulsion system which was attached to the rear of the robot using a fix mounting system. Robot Operating System (ROS) was implemented as the main software architecture system. ASV was equipped with navigation sensors such as the Global Positioning System (GPS), Compass, Inertial Measuring Unit (IMU), and gyroscope. The navigation sensor was used as a program control reference for autonomous navigation. The ASV also used an ultrasonic sensor to support obstacle avoidance capabilities. To support the functionality to the actuator, STM32F4 was used as a microcontroller. Computer Vision was added to enhance the ability of the ASV to detect marine accident victims using cameras. YOLOv4 CNN Architecture was used as a drowning person detector. TensorRT was also used as an optimizer model with Nvidia graphics processing unit (GPU) devices.

By using the developed navigation system, the ASV could navigate quite well through the waters for rescue operation. The navigation system could provide good output control although there was noise or interference in compass sensor. ASV could avoid obstacles quite well at low speeds. Dataset annotation was done manually, with image data taken at Lake 8 ITS. YOLOv4 as the chosen architecture produced the evaluation of mAP@IoU=0.5 on the testing data of 0.840203. There was an increase in inference performance, the YOLO darknet model was changed to the TensorRT model, from 27 FPS to 85 FPS. As the performance inference time increased, the mAP@IoU=0.5 of the tensorRT model decreased to 0.776.

Keywords: *Autonomous Surface Vehicle, Computer Vision, Search and Rescue, Global Navigation Control, Catamaran.*

KATA PENGANTAR

Bismillahirrahmanirrahim

Puji syukur kepada Allah Yang Maha Esa atas segala karunia dan rahmat-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul:

AUTONOMOUS SURFACE VEHICLE PENCARI KORBAN KECELAKAAN LAUT BERBASIS COMPUTER VISION

Pengerjaan tugas akhir ini dilakukan untuk memenuhi salah satu syarat meraih gelar Sarjana di Departemen Teknik Informatika, Fakultas Teknologi Elektro dan Informatika Cerdas Institut Teknologi Sepuluh Nopember Surabaya.

Harapan dari penulis, dengan selesainya tugas akhir yang telah dikerjakan dapat memberikan manfaat bagi perkembangan ilmu pengetahuan terutama di bidang robotika serta bagi diri penulis selaku peneliti.

Dalam pelaksanaan dan pembuatan tugas akhir ini tentunya sangat banyak bantuan yang penulis terima dari berbagai pihak, tanpa mengurangi rasa hormat penulis ingin mengucapkan terima kasih sebesar-besarnya kepada:

1. Allah SWT. Yang telah memberikan kesehatan umur serta kelancaran kepada penulis dalam mengerjakan tugas akhir dan Nabi Muhammad SAW. yang telah membimbing penulis selama hidup.
2. Keluarga penulis (Papa, Mama, dan keluarga penulis yang lain) yang selalu memberikan dukungan baik berupa doa, moral, dan material yang tak terhingga kepada penulis, sehingga penulis dapat menyelesaikan tugas akhir ini.
3. Ibu Dr. Eng. Chastine Faticah, S.Kom., M.Kom. selaku Dosen Pembimbing yang telah membimbing penulis selama penyelesaian tugas akhir ini. Dosen yang selalu memberikan ilmu, nasihat serta motivasi kepada penulis selama berkuliahan di Departemen Teknik Informatika ITS. Selaku kepala Departemen Teknik Informatika, beliau juga telah menyediakan berbagai fasilitas penunjang pembelajaran.
4. Bapak Dr. Rudy Dikairono, S.T., M.T., M.Sc. selaku Dosen Pembimbing penulis yang telah memberikan ilmu serta masukan kepada penulis selama berada di tim riset robotika dan memberikan masukan selama penyelesaian tugas akhir ini.
5. Bapak dan Ibu Dosen yang telah memberikan ilmunya selama penulis berkuliahan di Informatika ITS.
6. Robotika ITS yang dipimpin, Bapak Muhtadin, S.T., M.T., telah memberikan fasilitas pendukung untuk penyelesaian Tugas Akhir penulis khususnya pada sektor alat bengkel serta peralatan robot lainnya.
7. Rekan penulis dari tim Barunastra ITS yang mendukung penulis selama berkuliahan hingga menyelesaikan Tugas Akhir di Teknik Informatika ITS. Ucapan terima kasih khusus penulis juga sampaikan kepada Edgar dan Febro yang telah membantu proses trial dalam pengerjaan tugas akhir. Nawab Aditya dan Halen yang menyediakan PCB dan Mikrokontroller. Rayyan yang membantu proses pembuatan mounting propulsi. Azayaka yang membantu pemasangan propulsi pada tahap awal.
8. Teman-teman angkatan 2018 Departemen Teknik Informatika ITS yang sudah menemani penulis selama berkuliahan di Teknik Informatika ITS.

9. Serta pihak-pihak lain yang tidak dapat disebutkan disini yang telah banyak membantu penulis dalam penyusunan tugas akhir ini.

Penulis telah berusaha sebaik-baiknya dalam menyusun tugas akhir ini. Namun, penulis memohon maaf apabila terdapat kekurangan, kesalahan maupun kelalaian yang telah penulis lakukan. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan selanjutnya. Semoga kita semua selalu diberi kebahagiaan lahir dan batin serta kesuksesan dunia akhirat. Aamiin.

Surabaya, Bulan 2022



Achmad Zidan Akbar

DAFTAR ISI

LEMBAR PENGESAHAN	i
PERNYATAAN ORISINALITAS	ii
ABSTRAK	iii
ABSTRACT	iv
KATA PENGANTAR	v
DAFTAR ISI	vii
DAFTAR GAMBAR	ix
DAFTAR TABEL	x
DAFTAR SIMBOL	xi
BAB 1 PENDAHULUAN	12
1.1 Latar Belakang	12
1.2 Rumusan Masalah	13
1.3 Batasan Masalah	13
1.4 Tujuan	13
1.5 Manfaat	13
1.6 Tahapan Tugas Akhir	13
1.6.1 Penyusunan Proposal Tugas Akhir	13
1.6.2 Studi Literatur	14
1.6.3 Desain dan Analisis	14
1.6.4 Implementasi Perangkat Lunak	14
1.6.5 Pengujian dan Evaluasi	14
1.6.6 Penyusunan Buku Tugas Akhir	15
1.7 Sistematika Penulisan Buku Tugas Akhir	15
BAB 2 TINJAUAN PUSTAKA	16
2.1 Hasil Penelitian Terdahulu	16
2.2 Dasar Teori	16
2.2.1 Autonomous Surface Vehicle	16
2.2.2 Autonomous Navigation System	16
2.2.3 Computer Vision	17
2.2.4 TensorRT	18
2.2.5 Communication	19
2.2.6 ROS	19
2.2.7 Evaluasi	19
2.2.8 Evaluasi Visi Komputer	19
2.2.9 Evaluasi Navigasi	19
BAB 3 METODELOGI PENELITIAN	20
3.1 Desain Mekanikal ASV	20

3.1.1	Lambung ASV	20
3.1.2	Propulsi ASV	20
3.2	Desain Elektrikal ASV	21
3.2.1	Sistem Daya dan Jalur Data	21
3.2.2	Mikrokontroller Nucleo-STM32F429ZI	21
3.2.3	Pixhawk Cube	22
3.2.4	Ultrasonic Sensor	22
3.3	Desain Arsitektur Perangkat Lunak ROS	23
3.3.1	PERCEPTION_NODE	23
3.3.2	CONTROL_NODE	24
3.4	Computer Vision	26
3.5	Antarmuka Monitoring dan Tuning	26
3.6	Implementasi PERCEPTION_NODE	27
3.6.1	asv_tf	27
3.6.2	map_image	28
3.6.3	rviz_plugin	29
3.7	Implementasi CONTROL_NODE	29
3.7.1	Local Control	30
3.7.2	Global Control	33
3.7.3	PID Controller	34
3.8	Implementasi Computer Vision	35
3.8.1	Pembuatan Dataset	35
3.8.2	Training Model	36
3.8.3	Konversi Model ke TensorRT	36
3.9	Implementasi STM32_COMMUNICATION	37
3.10	Implementasi Antarmuka Tuning	38
BAB 4	HASIL DAN PEMBAHASAN	39
4.1	Lingkungan Uji Coba	39
4.2	Skenario Uji Coba dan Analisis	40
4.2.1	Local Control	40
4.2.2	Global Control	43
4.2.3	Computer Vision	44
BAB 5	KESIMPULAN DAN SARAN	48
5.1	Kesimpulan	48
5.2	Saran	48
	DAFTAR PUSTAKA	49
	LAMPIRAN	51
	BIODATA PENULIS	54

DAFTAR GAMBAR

Gambar 3.1 Desain Prototipe ASV	20
Gambar 3.2 Thruster T200 Bluerobotics yang dipasang dengan konfigurasi fix.....	20
Gambar 3.3 Diagram Aliran Daya dan Aliran Data	21
Gambar 3.4 Nucleo-STM32F429ZI dengan PCB	22
Gambar 3.5 Pixhawk Cube dengan External GPS	22
Gambar 3.6 Arsitektur ROS Node.....	23
Gambar 3.7 Orientasi TF	24
Gambar 3.8 Flowchart CONTROL_NODE	24
Gambar 3.9 Ilustrasi Error WaypointControl	26
Gambar 3.10 Antarmuka PERCEPTION_NODE rviz.....	27
Gambar 3.11 Struktur Kelas CONTROL_NODE	29
Gambar 3.12 Visualisasi CameraControl	31
Gambar 3.13 Penempatan Sensor Ultrasonic pada ASV.....	32
Gambar 3.14 Dataset yang Digunakan dalam Proses Training	35
Gambar 3.15 Datest yang Dibuat di luar Dataset Training	36
Gambar 3.16 Tampilan Antarmuka berbasis Qt untuk Kebutuhan Tuning Parameter	38
Gambar 4.1 Peta Satelit dari Danau 8 ITS.....	39
Gambar 4.2 Prototipe ASV saat Melakukan Uji Coba di Danau 8 ITS	39
Gambar 4.3 Grafik Error CameraControl.....	41
Gambar 4.4 Grafik Error CameraControl dengan Kondisi Korban Cenderung Diam	41
Gambar 4.5 Grafik Error Global Control	44
Gambar 4.6 Grafik Evaluasi mAP pada saat Training Menggunakan Darknet.....	45

DAFTAR TABEL

Tabel 4.1 Spesifikasi Perangkat Keras yang Digunakan oleh Perangkat Lunak.....	40
Tabel 4.2 Spesifikasi Lingkungan Perangkat Lunak	40
Tabel 4.3 Percobaan Obstacle Avoidance Menggunakan Sensor Ultrasonic.....	42
Tabel 4.4 Perbandingan Performa Inference Model.....	45
Tabel 4.5 Perbandingan Evaluasi mAP dari Model	46

DAFTAR KODE SUMBER

Kode Sumber 3.1 TF Publisher	28
Kode Sumber 3.2 map_image publish satelite image to rviz	28
Kode Sumber 3.3 State Machine CONTROL_NODE	30
Kode Sumber 3.4 Implementasi CameraControl.....	31
Kode Sumber 3.5 Implementasi Obstacle Avoidance Menggunakan Tiga Sensor Ultrasonic	33
Kode Sumber 3.6 Implementasi Global Control	34
Kode Sumber 3.7 Implementasi PID Controller	35
Kode Sumber 3.8 Implementasi Pubsliher Bounding Box Hasil Deteksi YOLO	37
Kode Sumber 3.9 Implementasi Publisher Image	37
Kode Sumber 3.10 Implementasi Komunikasi UDP ke Mikrokontroller STM32	38

BAB 1 PENDAHULUAN

Pada bab ini dijelaskan mengenai garis besar tugas akhir yang meliputi latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan tugas akhir. Diharapkan dari penjelasan dalam bab ini, gambaran tugas akhir secara umum dapat dipahami.

1.1 Latar Belakang

Kecelakaan merupakan kondisi tak terduga yang sering terjadi dalam kehidupan, tak terkecuali dalam ruang lingkup maritim. Berdasarkan laporan hasil investigasi Komite Nasional Keselamatan Transportasi (KNKT) di wilayah perairan Indonesia pada kurun waktu tahun 2010-2016, terjadi 54 kecelakaan di laut dengan berbagai jenis kejadian seperti tenggelam, terguling, kandas, dan tabrakan [1]. Ketika terjadi kecelakaan, evakuasi penumpang dan kru harus segera dilakukan secepat dan seefisien mungkin. Namun, terdapat faktor-faktor lain yang berkontribusi pada proses pencarian korban, antara lain yaitu cuaca, lokasi kecelakaan, serta peralatan dan teknologi yang digunakan oleh Tim *Search and Rescue* (SAR). Seperti pada kasus kecelakaan pesawat Sriwijaya Air SJ 182 diawal 2021, tim penyelamat mengungkapkan bahwa pencarian korban bukanlah hal yang mudah karena terhambat cuaca, penglihatan yang kurang ketika malam hari, serta membutuhkan banyak relawan yang harus diterjunkan ke lapangan [2]. Hambatan yang ada dalam proses pencarian dan evakuasi tentunya akan berdampak pada turunnya probabilitas korban untuk selamat.

Inovasi teknologi untuk membantu proses pencarian korban terus dikembangkan, salah satunya E.M.I.L.Y. E.M.I.L.Y merupakan suatu *boat* berbentuk pelampung yang digunakan untuk menyelamatkan korban kecelakaan laut dengan menggunakan *remote control*. Alat tersebut ditujukan untuk proses evakuasi korban, sehingga tidak dapat digunakan untuk proses pencarian dimana korban belum diketahui posisinya. Seiring berkembangnya teknologi di dunia deep learning khususnya convolutional neural network (CNN) memberikan solusi terhadap berbagai masalah visi computer. Telah dikembangkan beberapa penelitian dalam penggunaan visi computer untuk keperluan deteksi suatu objek. Terdapat penelitian mengenai bagaimana menghitung jumlah orang menggunakan CNN [3]. Telah dikembangkan juga sebuah *drone* yang mampu mendeteksi kendaraan yang ada di suatu jalan raya untuk keperluan infrastruktur transportasi cerdas [4]. Adapun penelitian untuk mendeteksi kapal menggunakan R-CNN menggunakan gambar SAR [5].

Untuk mengakomodir permasalahan yang telah dijelaskan sebelumnya dan berdasarkan pada penelitian – penelitian yang telah ada, pada Tugas Akhir ini diusulkan sebuah prototipe *Autonomous Surface Vehicle* (ASV) yang dapat melakukan proses pencarian korban kecelakaan laut dengan basis kamera melalui *computer vision*. Adapun *deep learning* yang digunakan berbasis CNN untuk melakukan deteksi korban kecelakaan laut dari kamera yang terpasang pada ASV. Kemampuan deteksi dari CNN untuk mendeteksi korban kecelakaan laut, kemudian digunakan dalam melakukan misi pencarian *autonomous*. Usulan ASV ini tentu juga akan dibekali sistem navigasi *autonomous* yang memperbolehkan ASV bernaligasi secara nirawak dalam kondisi lingkungan yang tidak diketahui. Dengan dukungan sensor – sensor navigasi seperti GPS, kompas, imu, dan sensor *ranging* seperti sensor *ultrasonic*, ASV yang diusulkan diharap mampu melakukan navigasi *autonomous* dengan baik.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut :

1. Bagaimana sistem *computer vision* yang digunakan untuk mendeteksi korban kecelakaan laut?
2. Bagaimana system navigasi *autonomous* yang digunakan untuk melakukan proses pencarian korban?
3. Bagaimana integrasi sistem *vision* dan sistem navigasi *autonomous* untuk melaksanakan misi pencarian korban kecelakaan laut.
4. Bagaimana evaluasi prototipe ASV dalam melaksanakan misi pencarian korban kecelakaan laut?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, di antaranya sebagai berikut.

1. Implementasi Tugas Akhir ini menggunakan bahasa pemrograman C/C++, Python, dengan menggunakan *framework Robot Operating System* (ROS).
2. Prototipe ASV yang digunakan berskala kecil milik Barunastra ITS.
3. Data diambil pada lingkungan sebenarnya di Danau 8 ITS.

1.4 Tujuan

Tujuan dari pembuatan Tugas Akhir ini adalah membuat prototipe ASV yang dapat melakukan proses pencarian korban kecelakaan laut dengan system navigasi tanpa awak.

1.5 Manfaat

Hasil dari penggerjaan Tugas Akhir ini memiliki manfaat untuk menghasilkan sebuah prototipe ASV pencari korban kecelakaan laut. Penelitian ini diharapkan menjadi awalan untuk pengembangan selanjutnya dengan skala yang lebih besar baik dari sisi *vehicle* maupun sensor yang digunakan. Sehingga dapat diimplementasikan guna membantu kinerja tim SAR dalam proses penanganan kecelakaan laut di Indonesia.

1.6 Tahapan Tugas Akhir

Tahapan-tahapan yang dilakukan dalam penggerjaan tugas akhir ini adalah sebagai berikut.

1.6.1 Penyusunan Proposal Tugas Akhir

Proposal tugas akhir ini berisi tentang deskripsi pendahuluan dari tugas akhir yang akan dikerjakan. Secara detail, proposal tugas akhir ini berisi tentang beberapa bagian yaitu latar belakang diajukannya tugas akhir, rumusan masalah yang diangkat, batasan masalah untuk tugas akhir, tujuan dari pembuatan tugas akhir, dan manfaat dari hasil pembuatan tugas akhir. Selain itu, dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan tugas akhir dan ringkasan isi yang membahas metode yang akan digunakan dalam

tugas akhir. Sub bab metodologi merupakan penjelasan mengenai tahapan penyusunan tugas akhir. Terdapat pula sub bab jadwal penggerjaan yang menjelaskan jadwal penggerjaan tugas akhir dan di akhir bagian terdapat daftar pustaka untuk mencantumkan referensi yang digunakan dalam tugas akhir.

1.6.2 Studi Literatur

Studi literatur yang dilakukan dalam penggerjaan Tugas Akhir ini adalah mencari berbagai macam referensi terkait penyelesaian masalah dalam topik yang diusulkan, yaitu pencarian korban kecelakaan laut melalui *computer vision* dengan ASV. Adapun referensi yang dimaksud berupa buku, *scientific paper*, artikel, materi kuliah, dan diskusi dengan pihak yang berpengalaman.

1.6.3 Desain dan Analisis

Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep teknologi. Tahap ini mendefinisikan alur implementasi dan langkah-langkah yang akan dikerjakan. Pada tahapan ini dilakukan desain sistem dan desain proses-proses yang ada. Tahapan dari analisis dan desain perangkat lunak yang akan dilakukan dengan tahapan sebagai berikut.

1. Pemilihan jenis *hull* ASV yang digunakan.
2. Perancangan sistem elektronik yang digunakan.
3. Pemilihan jenis arsitektur *computer vision*.
4. Desain dan uji coba system komunikasi ASV dengan *ground station*.
5. Pemilihan system navigasi yang sesuai dengan kasus terkait.
6. Perancangan *Performance Indicator* untuk menilai apakah prototipe ASV ini telah berjalan sesuai dengan yang diinginkan.

1.6.4 Implementasi Perangkat Lunak

Implementasi meliputi implementasi algoritma dan metode yang telah didukung oleh hasil analisis dan desain pada tahap sebelumnya. Implementasi ini dilakukan dengan menggunakan bahasa pemrograman C/C++ dan python yang diintegrasikan menggunakan *framework* ROS. *Hardware* yang digunakan untuk implementasi sistem meliputi:

1. *Microcontroller* Nuclueo-STM32F429ZI
2. Laptop MSI GL65 dengan RTX 2060
3. Kamera USB
4. 3 sensor *ranging* berupa SRF
5. FCU Pixhawk PX4, berisikan GNSS
6. Motor sebagai sistem aktuator penggerak ASV

1.6.5 Pengujian dan Evaluasi

Pengujian dan evaluasi dari hasil Tugas Akhir ini akan diujicobakan pada danau 8 ITS. Evaluasi dan perbaikan akan dilakukan hingga *system* mendekati performa yang diinginkan.

1.6.6 Penyusunan Buku Tugas Akhir

Pada tahapan ini, dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi prototipe ASV yang telah dibuat.

1.7 Sistematika Penulisan Buku Tugas Akhir

Buku tugas akhir ini merupakan laporan secara lengkap mengenai tugas akhir yang telah dikerjakan baik dari sisi teori, rancangan, maupun implementasi sehingga memudahkan bagi pembaca dan juga pihak yang ingin mengembangkan lebih lanjut. Sistematika penulisan buku tugas akhir secara garis besar dapat dilihat seperti dibawah ini.

Bab I Pendahuluan

Bab ini berisi penjelasan latar belakang, rumusan masalah, batasan masalah, dan tujuan pembuatan tugas akhir. Selain itu, tahapan penggerjaan tugas akhir dan sistematika penulisan laporan tugas akhir juga dijelaskan didalamnya.

Bab II Tinjauan Pustaka

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan tugas akhir ini.

Bab III Metodologi

Bab ini berisi tentang desain prototipe yang dikembangkan dalam penyelesaian permasalahan. Bab ini juga berisi analisis strategi penyelesaian permasalahan serta penjelasan implementasi berdasarkan desain yang telah dilakukan pada tahap desain.

Bab IV Hasil dan Pembahasan

Bab ini berisi pengujian dan evaluasi dari implementasi yang dilakukan menggunakan metode evaluasi yang telah ditentunkan.

Bab V Kesimpulan dan Saran

Bab ini merupakan bab terakhir yang menjelaskan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan prototipe ke depannya.

BAB 2 TINJAUAN PUSTAKA

2.1 Hasil Penelitian Terdahulu

Dalam melakukan penelitian ini, didasari oleh beberapa penelitian yang telah ada. Adapun penelitian-penelitian tersebut yang memiliki keterkaitan dengan penelitian yang penulis sedang lakukan, sehingga dapat dijadikan suatu landasan.

YOLOv3 merupakan arsitektur YOLO yang dikembangkan oleh Joseph Redmon, dkk. *Update* terhadap original YOLO memiliki tingkat akurasi yang meningkat dengan tetap mempertahankan performa *inference*-nya, sehingga dapat digunakan untuk kebutuhan *real-time detection* [6]. Dengan input 320 x 320, YOLOv3 membutuhkan waktu 22 ms dengan mAP 28.2.

Pengembangan *navigation*, *guidance*, dan *obstacle avoidance algorithm* untuk *Unmanned Surface Vehicle* (USV) menggunakan algoritma *fusion*[7]. Hal tersebut bertujuan untuk meminimalkan kesalahan navigasi saat USV berjalan secara mandiri. Sehingga, diharapkan USV mampu melakukan navigasi mandiri dengan baik.

Penelitian yang dilakukan Rui Wang, dkk. *A Real-Time Object Detector for Autonomous Vehicles Based on YOLOv4* menunjukkan bahwa *Object detection* merupakan elemen penting dalam teknologi kendaraan *autonomous* [8]. Untuk memberikan *feedback* visi yang baik, *object detection* yang digunakan harus memiliki tingkat keakuratan yang tinggi dengan performa *real-time*.

Penghitung jumlah orang berbasis visi computer juga pernah diteliti oleh In Cho S. Dalam proses *training* yang dilakukan, digunakan augmentasi untuk mengurangi *overfitting*. Penelitian ini menghasilkan luaran yang baik [3].

2.2 Dasar Teori

Bab ini berisi pembahasan mengenai teori-teori dasar yang digunakan dalam tugas akhir. Pada bab ini akan dijelaskan mengenai ASV, *Autonomous Navigation System*, *Computer Vision*, ROS. Dimana dari tinjauan tersebut, akan digunakan pada tugas akhir ini.

2.2.1 Autonomous Surface Vehicle

Autonomous Surface Vehicle (ASV; Unmanned Surface Vessels (USVs)) merupakan kapal permukaan yang mampu beroperasi tanpa awak [8]. ASV menggunakan sensor sebagai alat pengindra lingkungan sekitar. Digunakan komputer untuk mengolah data dari sensor untuk menentukan arah pergerakan ASV yang selanjutnya akan disebut *Autonomous Navigation System*.

2.2.2 Autonomous Navigation System

Navigation system merupakan sistem computer yang dapat melakukan proses navigasi tanpa awak secara otomatis. Sistem ini ditanamkan pada ASV dengan dibekali beberapa sensor posisi dan orientasi. *Global Navigation Satellite System* (GNSS) dan *Inertial Measurement Unit* (IMU) digunakan sebagai input *Navigation System* yang memberikan posisi serta keadaan vektor dari ASV. Sistem Navigasi ini memberikan output berupa lokalisasi, *map*, dan arah gerak robot [9].

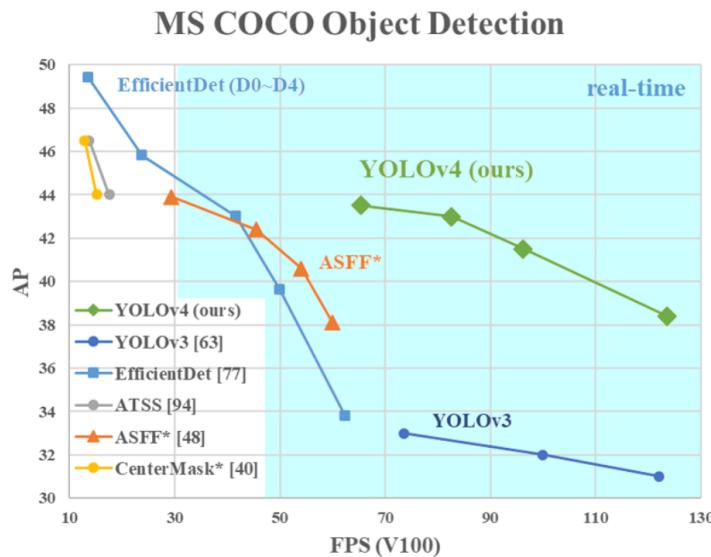
Vector Field Histogram (VFH) merupakan algoritma perencanaan gerakan (*motion planning*). VFH menggunakan representasi *statistic* dari robot terhadap lingkungan sehingga disebut *histogram grid* [10].

Proportional-Integral-Derivative (PID) diperlukan untuk mengestimasi output ke *actuator* dari sebuah error. PID merupakan mekanisme *control loop feedback* yang banyak digunakan dalam berbagai implementasi *continuously control* [11].

2.2.3 Computer Vision

Computer vision merupakan ranah dari kecerdasan buatan yang memungkinkan komputer memahami informasi dari suatu gambar digital. Berbeda dengan manusia yang lebih dulu “belajar” memahami suatu informasi, konteks, seberapa jauh suatu objek, apakah suatu objek bergerak atau diam, *computer vision* masih tergolong baru dan merupakan topik yang menarik di bidang kecerdasan buatan.

Saat ini, telah banyak dikembangkan berbagai arsitektur *Convolutional Neural Network* (CNN) untuk berbagai skenario. YOLOv4 sebagai basis arsitektur *computer vision* yang digunakan dengan berbagai kesesuaianya dengan skenario yang dihadapi [12]. YOLOv4 menawarkan kecepatan *inference* hingga dua kali lipat dibandingkan EfficientDet. Terdapat peningkatan dibandingkan YOLOv3 12% pada sektor FPS dan 10% pada sektor AP. Perbandingan lengkap dibandingkan model lainnya dapat dilihat pada gambar 2.1.

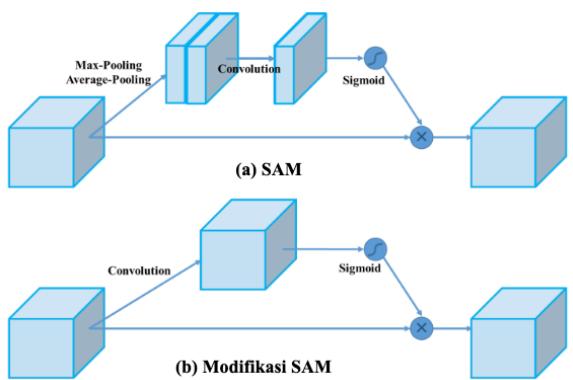


Gambar 2.1 Perbandingan YOLOv4 dan Model *Object Detector* Lainnya

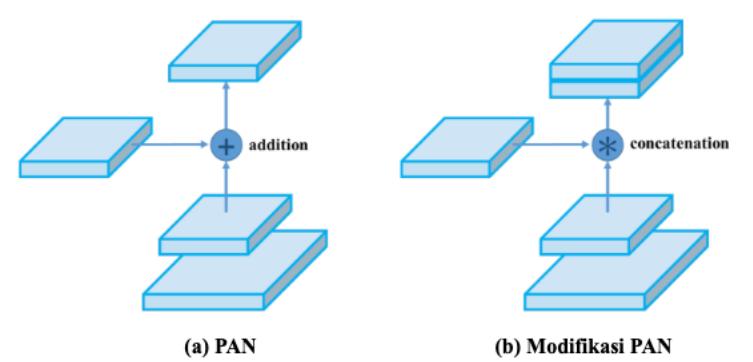
YOLOv4 berkontribusi dengan melakukan beberapa perubahan sehingga menghasilkan performa seperti yang ditunjukkan pada gambar 2.1. YOLOv4 dikembangkan dengan efisien dan *powerful* sebagai model *object detection*. YOLOv4 dapat dilatih menggunakan *graphics processing unit* (GPU) yang berbasis konsumen seperti Nvidia 1080 Ti atau 2080 Ti. YOLOv4 terdiri dari CSPDarknet53 sebagai *backbone*, SPP dan PAN sebagai *neck*, YOLOv3 sebagai *head*.

Dari hasil percobaan yang dilakukan terhadap berbagai konfigurasi *Bag of Freebies* (BoF), dipilih BoF untuk *backbone* yaitu CutMix, *Mosaic data augmentation*, DropBlock *regularization*, *Class label smoothing*. *Bag of Specials* (BoS) pada *backbone* yang digunakan adalah *Mish activation*, *Cross-stage partial connection* (CSP), *Multi-input weighted residual connections* (MiWRC). BoF untuk *detector* YOLOv4 menggunakan CIoU-loss, CmBN,

DropBlock regularization, augmentasi Mosaic, Self-Adversarial Training, Menghilangkan grid sensitivity, menggunakan beragam anchors untuk satu ground truth, Cosine annealing scheduler [13], Optimal hyperparameters, training shapes tak beraturan. BoS untuk detector digunakan Mish activation, SPP-block, SAM-block, PAN path-aggregation block, DIoU-NMS. Adapun SAM yang dimodifikasi sebagai mana gambar 2.2. Sedangkan untuk PAN yang dimodifikasi sebagaimana gambar 2.3



Gambar 2.2 Modifikasi SAM



Gambar 2.3 Modifikasi PAN

Saat ini YOLOv4 memiliki *pretrained weight* menggunakan dataset imangenet. Dataset *pretrained* tersebut memiliki kemiripan dengan dataset yang akan digunakan pada Tugas Akhir ini, sehingga dimungkinkan untuk dilakukan *transfer learning* [14].

2.2.4 TensorRT

Seiring berkembangnya Deep Learning (DL), aplikasi *inference* terus bertambah. *Embedded device* mulai muncul dengan Neural Processing Units (NPUs) sebagai tambahan atas CPU dan GPU. Untuk menghasilkan *environment* pengembangan yang efisien, TensorRT menyediakan sebuah *development kit* yang bekerja pada perangkat keras NVIDIA. TensorRT dapat melakukan optimasi yang berdampak pada berkurangnya *latency* dengan tetap mempertahankan output dari sebuah model [15]. Pada umumnya DL *frameworks* akan berjalan pada GPU atau NPU saja, TensorRT menawarkan sebuah metode *parallelization* sehingga mampu *me-utilize* GPU dan NPU secara bersamaan. Dari hasil percobaan yang dilakukan, tensorRT mampu menghasilkan peningkatan 80% – 391%.

2.2.5 Communication

Sistem komunikasi yang digunakan berbasis radio (915Mhz) dan WiFi (2.4Ghz). Sistem komunikasi menggunakan radio lebih unggul dari segi daya dan *range*, namun terbatas oleh jumlah data yang dapat dikirimkan. Sedangkan WiFi, memiliki *stream data* yang lebih tinggi namun dengan *cost* daya yang lebih tinggi. Dalam penggunaan WiFi, *protocol* yang dipilih adalah UDP dikarenakan memberikan kemampuan *real-time* [16].

2.2.6 ROS

Robot Operating System (ROS) merupakan *middleware open-source* yang digunakan untuk merancang sistem perangkat lunak *robotic*. Meskipun ROS bukan merupakan *operating system* melainkan merupakan *frameworks*. ROS memiliki kemampuan komunikasi *node* berbasiskan jaringan sehingga pertukaran data lebih mudah untuk ditata dan dikelola[17].

2.2.7 Evaluasi

Untuk dapat mengukur performa dari prototipe yang dihasilkan, diperlukan skema evaluasi. Evaluasi terdiri dari sisi visi computer dan sistem navigasi.

2.2.8 Evaluasi Visi Komputer

Evaluasi untuk mengetahui bagaimana visi komputer yang digunakan telah mencukupi untuk melakukan identifikasi korban. Evaluasi yang digunakan berupa *mean Average Precision* (mAP). mAP digunakan untuk menentukan akurasi dari hasil deteksi *object detection* yang diprediksi oleh sebuah model dibandingkan *ground-truth* [18].

Intersection over Union (IoU) digunakan dalam perhitungan mAP. IoU mempunyai nilai 0 sampai 1 yang merepresentasikan banyaknya *overlap* antara *bounding box* yang diprediksi oleh model dengan *ground-truth*. Adapun persamaan 2.1 yang digunakan untuk menghitung IoU.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} \quad (2.1)$$

- Nilai IoU = 0, tidak ada *overlap* antara dua *bounding box* tersebut.
- Nilai IoU = 1, dua *bounding box* saling bertumpuk secara sempurna.

2.2.9 Evaluasi Navigasi

Sistem navigasi dapat dievaluasi dengan merekam *path* yang dilalui oleh ASV lalu dibandingkan dengan *path* yang diinginkan. Dengan menggunakan persamaan 2.2.

$$\text{error} = \text{heading}_{\text{error}} + \text{perpindicular}_{\text{distance}_{\text{error}}} \quad (2.2)$$

BAB 3 METODELOGI PENELITIAN

Bab ini menjelaskan mengenai desain mekanikal, desain elektrikal, dan arsitektur perangkat lunak dari prototipe ASV pencari korban kecelakaan laut.

3.1 Desain Mekanikal ASV

Perancangan perangkat keras mekanikal bertujuan untuk menciptakan *platform* untuk meletakkan keseluruhan sistem sehingga dapat beroperasi di atas permukaan air. Desain mekanikal ini terdiri dari lambung dan sistem propulsi ASV.

3.1.1 Lambung ASV

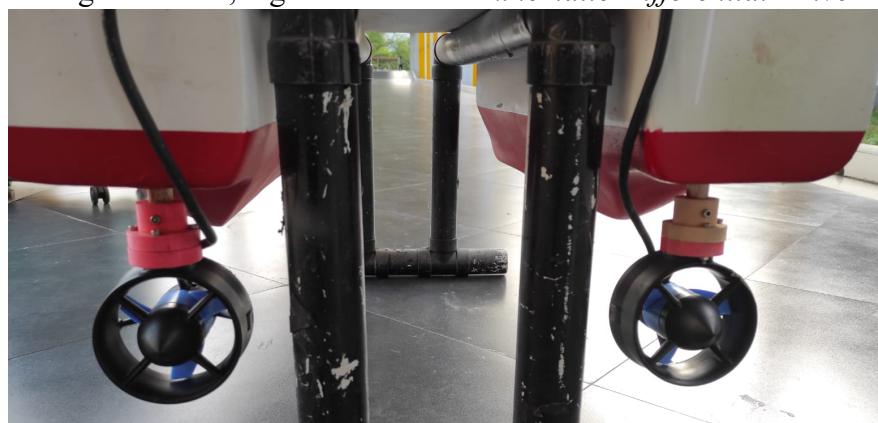
Lambung kapal didesain untuk memenuhi kebutuhan masalah dalam pencarian korban kecelakaan laut. Dipilih desain lambung katamaran karena memiliki kestabilan yang lebih tinggi dan hambatan air yang lebih kecil apabila dibandingkan dengan tipe lambung *monohull* [19]. Lambung kapal menggunakan bahan *carbon-fibre glass*, sehingga memberikan karakteristik kapal yang kuat dan ringan. Adapun desain prototipe yang digunakan ditampilkan pada gambar 3.1.



Gambar 3.1 Desain Prototipe ASV

3.1.2 Propulsi ASV

Propulsi yang digunakan pada ASV adalah dua buah *electric thruster* T200 BlueRobotics yang dipasang pada bagian belakang ASV. *Thruster* tersebut dipasang dengan konfigurasi *fix mounting*, sehingga posisinya tetap tidak dapat bergerak, diperlihatkan pada gambar 3.2. Untuk melakukan gerak belok, digunakan metode *kinematic Differential Drive*.



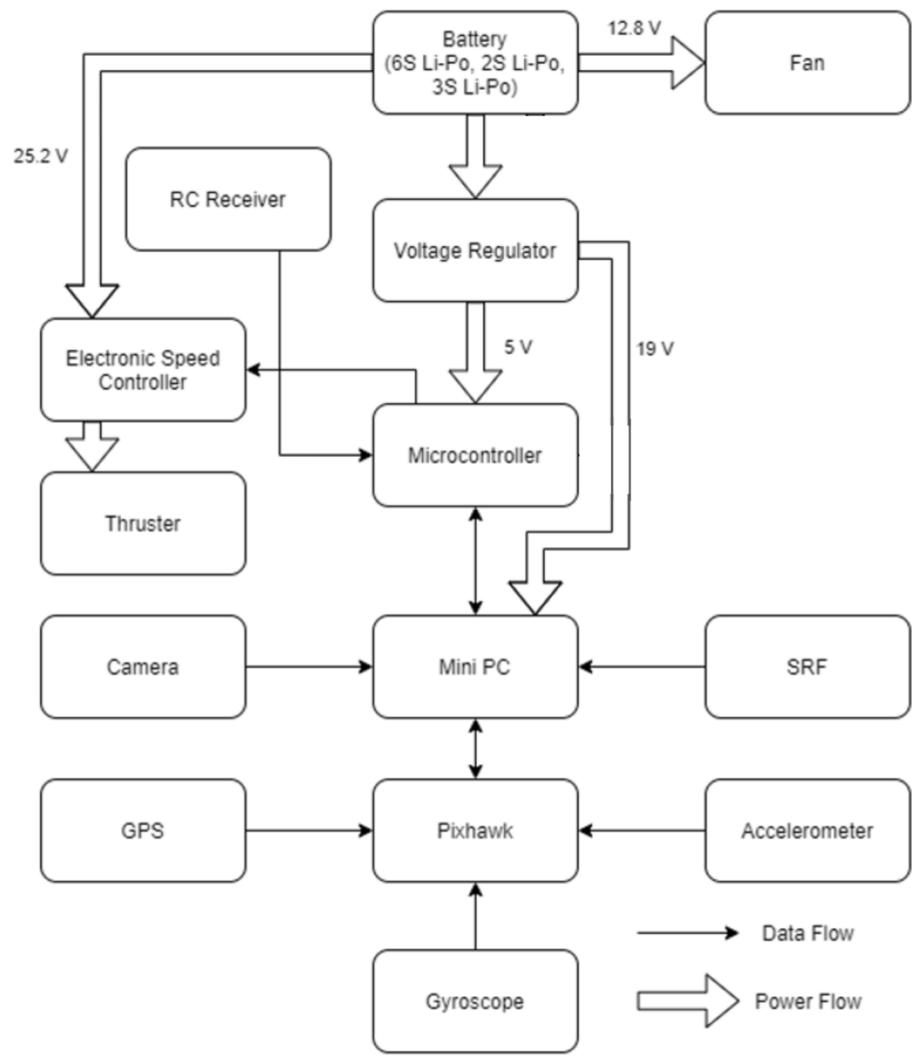
Gambar 3.2 *Thruster* T200 BlueRobotics yang Dipasang dengan Konfigurasi *Fix Mounting*

3.2 Desain Elektrikal ASV

Perancangan perangkat keras elektrikal bertujuan untuk menciptakan sistem daya, *control actuator*, dan antarmuka komunikasi komputer ke aktuator.

3.2.1 Sistem Daya dan Jalur Data

Untuk memenuhi kebutuhan daya untuk semua komponen elektrik yang ada pada ASV, diperlukan sistem yang dapat mengatur kebutuhan tegangan untuk setiap komponen yang ada. Pada ASV ini, terdapat sumber daya dari baterai yang kemudian masuk melalui regulator dan kemudian digunakan untuk menyalaikan komponen-komponen yang dibutuhkan. Untuk melakukan pertukaran data baik digital maupun analog, digunakan *Printed Circuit Board* (PCB) dan perkabelan antar komponen yang berkomunikasi. Rangkaian sistem daya dan jalur data digambarkan pada gambar 3.3.

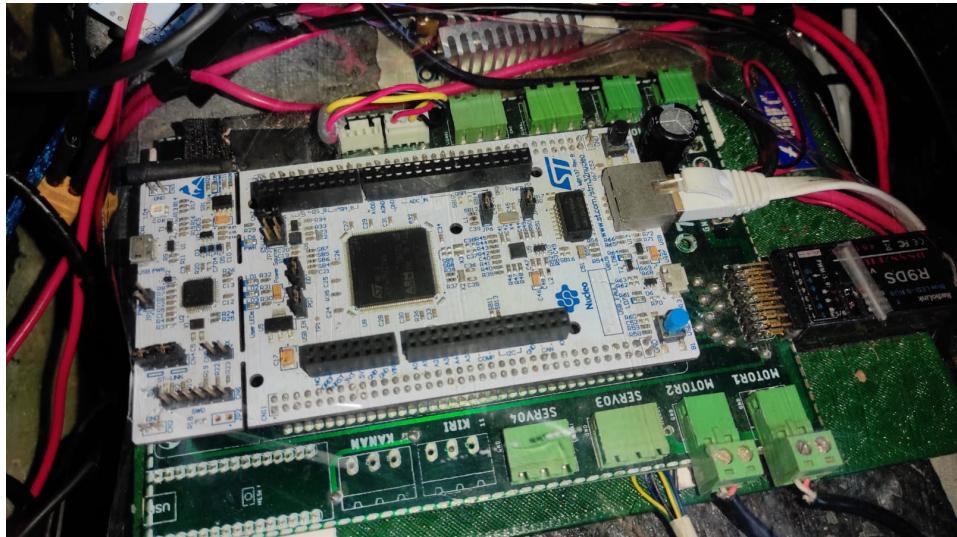


Gambar 3.3 Diagram Aliran Daya dan Aliran Data

3.2.2 Microcontroller Nucleo-STM32F429ZI

STM32F429ZI menjadi prosessor utama untuk menangani *interface* dengan komponen *actuator*. Mikrokontroler ini juga menangani komunikasi data dari *remote control* melalui *receiver*. Sehingga, apabila ada hal yang tidak diinginkan, operator dapat melakukan *take over* sebagai prosedur keselamatan. Nucleo-STM32F429ZI juga dilengkapi dengan kemampuan

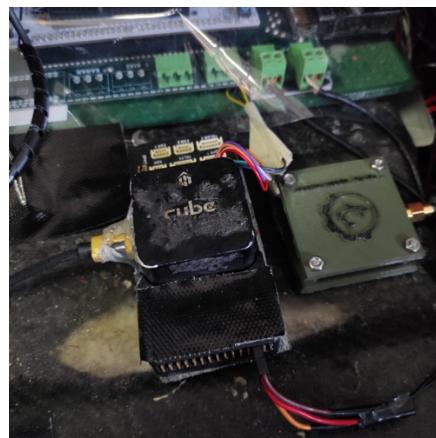
komunikasi melalui *ethernet*, sehingga komunikasi dari dan ke *computer* dapat lebih *reliable* menggunakan *protocol* UDP. Selanjutnya kaki dari mikrokontroler ini dipasangkan pada *Printed Circuit Board* (PCB) yang telah didesain untuk kebutuhan yang ada, diperlihatkan pada gambar 3.4.



Gambar 3.4 Nucleo-STM32F429ZI dengan PCB

3.2.3 Pixhawk Cube

Pixhawk merupakan *flight controller unit* yang umumnya digunakan pada *drone*. Pixhawk memiliki berbagai sensor seperti *Accelerometer* sebagai *Inertial Measuring Unit* (IMU), *gyroscope*, *compass*, barometer, dan *external GPS*. Sensor – sensor tersebut dibutuhkan untuk mengestimasi posisi. Selain itu, Pixhawk memiliki *firmware* yang dilengkapi dengan estimator seperti *Extended Kalman Filter* (EKF) sehingga pembacaan dari sensor yang ada menjadi minim *noise*. Adapun Pixhawk dan *external GPS* yang digunakan, diperlihatkan pada gambar 3.5.



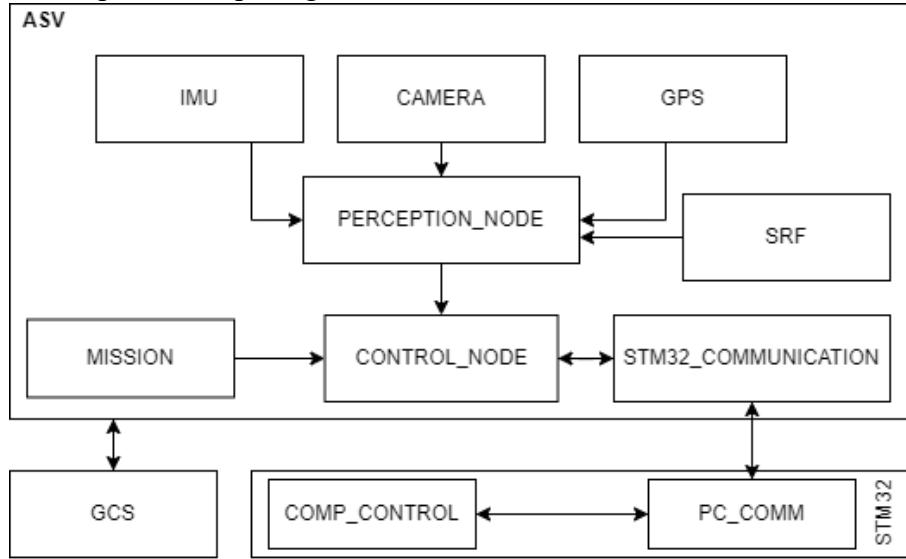
Gambar 3.5 Pixhawk Cube dengan External GPS

3.2.4 Ultrasonic Sensor

Digunakan *ultrasonic sensor* untuk mengetahui jarak dari suatu benda yang ada di depan ASV. *Ultrasonic sensor* yang digunakan adalah USR-100. Data jarak yang didapatkan dari sensor ini kemudian dapat dijadikan referensi oleh *program control* untuk menghindari halangan di depan ASV.

3.3 Desain Arsitektur Perangkat Lunak ROS

Pada bagian perangkat lunak, digunakan *framework* ROS menggunakan bahasa C/C++ dan python. Arsitektur dapat dilihat pada gambar 3.6.

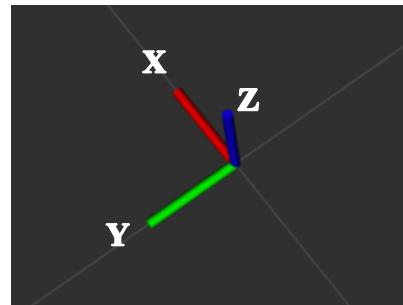


Gambar 3.6 Arsitektur ROS Node

Arsitektur ASV terdiri dari beberapa sensor yang di-*handle* oleh *node* yang bersangkutan. Sensor kamera masuk dalam node CAMERA. Node CAMERA memberikan luaran berupa posisi lokasi dari korban kecelakaan dengan menggunakan visi computer. Digunakan perangkat Pixhawk yang selanjutnya akan diproses melalui mavros oleh node IMU dan GPS. GPS node bertanggung jawab untuk memproses sensor navigasi posisi, sehingga output dari node ini berupa lokasi ASV. Sementara, node IMU memberikan output berupa orientasi ASV berupa *roll*, *yaw*, *pitch*, dan akselerasi. Node SRF bertugas untuk memberikan output berupa hasil pembacaan dari beberapa sensor *ranging*. Output dari node sensor selanjutnya menjadi input untuk node PERCEPATION_NODE. Pada node ini, informasi dari sensor dibentuk menjadi sebuah persepsi. Persepsi tersebut akan digunakan oleh CONTROL_NODE sebagai acuan kondisi lingkungan yang dihadapi ASV. CONTROL_NODE bertugas untuk melakukan pengambilan keputusan sesuai dengan misi yang sedang dilakukan. Untuk berkomunikasi dengan aktuator, terdapat STM32_COMMUNICATION yang terhubung dengan kabel *ethernet* RJ45. GCS merupakan *Ground Control Station* atau *Basestation* yang digunakan untuk *monitoring* dan mengatur misi yang sedang dijalankan oleh ASV.

3.3.1 PERCEPTION_NODE

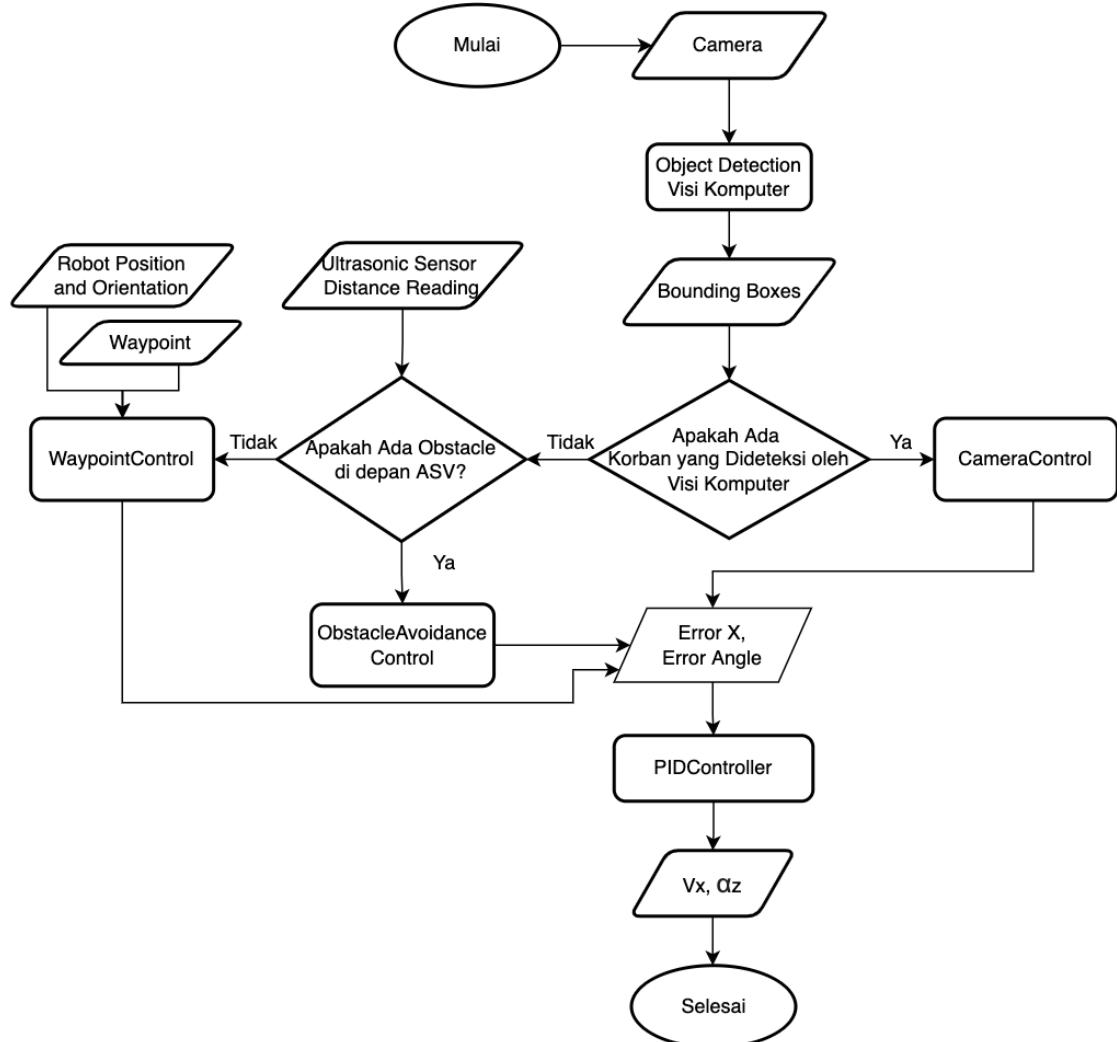
Merupakan *node* yang bertugas untuk mem-*publish* TF dan beberapa *element marker* dari sensor untuk memberikan gambaran atau persepsi pada *program control*. Pada PERCEPTION_NODE menghasilkan orientasi TF seperti gambar 3.7, x+ didefinisikan depan ASV, y+ didefinisikan kiri ASV, dan z+ didefinisikan diatas ASV.



Gambar 3.7 Orientasi TF

3.3.2 CONTROL_NODE

Pada dasarnya kontrol navigasi mengimplementasikan *flowchart* pada gambar 3.8 dalam menjalankan misi pencarian korban kecelakaan laut. Untuk melakukan navigasi tanpa awak, beberapa algoritma navigasi digunakan. Dalam node CONTROL_NODE, terdiri atas dua control navigasi utama yaitu *global control* dan *local control*. Terdapat PID Controller yang digunakan untuk mengestimasi output V_x , α_z dari error yang diperoleh dari perhitungan *control*.



Gambar 3.8 Flowchart CONTROL_NODE

3.3.2.1 Local Control

Local control digunakan saat ASV mendapatkan *local feature* dari lingkungan. *Feature* tersebut berupa korban yang terdeteksi oleh *node CAMERA* dan jarak dari suatu *object* yang terdeteksi oleh *node SRF*. Saat *local control* digunakan, maka *CONTROL_NODE* akan melepas referensi *global* dan beralih ke referensi *local*. Hal ini bertujuan untuk meningkatkan keakuratan *program control*, dikarenakan referensi *global* memiliki data yang *noise*. *Local control* dibagi menjadi dua yaitu saat menemukan korban (*CameraControl*) dan saat mendapati halangan (*ObstacleAvoidanceControl*).

a. CameraControl

CameraControl menggunakan output deteksi dari *node CAMERA*. Output dari *node CAMERA* memiliki *property center_x*, *center_y*, lalu *CameraControl* akan mencoba mendekati dan mengarahkan ASV ke target (korban) dengan rumus error,

$$error_y = center_x - 0.5 \quad (3.1)$$

$$error_x = critline - center_y \quad (3.2)$$

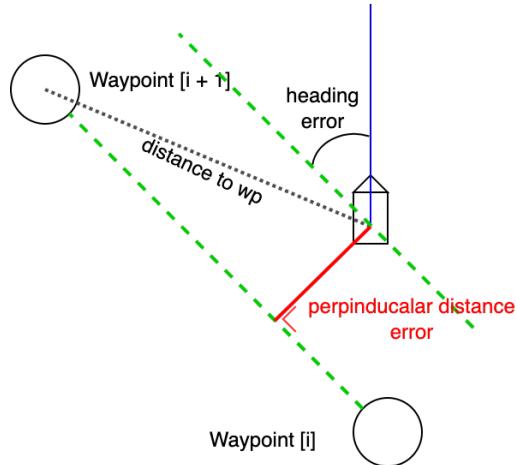
error_y menghasilkan gerak belok, ASV berusaha menghadap ke korban. 0.5 merupakan bagian tengah dari *frame* kamera dengan asumsi hasil deteksi telah dinormalisasi dengan lebar dan tinggi *frame*. *Error_x* menghasilkan gerak maju mundur, *critline* merupakan variable berupa posisi y dimana posisi ASV akan dipertahankan terhadap korban.

b. ObstacleAvoidanceControl

ObstacleAvoidanceControl menggunakan data *ranging* dari *node SRF* yang kemudian digunakan sebagai input pada algoritma *Obstacle Avoidance*. Output dari *control* ini berupa *error_x* dan *error_y*. Sensor *ultrasonic* dipasang dengan sudut a,b,c terhadap ASV. Adapun algoritma yang digunakan adalah dengan memilih diantara sudut a,b,c yang memiliki area kosong atau aman. Dikarenakan sensor *ultrasonic* ini rentan terhadap *noise*, ditambahkan sebuah *filter* sederhana untuk memastikan kebenaran datanya.

3.3.2.2 Global Control

Global control digunakan saat ASV tidak mendapatkan *feature* dari lingkungan. *Feature* tersebut berupa korban yang terdeteksi oleh *node CAMERA* dan jarak dari suatu *object* yang terdeteksi oleh *node SRF*. *Global control* menggunakan GPS dan *Compass* sebagai referensi kontrol. Pada *control* ini terdapat WaypointControl dimana berperan untuk mengarahkan kapal menuju titik yang ditentukan. Adapun rumus error yang digunakan dapat dilihat pada persamaan 3.3. Adapun dijelaskan melalui gambar 3.9 mengenai representasi error yang dimaksud.



Gambar 3.9 Ilustrasi Error WaypointControl

$$error = heading_{error} + perpendicular\ distance_{error} \quad (3.3)$$

Adapun kondisi dimana *waypoint* telah tercapai apabila kondisi 3.4 dan kondisi 3.4 terpenuhi.

$$distance\ to\ wp < threshold\ distance \quad (3.4)$$

$$angle\ to\ wp > threshold\ angle \quad (3.5)$$

3.3.2.3 PID Controller

ASV memiliki *thruster* dengan konfigurasi *differential drive*. Untuk melakukan gerak, ASV membutuhkan nilai Vx, dan Az. Nilai tersebut diestimasikan menggunakan PID dengan input nilai error. Adapun PID yang dimaksud merupakan persamaan 3.6.

$$output = (P * error) + (I * error_{accumulative}) + (D * (error - error_{before})) \quad (3.6)$$

3.4 Computer Vision

Digunakan *computer vision* untuk melakukan tugas deteksi korban kecelakaan laut yang mengapung di permukaan air. Dipilih YoloV4 sebagai arsitektur CNN karena memiliki kemampuan deteksi yang cukup baik, baik objek yang berukuran besar maupun kecil. Digunakan darknet untuk melakukan pelatihan. Dalam proses pelatihan digunakan *pretrained weights* yang sebelumnya telah dilatih pada *dataset* MS COCO, hal ini bertujuan untuk mengurangi waktu *train*. Input size dari arsitektur yang dipilih berukuran 416 x 416 px. Adapun kamera yang digunakan adalah Logitech C930 dengan resolusi 640 x 480. *Dataset* yang digunakan dibuat mandiri secara manual dengan mengambil gambar orang tenggelam (mengapung) di permukaan air. *Dataset* tersebut kemudian diaugmentasi dengan *property exposure*, mengingat dalam implementasi dimungkinkan adanya perbedaan intensitas cahaya karena perubahan cuaca. Setelah proses pelatihan selesai dilakukan, *weights* dari darknet akan diubah ke dalam bentuk tensorRT untuk menunjang performa waktu *inference*.

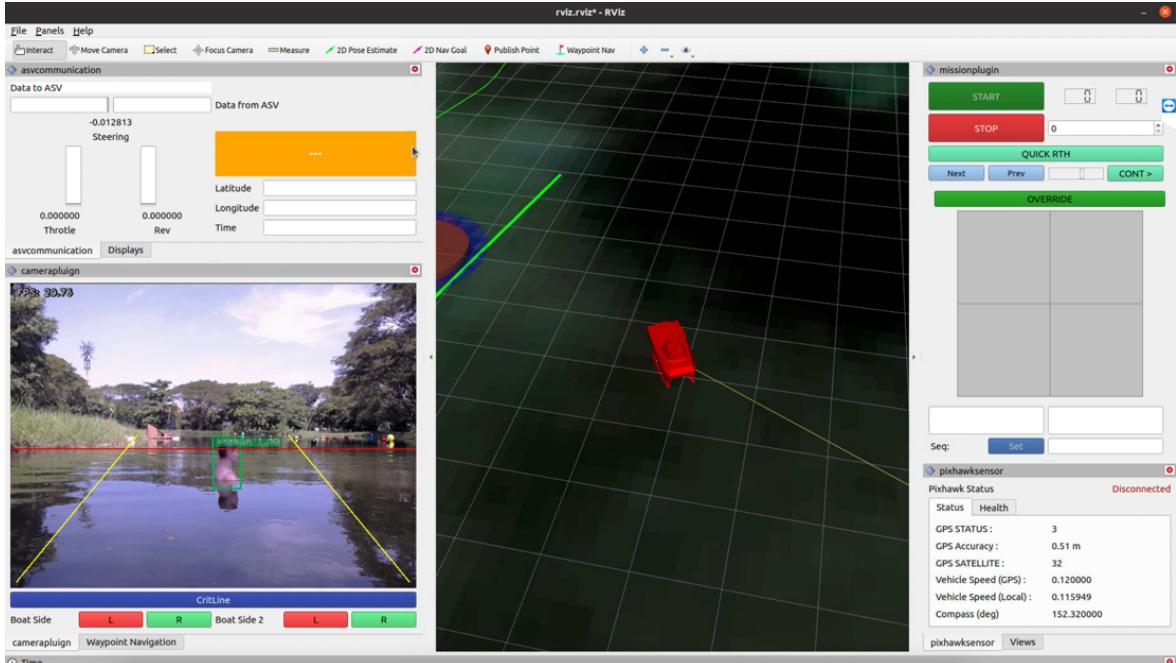
3.5 Antarmuka Monitoring dan Tuning

Untuk memudahkan proses evaluasi dan *tuning* untuk penyesuaian parameter, dibuat antarmuka dengan menggunakan rviz dan Qt. Marker yang dipublish oleh *node*

PERCEPTION NODE akan divisualisasikan di rviz. Parameter yang akan *detuning* ditampilkan dan dapat diubah pada antarmuka Qt yang langsung terhubung CONTROL_NODE.

3.6 Implementasi PERCEPTION_NODE

Pada PERCEPTION_NODE semua data sensor diolah dan divisualisasikan pada rviz seperti pada gambar 3.11. PERCEPTION_NODE pada implementasinya terdiri dari beberapa sub *node* yaitu asv_tf, mavros, map_image, rviz_plugin. Output *node* ini berupa persepsi dari lingkungan sekitar yang selanjutnya dapat diolah oleh CONTROL_NODE dan dapat dimonitor.



Gambar 3.10 Antarmuka PERCEPTION_NODE rviz

3.6.1 asv_tf

asv_tf bertanggung jawab untuk mem-*publish* TF dengan informasi yang didapat dari *node* mavros berupa posisi *latitude*, *longitude*, dan *compass*. Mavros merupakan ROS package yang mengimplementasikan mavlink sehingga dapat berkomunikasi dengan Pixhawk. Untuk implementasi dapat dilihat pada kode sumber 3.1.

1	static tf2_ros::TransformBroadcaster br;
2	geometry_msgs::TransformStamped transformStamped;
3	transformStamped.header.stamp = ros::Time::now();
4	transformStamped.header.frame_id = "map";
5	transformStamped.child_frame_id = "asv/odom";
6	transformStamped.transform.translation.x = LAT_TO_METER*(lat_center - lat_);
7	transformStamped.transform.translation.y = LON_TO_METER*(long_center - long_);
8	transformStamped.transform.translation.z = 0.0;
9	tf2::Quaternion q;
10	q.setRPY(0, 0, local_yaw);
11	transformStamped.transform.rotation.x = q.x();
12	transformStamped.transform.rotation.y = q.y();

13	<code>transformStamped.transform.rotation.z = q.z();</code>
14	<code>transformStamped.transform.rotation.w = q.w();</code>
15	<code>br.sendTransform(transformStamped);</code>

Kode Sumber 3.1 TF Publisher

3.6.2 map_image

map_image bertanggung jawab untuk mem-*publish* marker berupa gambar *satellite*, untuk memudahkan pengaturan perimeter. Dikarenakan batasan rviz, maka untuk menampilkan gambar di dalam rviz perlu dilakukan penyesuaian. Cara yang dipilih adalah dengan cara melakukan iterasi tiap *pixel* dari sebuah gambar, lalu diambil nilai RGB nya untuk kemudian dijadikan dua segitiga *marker*. Dari kumpulan *marker* segitiga yang mewakili *pixel – pixel* gambar tersebut kemudian digabung dan di-*publish*. Untuk implementasi yang dilakukan dapat dilihat pada kode sumber 3.2.

1	<code>visualization_msgs::Marker image;</code>
2	<code>image.type = visualization_msgs::Marker::TRIANGLE_LIST;</code>
3	<code>geometry_msgs::Point p;</code>
4	<code>std_msgs::ColorRGBA crgb;</code>
5	<code></code>
6	<code>for(int r = 0; r < src.rows; ++r) {</code>
7	<code> for(int c = 0; c < src.cols; ++c) {</code>
8	<code> cv::Vec3b intensity = src.at<cv::Vec3b>(r, c);</code>
9	<code> crgb.r = intensity.val[2] / 255.0;</code>
10	<code> crgb.g = intensity.val[1] / 255.0;</code>
11	<code> crgb.b = intensity.val[0] / 255.0;</code>
12	<code> crgb.a = 1.0;</code>
13	<code></code>
14	<code> p.z = -0.1;</code>
15	<code> p.x = (LUCorner.first + r * pix_row) * -1;</code>
16	<code> p.y = LUCorner.second + c * pix_col;</code>
17	<code> image.points.push_back(p);</code>
18	<code> image.colors.push_back(crgb);</code>
19	<code> p.x = (LUCorner.first + (r + 1) * pix_row) * -1;</code>
20	<code> p.y = LUCorner.second + c * pix_col;</code>
21	<code> image.points.push_back(p);</code>
22	<code> image.colors.push_back(crgb);</code>
23	<code> p.x = (LUCorner.first + r * pix_row) * -1;</code>
24	<code> p.y = LUCorner.second + (c + 1) * pix_col;</code>
25	<code> image.points.push_back(p);</code>
26	<code> image.colors.push_back(crgb);</code>
27	<code> p.x = (LUCorner.first + (r + 1) * pix_row) * -1;</code>
28	<code> p.y = LUCorner.second + c * pix_col;</code>
29	<code> image.points.push_back(p);</code>
30	<code> image.colors.push_back(crgb);</code>
31	<code> p.x = (LUCorner.first + (r + 1) * pix_row) * -1;</code>
32	<code> p.y = LUCorner.second + (c + 1) * pix_col;</code>
33	<code> image.points.push_back(p);</code>
34	<code> image.colors.push_back(crgb);</code>
35	<code> p.x = (LUCorner.first + r * pix_row) * -1;</code>
36	<code> p.y = LUCorner.second + (c + 1) * pix_col;</code>
37	<code> image.points.push_back(p);</code>
38	<code> image.colors.push_back(crgb);</code>
39	<code> }</code>
40	<code>}</code>
41	<code>return image;</code>

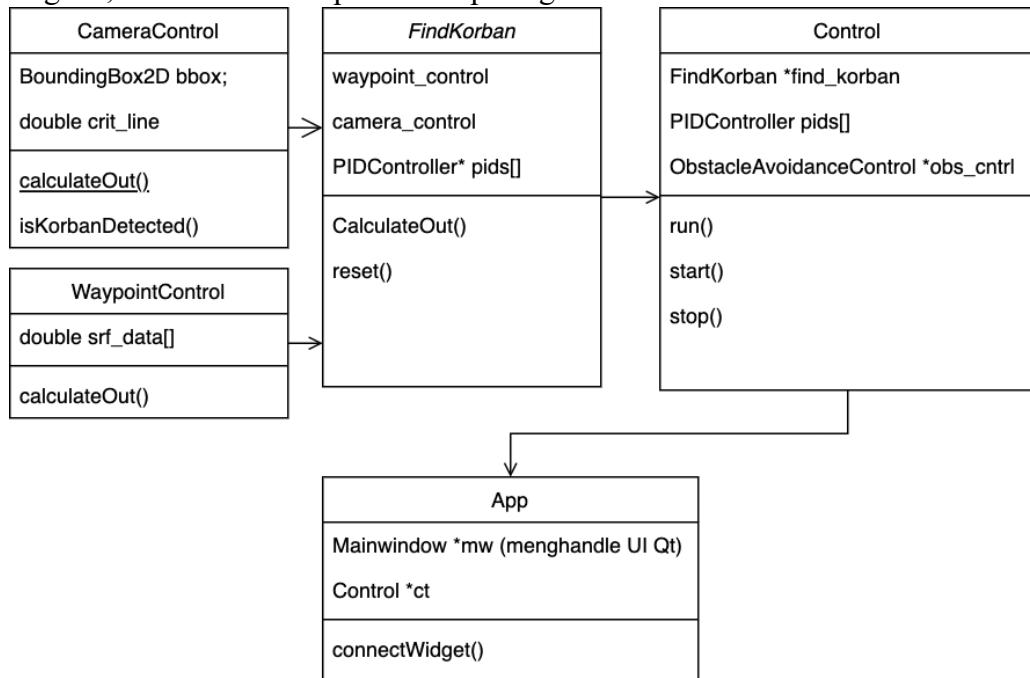
Kode Sumber 3.2 map_image Publish Satelite Image ke rviz

3.6.3 rviz_plugin

node rviz_plugin bertanggung jawab dalam menyajikan antarmuka atas data-data yang diterima dan dikeluarkan dari CONTROL_NODE. Dengan digunakannya antarmuka keadaan yang sedang dialami ASV, memudahkan *monitoring* dan evaluasi performa dalam menjalankan misi. Adapun beberapa rviz plugin yang dibuat untuk memenuhi kebutuhan tersebut. asvCommunication menyajikan data output *control* dari CONTROL_NODE menuju STM32COMMUNICATION. CameraPlugin menyajikan output *image* dari CAMERA dan menerima input berupa parameter *crit_line* (akan dibahas pada sub-bab berikutnya). MissionPlugin merupakan *plugin* yang ditujukan untuk pusat kendali jalannya misi, memulai misi, menghentikan misi, dan melakukan *take over*. PixhawkSensor merupakan *plugin* yang menyajikan data jumlah *satellite* GPS yang terkoneksi, kecepatan, dan kondisi sensor. WaypointNavTool merupakan *plugin* yang dibuat untuk melakukan pengaturan *waypoint* atau perimeter, kemana ASV akan mencari korban kecelakaan laut.

3.7 Implementasi CONTROL_NODE

CONTROL_NODE merupakan *node* utama dalam system navigasi *autonomous* yang dikembangkan, struktur kelas dapat dilihat pada gambar 3.11.



Gambar 3.11 Struktur Kelas CONTROL_NODE

Adapun implemenetasi **state machine** utama CONTROL_NODE dapat dilihat pada kode sumber 3.3.

1	ros::Rate rr(50);
2	while (ros::ok())
3	{
4	listenASVTF();
5	if(mission_state.data[0] == 1){
6	//start mission
7	out_cmd = find_korban->calculateOut();

```

8     if(out_cmd.linear.z != 1){ //bukan control camera
9         mission_status_msg.data = "Waypoint Control";
10        mission_status_string_pub.publish(mission_status_msg);
11        geometry_msgs::Twist obs_cmd;
12        obs_cmd = obstacle_avoid_control->calculateOut();
13        if(obs_cmd.linear.x != 0 && obs_cmd.angular.z != 0){
14            out_cmd = obs_cmd;
15            std::cout << "MENHINDARRR\n";
16            mission_status_msg.data = "SRF Control";
17            mission_status_string_pub.publish(mission_status_msg);
18        }
19    }else{
20        mission_status_msg.data = "Camera Control";
21        mission_status_string_pub.publish(mission_status_msg);
22    }
23 }else if(mission_state.data[0] == 0){//stop mission
24     find_korban->stop();
25     if(is_test_motor){
26         out_cmd.linear.x = thrust_trim;
27         out_cmd.angular.z = steer_trim;
28         sendCmdVel();
29     }
30 }
31 if(mission_state.data[0] != 0){
32     out_cmd.angular.z += steer_trim;
33     sendCmdVel();
34 }
35 rr.sleep();
36 ros::spinOnce();
37 }

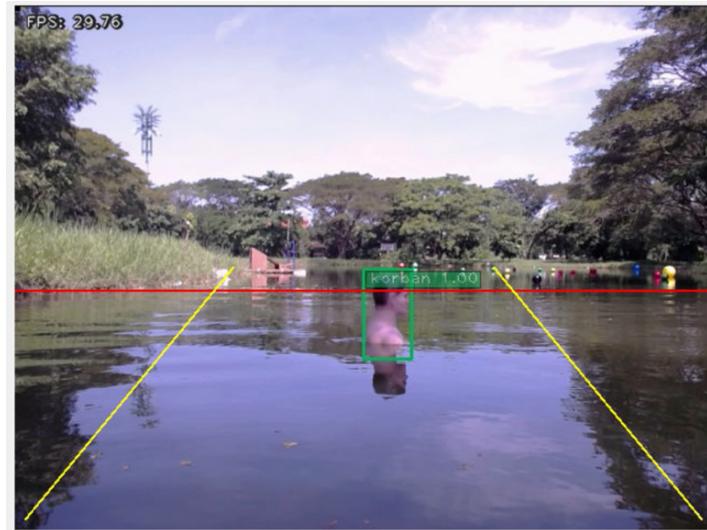
```

Kode Sumber 3.3 State Machine CONTROL_NODE

3.7.1 Local Control

3.7.1.1 CameraControl

Pada CameraControl terdapat variabel berupa crit_line yang didefinisikan sebagai garis horizontal berwarna merah. Crit-line (divisualisasikan dengan garis horizontal warna merah pada gambar 3.12) merupakan target titik *control_error_x*, dimana CameraControl akan berusaha mempertahankan posisi korban untuk berada pada *center_y = crit_line*. Sedangkan *error_y* diperoleh dari selisih jarak *center_x* terhadap titik tengah *frame* yaitu 0.5. Kedua error tersebut (*error_x* dan *error_y*) kemudian masuk kedalam PIDController. Adapun implementasi dari CameraControl dapat dilihat pada kode sumber 3.4.



Gambar 3.12 Visualisasi CameraControl

```

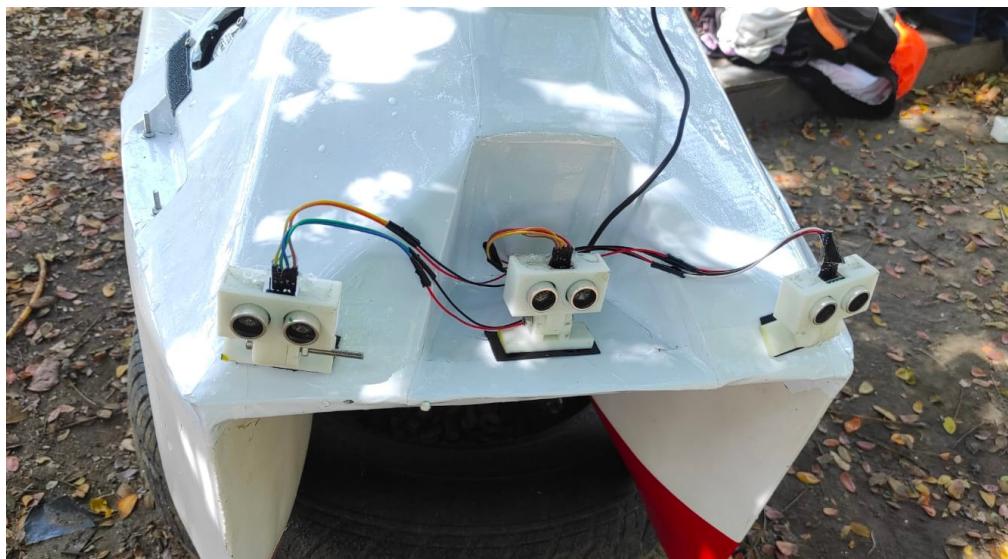
1 geometry_msgs::Twist out_cmd;
2
3 if(obj_person_detected.boxes.size()){
4     double error_cam_x = 0.5 - obj_person_detected.boxes[0].center.x;
5     double error_cam_y = crit_line - obj_person_detected.boxes[0].center.y;
6
7     out_cmd.angular.z = pid_x->updateError(error_cam_x);
8     out_cmd.linear.x = pid_y->updateError(error_cam_y);
9 }else if(confident_counter < confident_threshold){ //sebelumnya ada korban
10    out_cmd.angular.z = 0;
11    out_cmd.linear.x = 0;
12    std::cout << "MENUNGGU KORBAN GLITCH " << "\n";
13
14 }else if(try_rotate_in_position){ //try rotate to find korban
15    out_cmd.angular.z = 0.3;
16    out_cmd.linear.x = 0;
17    if(start_angle == NOT_CONDUCTED)
18        start_angle = ct->getRobotTf().yaw;
19    if(start_angle > ct->getRobotTf().yaw && ct->getRobotTf().yaw >
   (start_angle - (M_PI/18)) ){
20        try_rotate_in_position--;
21    }
22
23    std::cout << "ROTATING UNTIL " << start_angle << "\n";
24 }else{
25
26 }
27
28 return out_cmd;

```

Kode Sumber 3.4 Implementasi CameraControl

3.7.1.2 ObstacleAvoidanceControl

Untuk menghindari *obstacle* di depan ASV digunakan ObstacleAvoidanceControl. *Control* ini mengambil input dari tiga sensor *ultrasonic* yang disusun dengan sudut aproksimasi -20 derajat, 0 derajat, 20 derajat terhadap x+ positif ASV, dapat dilihat pada gambar 3.13. Implementasi yang dilakukan dapat dilihat pada kode sumber 3.5.



Gambar 3.13 Penempatan Sensor Ultrasonic pada ASV

```
1 for (int i = 0; i < SRF_NUMBER; i++){
2     free_gap[i] = true;
3     if(obs_gap_confident[i] > 0)
4         free_gap[i] = false;
5
6     early_free_gap[i] = true;
7     if(early_obs_gap_confident[i] > 0)
8         early_free_gap[i] = false;
9 }
10
11 bool any_gap = false;
12 bool all_gap_free = true;
13 bool all_early_gap_free = true;
14 for (int i = 0; i < SRF_NUMBER; i++){
15     any_gap |= free_gap[i];
16     all_gap_free &= free_gap[i];
17     all_early_gap_free &= early_free_gap[i];
18 }
19
20 std::vector<std::pair<int, int> > sorted_distance;
21 sorted_distance = ct->sortArr(srf_data, SRF_NUMBER);
22 if (!any_gap){
23     //mundur tidak ada jalan di depan
```

```

24     out_cmd.linear.x = - ct->speedControlCalculate(-1);
25     return out_cmd;
26 }else if(all_gap_free){
27     if(all_early_gap_free){
28         is_there_is_obstacle = false;
29         return out_cmd;
30     }else{
31         //early obstacle control
32         double error_mpx = 150 - sorted_distance[0].first;
33         int choosen_gap_idx = sorted_distance[SRF_NUMBER-1].second;
34         int avoided_gap_idx = sorted_distance[0].second;
35         int steer_angle = srf_data_angle[choosen_gap_idx]
36                         - srf_data_angle[avoided_gap_idx]/2;
37         std::cout << error_mpx << " " <<
38             choosen_gap_idx << " " << steer_angle << "\n" ;
39
40         out_cmd.linear.x = ct->speedControlCalculate(1);
41         out_cmd.angular.z = -((steer_angle * (error_mpx/4)) / 1000)
42             * pid_angle->getP();
43
44     return out_cmd;
45 }
46 }
```

Kode Sumber 3.5 Implementasi *Obstacle Avoidance* Menggunakan Tiga Sensor Ultrasonic

3.7.2 Global Control

Pada jenis *control global*, digunakan GPS dan *compass* sebagai referensi *control*. Dengan input berupa *waypoint* dimana ASV harus mencapai titik tersebut. Adapun implementasi dapat dilihat pada kode sumber 3.6.

```

1 bool WaypointControl::isArrivedPath() {
2     geometry_msgs::Point controlled_path_point =
3         main_path.poses[path_idx].pose.position;
4
5     double delta_x = ct_->getRobotTf().pos.x - controlled_path_point.x;
6     double delta_y = ct_->getRobotTf().pos.y - controlled_path_point.y;
7     double angle = utils::radToDeg(
8         utils::getAngle(ct_->getRobotTf().pos, controlled_path_point)
9         - utils::getAngle(controlled_path_point_before,
10            controlled_path_point)
11    );
12
13    if (fabs(angle) >= 180){
14        angle = 360 - fabs(angle);
15    }
16 }
```

14	}
15	
16	return !(fabs(angle) < path_angle_limit and pow(delta_x, 2) + pow(delta_y, 2) > pow(path_keep_distance,2));
17	
18	geometry_msgs::Twist output;
19	
20	if (isArrivedPath()){
21	path_idx++;
22	path_idx_before = path_idx - path_length_trace;
23	if(path_idx_before < 0) {
24	path_idx_before = 0;
25	}
26	
27	double angle_path = utils::getAngleInvert(main_path.poses[path_idx_before].pose.position,
28	main_path.poses[path_idx].pose.position);
29	path_error_angle = utils::radToDeg(angle_path - ct_->getRobotTf().yaw);
30	
31	if (path_error_angle < -180) path_error_angle += 360; //waypoint behind robot heading
32	else if (path_error_angle > 180) path_error_angle -= 360;
33	
34	path_error_dist = distanceLineToPoint(getInitPathPoint(), getNextPathPoint(), ct_->getRobotTf().pos);
35	
36	error_acc_angle_path = path_error_angle + path_error_angle_before;
37	error_acc_dist_path = path_error_dist + path_error_dist_before;
38	
39	path_controlDistanceOut = pid_distance->updateError(path_error_dist);
40	path_controlAngleOut = pid_angle->updateError(path_error_angle);
41	
42	double result = path_controlAngleOut + path_controlDistanceOut;
43	output.angular.z = result;
44	return output;

Kode Sumber 3.6 Implementasi *Global Control*

3.7.3 PIDController

Untuk mengestimasi output berupa Vx dan Az dari nilai error yang dihasilkan oleh *program control*, diperlukan PID *Controller*. Dibuat kelas PIDController dikarenakan PID ini digunakan secara berulang berulang dengan jumlah *instance* yang banyak. Dalam PID memiliki *variable* p, i, d, dan error. PID Controller diimplementasikan dengan kode sumber 3.7.

1	double PIDController::updateError(double error_now_){
2	error_now = error_now_;
3	
4	if (use_acc_error)

```

5     error_acc += error_now + error_before;
6     else
7         error_acc = error_now + error_before;
8
9     out_now = (double)(p * error_now + i * (error_acc)
10        + d * (error_now - error_before));
11     error_before = error_now;
12     return out_now;

```

Kode Sumber 3.7 Implementasi PID Controller

3.8 Implementasi *Computer Vision*

Untuk melakukan deteksi korban digunakan CNN dengan arsitektur YOLOv4. CNN tersebut sebagai *computer vision* yang dimiliki ASV.

3.8.1 Pembuatan Dataset

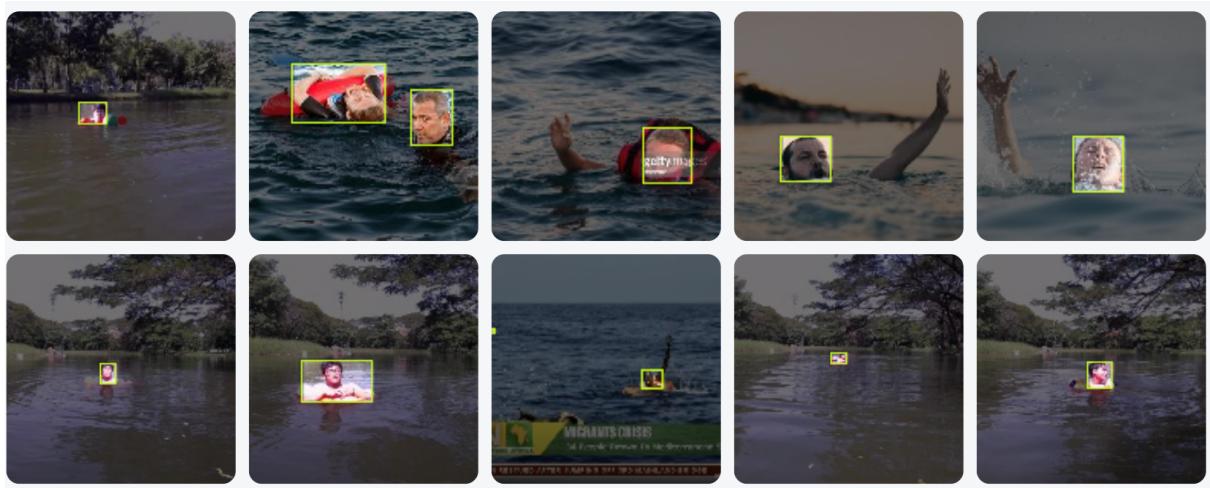
Dataset dibuat dengan anotasi manual menggunakan roboflow.com. Dataset terdiri foto orang mengapung di danau 8 ITS. *Dataset* ini diambil menggunakan kamera Logitech C930 yang terpasang pada bagian atas kapal. *Sample dataset* dapat dilihat pada gambar 3.14. *Dataset* ini kemudian diaugmentasi dengan variasi *brightness*. Terdapat 172 data gambar dengan jumlah anotasi sebanyak 249, selanjutnya dipisahkan 142 gambar sebagai *data training*, dan 30 gambar sebagai data validasi. Dari 142 gambar *training*, dilakukan augmentasi hingga menjadi 426 gambar.



Gambar 3.14 Dataset yang Digunakan dalam Proses *Training*

Adapun disiapkan *data test* yang selanjutnya digunakan untuk mengevaluasi kinerja model yang telah dibuat. Data *test* ini dibuat dengan perpaduan orang tenggelam (mengapung) di

danau 8 dan di luar danau 8 yang didapatkan dari internet. Jumlah *data test* adalah 36 gambar dengan 49 anotasi. Adapun *sample* dari data *test* dapat dilihat pada gambar 3.15.



Gambar 3.15 Data *test* yang Dibuat di luar *Dataset Training*

3.8.2 *Training Model*

Digunakan darknet sebagai program *training model* yolov4 [20]. Adapun arsitektur yang dipilih yolov4 dengan penyesuaian jumlah *layer* yang bersangkutan dengan kelas yang akan dilakukan pelatihan. Pada *layer* yolo, mengubah jumlah *classes* menjadi 1. Jumlah *filter* sebelum *layer* yolo diubah menjadi 18. *Steps* diubah menjadi 4800 dengan *max batches* 5400. *Training* yang dilakukan menggunakan GPU Nvidia P100.

3.8.3 Konversi Model ke TensorRT

Untuk dapat menggunakan Nvidia TensorRT, diperlukan konversi model darknet “yolo.weight” menjadi format .trt. Digunakan kode sumber yang dikembangkan oleh jkjung, tensorrt_demos [21]. Awalnya format “darknet.weight” diubah menjadi *Open Neural Network Exchange*, selanjutnya .onnx diubah menjadi .trt. Pada kode sumber TensorRT ini dilakukan penyesuaian pada bagian *inference*-nya sehingga dapat memenuhi kebutuhan CONTROL_NODE. Adapun penyesuaian yang dilakukan antara lain membuat *publisher image* dan *publisher bounding box object*. Implementasi dari penyesuaian tersebut dapat dilihat pada kode sumber 3.8 untuk mem-*publish* informasi obyek yang terdeteksi dan kode sumber 3.9 untuk mem-*publish* *image*.

1	boxes, confs, clss = trt_yolo.detect(img, conf_th)
2	bb_msg = BoundingBox2DArray()
3	bb_msg.header.stamp = rospy.Time.now()
4	bb_msg_raw = BoundingBox2DArray()
5	bb_msg_raw.header.stamp = rospy.Time.now()
6	
7	bbox_sorted = list(boxes)
8	bbox_sorted.sort(key = lambda x : x[3], reverse=True)
9	
10	for i in bbox_sorted:
11	x_min = min(max(0,i[0]),camera_res_w)
12	x_max = min(max(0,i[2]),camera_res_w)

13	y_min = min(max(0,i[1]),camera_res_h)
14	y_max = min(max(0,i[3]),camera_res_h)
15	bb_ins = BoundingBox2D()
16	bb_ins.center.x = (x_max + x_min) / 2
17	bb_ins.center.y = (y_max + y_min) / 2
18	bb_ins.size_x = abs(x_max - x_min)
19	bb_ins.size_y = abs(y_max - y_min)
20	
21	bb_msg_raw.boxes.append(bb_ins)
22	
23	bb_ins_raw = BoundingBox2D()
24	bb_ins_raw.center.x = bb_ins.center.x / camera_res_w
25	bb_ins_raw.center.y = bb_ins.center.y / camera_res_h
26	bb_ins_raw.size_x = bb_ins.size_x / camera_res_w
27	bb_ins_raw.size_y = bb_ins.size_y / camera_res_h
28	bb_msg.boxes.append(bb_ins_raw)
29	
30	objPublisher.publish(bb_msg)

Kode Sumber 3.8 Implementasi *Publisher Bounding Box Hasil Deteksi YOLO*

1	def compress_thread():
2	global img_global, img_raw_global, bridge, imgCompressedPublisher, imgCompressedRawPublisher
3	while(not rospy.is_shutdown()):
4	imgMsgComp = bridge.cv2_to_compressed_imgmsg(img_global)
5	imgCompressedPublisher.publish(imgMsgComp)
6	
7	imgRawComp = bridge.cv2_to_compressed_imgmsg(img_raw_global)
8	imgCompressedRawPublisher.publish(imgRawComp)
9	
10	time.sleep(0.01)

Kode Sumber 3.9 Implementasi *Publisher Image*

3.9 Implementasi STM32_COMMUNICATION

Digunakan UDP untuk melakukan pertukaran data antara *microcontroller* STM32 dan *computer*. Kedua *device* tersebut dihubungkan menggunakan kabel *ethernet* RJ45. Dengan penggunaan *kinematic differential drive*, maka nilai Az akan dijumlahkan ke masing-masing nilai *actuator*. Adapun implementasi dapat dilihat pada kode sumber 3.10.

1	void ConnectionUDP::controlCallback(const geometry_msgs::Twist::ConstPtr& msg){
2	motorL = motorR = msg->linear.x*500 + 1500;
3	
4	motorL += msg->angular.z *500;
5	motorR -= msg->angular.z *500;
6	
7	write_to_udp(motorL, motorR);
8	}

```

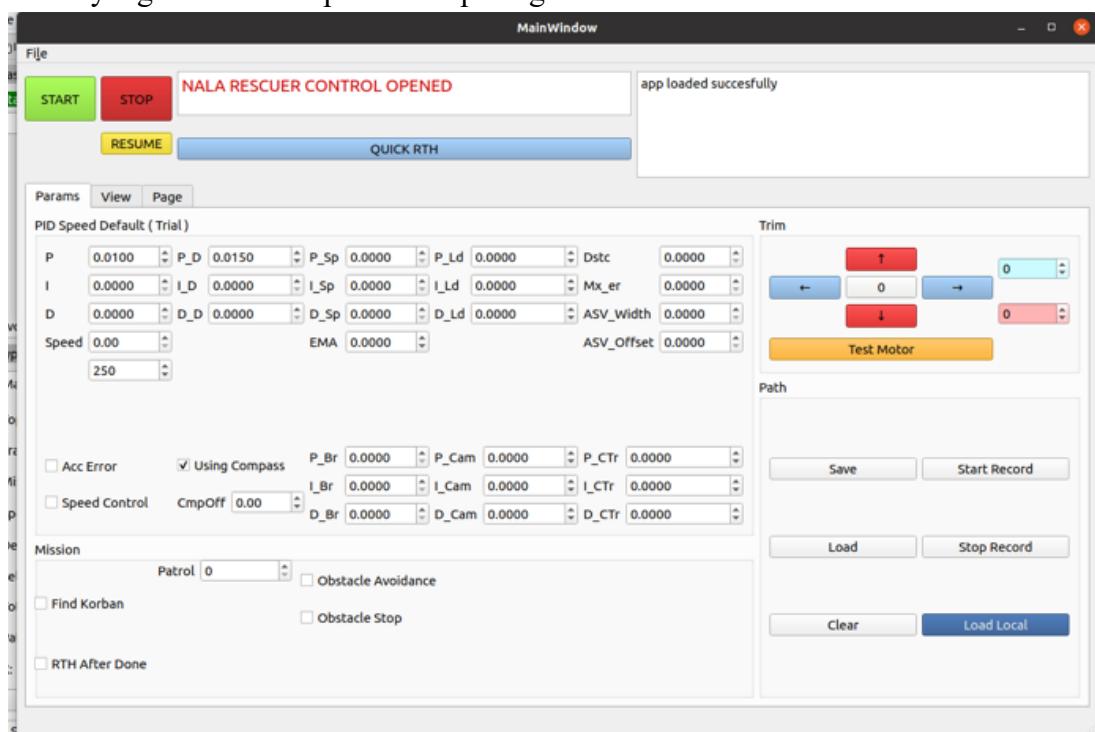
9
10 void ConnectionUDP::write_to_udp(int motor_L, int motor_R, int servo_L, int
servo_R, int tekin_L, int tekin_R)
11 {
12     short int motorL_ = motor_L;
13     short int motorR_ = motor_R;
14
15     serial_tx[0] = 'i';
16     serial_tx[1] = 't';
17     serial_tx[2] = 's';
18     memcpy(serial_tx + 3, &motorR_, 2);
19     memcpy(serial_tx + 5, &motorL_, 2);
20
21     tx_len = sendto(sockfd_rx, serial_tx, 24, 0, (struct sockaddr*)&client_addr,
sizeof(client_addr));

```

Kode Sumber 3.10 Implementasi Komunikasi UDP ke Microcontroller STM32

3.10 Implementasi Antarmuka Tuning

Untuk memudahkan proses *tuning* PID dibuat sebuah antarmuka berbasis Qt. Adapun antarmuka yang dimaksud dapat dilihat pada gambar.



Gambar 3.16 Tampilan Antarmuka berbasis Qt untuk Kebutuhan Tuning Parameter

BAB 4 HASIL DAN PEMBAHASAN

Pada bab ini dijelaskan tentang uji coba dan analisis dari implementasi yang telah dilakukan pada tugas akhir ini.

4.1 Lingkungan Uji Coba

Lingkungan uji coba yang digunakan untuk uji performa prototipe ASV merupakan perairan tenang. Dipilih danau yang berlokasi di depan Fakultas Teknik Elektro, Danau 8, gambar *satellite* dapat dilihat pada gambar 4.1. Dalam pengujian ini digunakan prototipe ASV berjenis *catamaran* dengan dua *thruster* T200 sebagaimana dijelaskan pada bagian 3.1, prototipe yang digunakan dapat dilihat pada gambar 4.2.



Gambar 4.1 Peta Satelit dari Danau 8 ITS



Gambar 4.2 Prototipe ASV saat Melakukan Uji Coba di Danau 8 ITS

Adapun perangkat keras yang digunakan sebagai pemroses perangkat lunak dan sensor pendukung lainnya memiliki spesifikasi sebagaimana dijelaskan pada tabel 4.1.

Tabel 4.1 Spesifikasi Perangkat Keras yang Digunakan oleh Perangkat Lunak

No	Nama Perangkat	Spesifikasi
1.	Komputer	MSI GL-63 : Intel Core i7-8750H, RTX 2060 Mobile, 16GB Ram
2.	Microcontroller	Nucleo-STM32F429ZI : ARM® 32-bit Cortex® -M4 CP, 180 MHz CPU
3.	Kamera	Logitech C930e : 1080p, 30fps, 90° dFoV
4.	CPE	Ubiquity Bullet M5 : 5Ghz Wifi frequency
5.	<i>Flight Control Unit</i>	Pixhawk Cube 2.1 : Gyro, IMU, Compass
6.	GPS	U-blox F9P
7.	Sensor Ultrasonic	US-100

Lingkungan uji coba pada bagian perangkat lunak memiliki spesifikasi yang dijelaskan pada tabel 4.2.

Tabel 4.2 Spesifikasi Lingkungan Perangkat Lunak

No	Nama	Spesifikasi
1.	Sistem Operasi Komputer	Linux Ubuntu 20.04
2.	<i>Robot Operating System</i>	Noetic
3.	CMake	Version >=3.2.0
4.	C/C++	C++11
5.	Python	Python3.7
6.	Qt	Qt5
7.	GPU Driver	CUDA 11.1, CuDNN 8, Driver 436.

4.2 Skenario Uji Coba dan Analisis

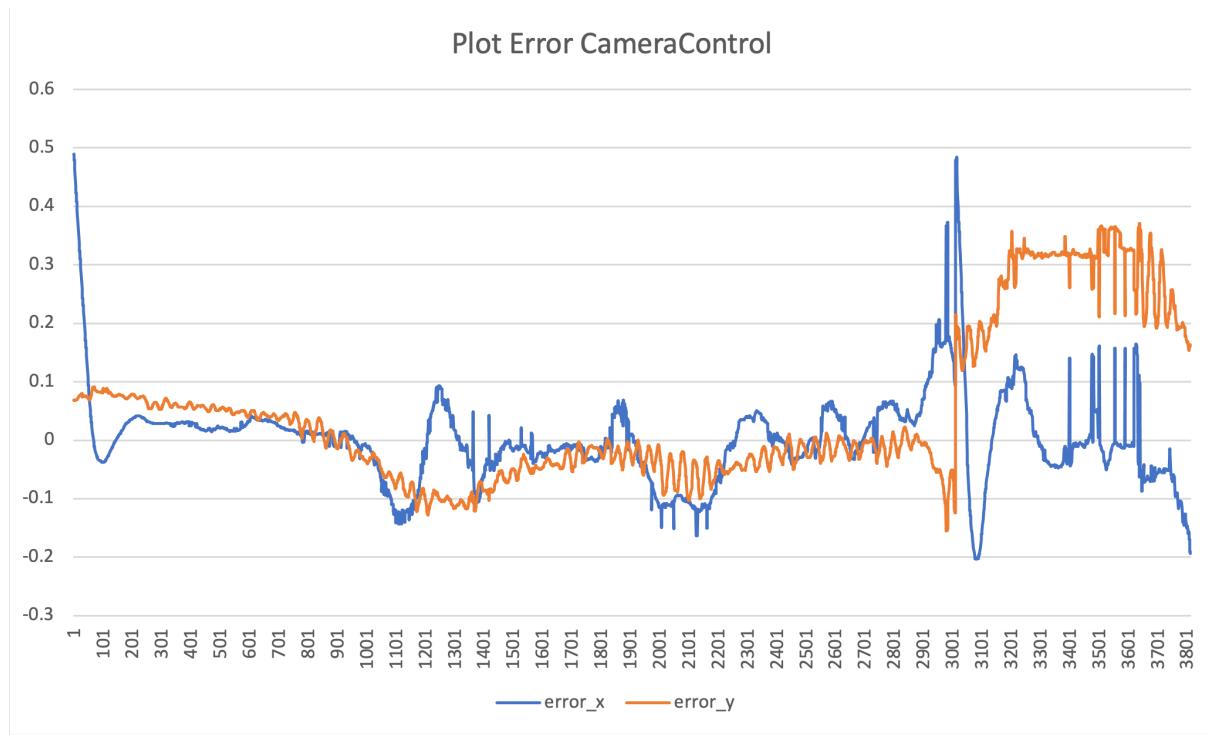
Pada subbab ini akan dijelaskan mengenai skenario – skenario pengujian untuk mengetahui performa dari prototipe *Autonomous Surface Vehicle* pencari korban kecelakaan laut yang diusulkan. Pengujian ini meliputi pengujian algoritma *local control*, *global control*, *computer vision*, dan uji keberhasilan secara kesuluruan dalam menjalankan misi.

4.2.1 Local Control

Pengujian ini dilakukan dengan melakukan beberapa kali *run* dengan kondisi yang berbeda – beda. Adapun *local control* ini kemudian dibagi menjadi dua sub uji coba yaitu uji coba CameraControl dan ObstacleAvoidanceControl.

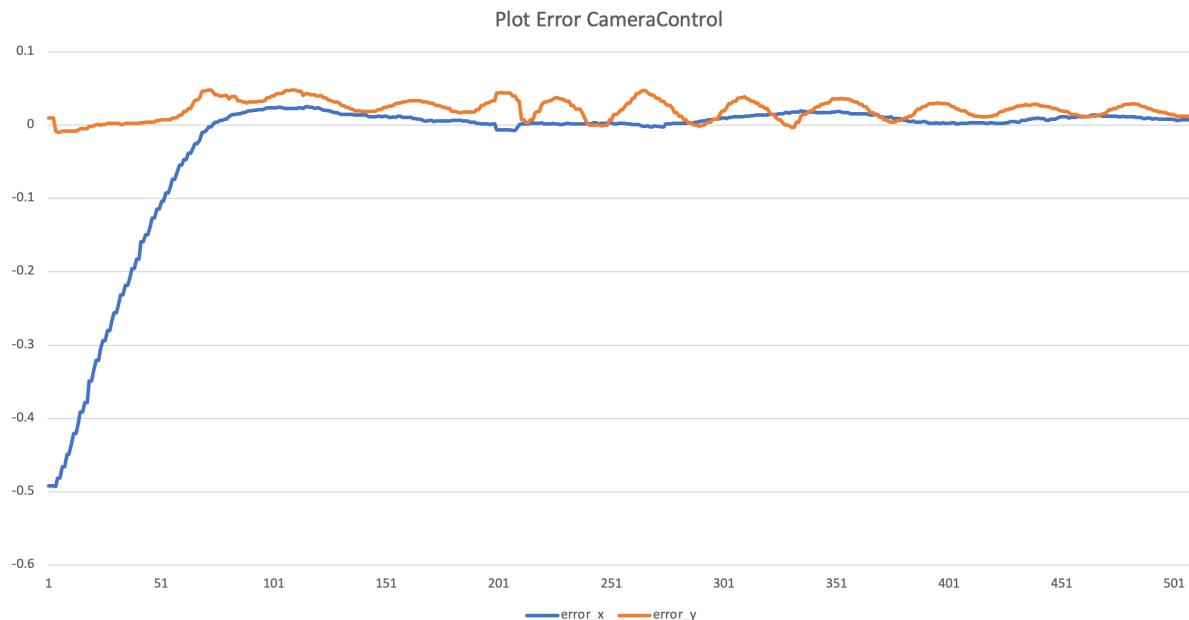
4.2.1.1 CameraControl

CameraControl menggunakan deteksi dari *computer* sebagai input dan crit_line sebagai parameter *control*. Uji coba CameraControl dilakukan diatas air pada danau 8 ITS. Adapun error_x dan error_y yang didapatkan dari hasil percobaan diperlihatkan pada gambar 4.3.



Gambar 4.3 Grafik Error CameraControl

Dari hasil data pada gambar 4.3 didapatkan dengan kondisi korban yang bergerak – gerak. Dapat diketahui terjadi sedikit osilasi pada `error_x`, dikarenakan CameraControl mengejar posisi target yang nilainya berubah – ubah, namun dapat dikatakan masih dalam *range* yang baik (osilasi di sekitar 0.1). Dapat dilihat bahwa `error_y` atau error yang menjadi referensi ASV untuk maju atau mundur terlihat lama untuk konvergen, namun lebih stabil. Hal ini disebabkan karena PID yang digunakan cukup kecil. Digunakan PID yang kecil bertujuan agar gerak ASV perlahan mendekat ke korban. Ditakutkan apabila gerak ASV terlalu cepat dan terjadi kesalahan deteksi, maka korban akan tertabrak. Pada *axis* x, data ke 3001 dicoba dilakukan pengubahan `crit_line` menjadi `crit_line + 0.3`, didapatkan `error_y` menuju kovergen di angka 0.3.



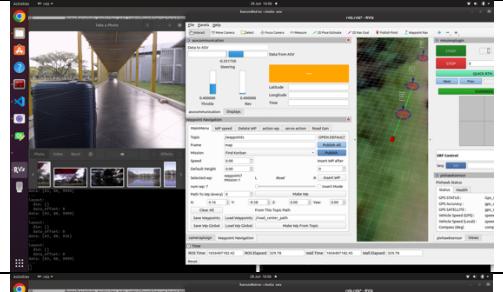
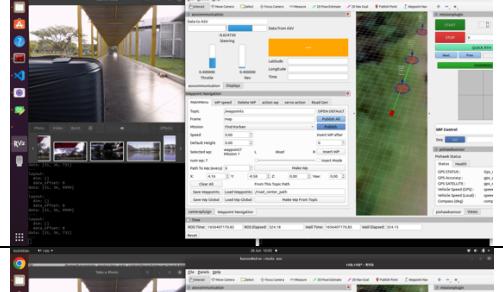
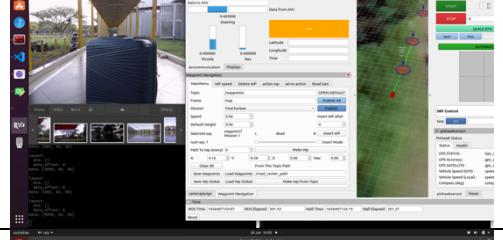
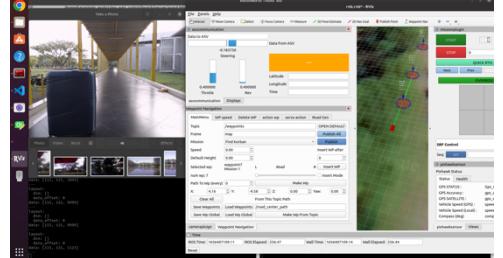
Gambar 4.4 Grafik Error CameraControl dengan Kondisi Korban Cenderung Diam

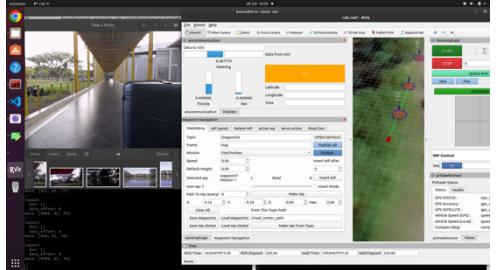
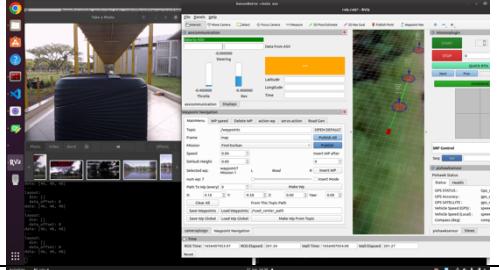
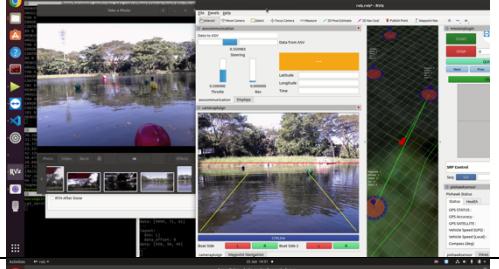
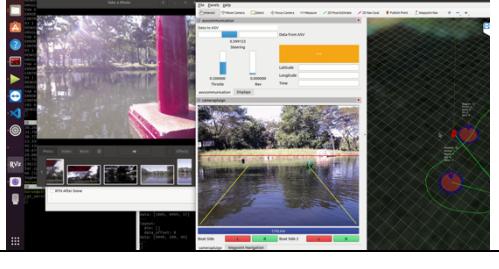
Data grafik yang ditunjukkan oleh gambar 4.4 didapatkan apabila korban tidak terlalu bergerak, sehingga errornya cenderung cepat konvergen dan tidak terjadi osilasi. Dari data tersebut, dinilai CameraControl mampu mendekat dan mengarahkan ASV menuju korban.

4.2.1.2 ObstacleAvoidanceControl

ObstacleAvoidanceControl menggunakan data dari tiga sensor *ultrasonic* sebagai input jarak *obstacle* di depan ASV. Dari tiga data jarak tersebut, kemudian dilakukan kalkulasi sehingga dihasilkan output berupa arah gerak kapal Az untuk menghindari *obstacle*. Adapun dari beberapa percobaan yang dilakukan di darat dan air dapat dilihat pada tabel 4.3.

Tabel 4.3 Percobaan *Obstacle Avoidance* Menggunakan Sensor *Ultrasonic*

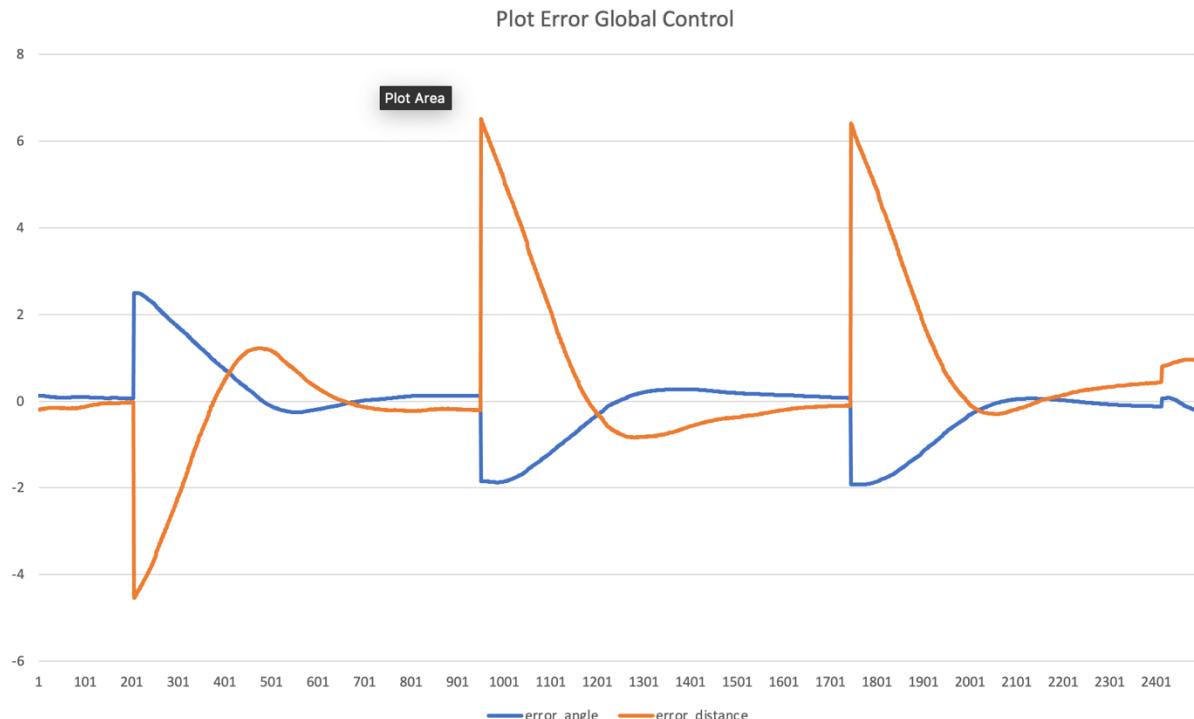
No	Input Data <i>Ultrasonic</i> dengan sudut (-20, 0, 20)	Output	Tangkap Layar
1.	(83, 88, 9999)	$Vx = 0.4$ $Az = -0.351750$	
2.	(-31, 36, 731)	$Vx = 0.4$ $Az = -0.624750$	
3.	(9999, 64, 58)	$Vx = 0.4$ $Az = 0.483000$	
4.	(115, 121, 1123)	$Vx = 0.4$ $Az = -0.183750$	

5.	(9999, 82, 76)	$Vx = 0.4$ $Az = 0.387779$	
6.	(46, 48, 48)	$Vx = -0.4$ $Az = 0$	
7	(958, 58, 49)	$Vx = 0.4$ $Az = 0.350983$	
8	(9999, 599, 49)	$Vx = 0.4$ $Az = 0.399123$	

Dari hasil percobaan baik di darat maupun di air, didapatkan hasil yang baik. ASV mampu menghindari obstacle. Namun, terdapat kelemahan dari sensor *ultrasonic* ini, saat ASV melaju dengan kecepatan $Vx > 0.5$, maka ASV akan telat merepson *obstacle*. Keterlambatan merespon ini diakibatkan dari kurangnya *rate* pembacaan atas sensor *ultrasonic* yang digunakan, sehingga ObstacleAvoidanceControl hanya mampu memberikan output *control* yang baik pada kecepatan rendah.

4.2.2 Global Control

Pada pengujian *global control* digunakan metode evaluasi yang telah dijelaskan pada bagian 2.7. Diambil satu data dari hasil percobaan *global control* untuk selanjutnya dianalisis. Data disajikan dalam bentuk grafik seperti pada gambar 4.5.



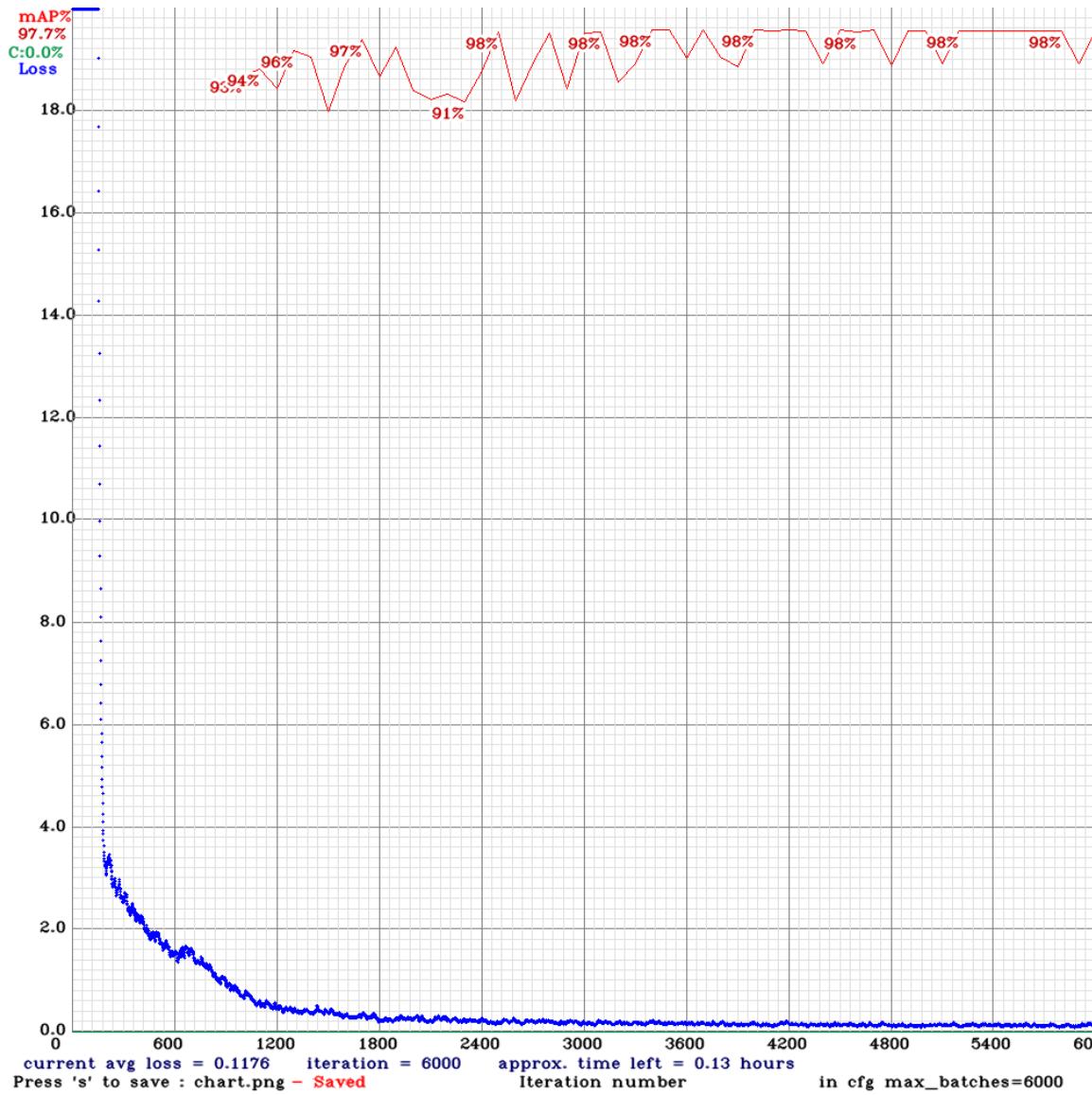
Gambar 4.5 Grafik Error *Global Control*

Dapat dilihat bahwa data *error_distance* dan *error_angle* cukup cepat konvergen. Namun pada data *error sudut*, didapatkan *error sudut* mengalami *steady state* error. *Steady state* error dari *error sudut* dimungkinkan disebabkan oleh *noise* yang ada pada *compass*. *Compass* menggunakan medan *magnetic bumi*, apabila terdapat medan *magnetic* lain disekitar sensor maka dapat mengganggu pembacaan dari pada sensor *compass*. Prototipe ASV yang digunakan memiliki bahan dasar *carbon fibre* yang memiliki sifat konduktor, sehingga dapat membuat suatu medan magnetnya sendiri dan mengganggu pembacaan sensor *compass*. Data pada gambar 4.5 terdapat lonjakan *error* tiba – tiba, hal ini disebabkan karena adanya perubahan target atau *waypoint* berganti ke *waypoint* selanjutnya, sehingga *Global Control* akan menyesuaikan kondisi ASV untuk memenuhi *trajectory waypoint* yang baru.

Meskipun terdapat *noise* pada *compass*, ASV tetap mampu menjalankan *global control* dengan baik. Hal tersebut dibuktikan dengan kecilnya *error jarak* dari ASV dengan titik yang diinginkan. Algoritma yang digunakan pada subbab 3.3.2.2 dinilai mampu untuk menghadapi masalah kompas tersebut dan memberikan performa *global control* yang baik.

4.2.3 Computer Vision

Pada pembuatan dan pengujian model, didapatkan data *training* dan data *testing* yang kemudian akan dibahas pada subbab ini. Proses *training* dilakukan menggunakan GPU Nvidia P100 dengan jumlah iterasi 6000. Selanjutnya diambil *weight* terbaik untuk selanjutnya digunakan. Adapaun grafik *training* menggunakan darknet yang berisikan mAP *training* dan mAP *validation* pada gambar 4.6.



Gambar 4.6 Grafik Evaluasi mAP pada saat *Training* Menggunakan Darknet

Dapat dilihat pada gambar 4.6 pada iterasi diatas 1200 nilai *loss* dengan cepat konvergen. Hal tersebut dimungkinkan karena *dataset* yang digunakan cenderung homogen. Hal serupa terjadi pada nilai mAP validasi yang langsung menyentuh di kisaran angka 98%. Tingginya nilai validasi dikarenakan hanya mendeteksi satu kelas yang cenderung homogen dan feature extractor sudah mampu mendeteksi objek tersebut, mengingat objek manusia ada dalam dataset COCO dimana darknet dilatih.

Setelah model CNN dilatih pada darknet, kemudian dilakukan konversi menjadi format TensorRT. Hal ini bertujuan untuk meningkatkan performa *inference*. Data perbandingan performa dengan menggunakan RTX 2060 Mobile antara darknet dan TensorRT dapat dilihat pada tabel 4.4.

Tabel 4.4 Perbandingan Performa *Inference* Model

Engine	FPS	Daya
Darknet	24-31	96W
TensorRT	80-90	96W
TensorRT	30	40W

Dengan adanya TensorRT menambah efisiensi dari penggunaan daya dengan performa yang tetap sama. Adapun nilai evaluasi mAP menggunakan data *test* dari model darknet tersebut jika dibandingkan dengan model yang telah diubah menjadi TensorRT, disajikan pada tabel 4.5.

Tabel 4.5 Perbandingan Evaluasi *Testing* mAP dari Model

<i>Engine</i>	mAP@IoU = 0.5
Darknet	0.840203
TensorRT	0.776

Dari tabel 4.5 dapat dilihat bahwa terdapat sedikit penurunan performa mAP dari darknet ke TensorRT. Hal tersebut mungkin dikarenakan adanya konversi model yang dilakukan menjadi FP16. Namun, hal ini tidak memberikan efek yang signifikan terhadap performa model dalam skenario uji coba penelitian ini. Model dipergunakan untuk mendeteksi korban kecelakaan laut kemudian mendapatkan posisi tengah dari korban (*center_x*, *center_y*), sehingga turunnya sedikit nilai dari IoU tidak berpengaruh. Adapun dari hasil percobaan di Danau 8, Model ini mampu mendeteksi korban hingga jarak 30 meter.

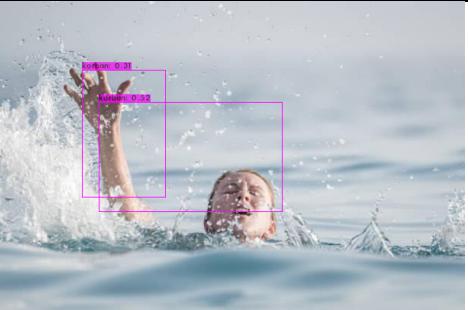
Tabel 4.6 Confusion Matrix Testing dari Model

		Predicted Label	
		Negative	Positive
<i>True Label</i>	<i>Negative</i>	---	6
	<i>Positive</i>	9	40

Adapun *confusion matrix* yang didapatkan dari hasil *testing* dengan batas IoU>0.5 disajikan pada tabel 4.6. Adapun beberapa deteksi yang mengalami kesalahan dapat dilihat pada tabel 4.7.

Tabel 4.7 Detail Kesalahan Deteksi pada Saat *Testing*

No.	Hasil Deteksi	Label (<i>ground truth</i>)	<i>Confusion Matrix</i>
1.			TP = 1 FP = 1
2.			TP = 1 FN = 2

3.			TP = 0 FN = 1
4.			TP = 0 FP = 1
5.			TP = 1 FP = 1

Dari tabel 4.7 dapat dilihat bahwa beberapa kesalahan tersebut terjadi karena hasil deteksi tidak memenuhi kriteria $\text{IoU} > 0.5$ atas *ground truth*. Hal tersebut mungkin diakibatkan kurangnya konsistensi dalam proses *labelling*, terutama pada *dataset* yang terlihat bagian kepala saja dan terlihat bagian kepala dan bahu. Pada hasil deteksi nomor 4, terlihat *ground truth* hanya melabeli pada bagian kepala saja, tetapi deteksi melabeli hingga bagian bahu, sehingga menyebabkan adanya kesalahan deteksi jika dilihat dari sisi IoU. Pada hasil deteksi nomor 2, dialami *false negative* dimana, objek korban bagian kepala tidak terdeteksi sama sekali, hal ini mungkin dikarenakan kurangnya *dataset* yang serupa, dilihat *data test* dengan ketinggian kamera yang rendah.

BAB 5 KESIMPULAN DAN SARAN

Pada bab ini akan dijelaskan mengenai kesimpulan dari hasil uji coba yang telah dilakukan serta saran-saran tentang pengembangan yang dapat dilakukan terhadap tugas akhir ini di masa yang akan datang.

5.1 Kesimpulan

Dari analisis dan uji coba yang dilakukan terhadap perfoma *Autonomous Surface Vehicle* untuk mencari korban kecelakaan laut yang berada di permukaan dapat ditarik kesimpulan bahwa prototipe ASV ini dapat memenuhi kebutuhan masalah yang ada dalam batasan *scope* yang telah ditentukan. Dari 10 percobaan, prototipe selalu mampu menemukan korban.

YOLOv4 mampu mendeteksi korban dengan jarak hingga 30 meter menggunakan kamera Logitech C930 dengan ukuran input *image* 640 x 480. Dilakukan *testing* pada model YOLOv4 didapatkan mAP@IoU=0.5 0.840203. Setelah dilakukan konversi model dari darknet ke TensorRT nilai mAP@IoU=0.5 didapatkan 0.776. Nilai mAP dari model yang dibangun cukup baik, meskipun hanya menggunakan 142 gambar *training*. Performa FPS menggunakan RTX 2060 Mobile dari model darknet dan TensorRT adalah 27 FPS dan 85 FPS.

Sistem navigasi *autonomous* yang diusulkan mampu menghasilkan navigasi ASV yang baik. Adapun kemampuan navigasi pada *Global Control* ini dapat meminimalkan *error_distance* dan *error_angle* yang digunakan sebagai rumus persamaan *global control*. Sehingga ASV dapat berjalan sesuai *path* yang telah ditentukan. Kemampuan navigasi *local control* juga didapati hasil yang baik. CameraControl mampu mendekat dan mengikuti korban dengan cepat dan tepat. Adapun ObstacleAvoidanceControl yang menggunakan sensor *ultrasonic* sebagai sensor *ranging*, mampu menghindari *obstacle*, meskipun hanya dapat bekerja baik ketika ASV sedang melaju pada kelajuan rendah, hal ini dikarenakan *refresh rate* dari sensor ini tidak cepat.

5.2 Saran

Pada tugas akhir kali ini tentunya terdapat kekurangan serta nilai-nilai yang dapat penulis ambil. Adapun saran yang penulis sampaikan untuk penelitian selanjutnya. Saat ini komputasi yang digunakan oleh penulis cukup tinggi. Namun seiring berkembangnya waktu, *embedded computing* juga makin berkembang, dengan digunakannya *embedded computer* maka penggunaan daya juga makin *minimum*, sehingga mampu menambah kemampuan waktu operasi. Penulis mengharapkan penelitian selanjutnya dapat digunakan model yang lebih efisien dan dapat berjalan di *embedded system*. Adapun dari sisi visi *computer*, dapat dilakukan perbaikan pada jumlah *dataset* yang digunakan, sehingga dapat menambah generalisasi dari objek korban kecelakaan yang dideteksi. Adapun sensor – sensor penunjang yang dapat digunakan untuk meningkatkan interpretasi *program control* untuk melihat lingkungan, seperti LiDAR, *thermal camera*, dan sonar. Prototipe ASV yang digunakan dalam penelitian ini memiliki bahan *carbon fiber*, sehingga rawan *noise* elektromagnetik, disarankan untuk melakukan isolasi secara khusus dalam peletakan sensor-sensor *sensitive*. Untuk mampu diimplementasikan pada medan sesungguhnya perbesaran skala prototipe juga suatu kebutuhan, sehingga nilai fungsinya dapat bertambah.

DAFTAR PUSTAKA

- [1] KNKT, “Data Investigasi Kecelakaan Pelayaran,” *knkt.dephub.go.id*, 2016.
- [2] CNN Indonesia, “Campur Aduk Emosi Penyelam SAR Sriwijaya Air SJ 182,” *CNN Indonesia*, Jan. 12, 2021.
- [3] S. In Cho, “Vision-based people counter using CNN-Based event classification,” *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 8, pp. 5308–5315, Aug. 2020, doi: 10.1109/TIM.2019.2959853.
- [4] G. Maria, E. Baccaglini, D. Brevi, M. Gavelli, and R. Scopigno, “A drone-based image processing system for car detection in a smart transport infrastructure,” *Proceedings of the 18th Mediterranean Electrotechnical Conference: Intelligent and Efficient Technologies and Services for the Citizen, MELECON 2016*, Jun. 2016, doi: 10.1109/MELCON.2016.7495454.
- [5] J. Li, C. Qu, and J. Shao, “Ship detection in SAR images based on an improved faster R-CNN,” *Proceedings of 2017 SAR in Big Data Era: Models, Methods and Applications, BIGSARDATA 2017*, vol. 2017-January, pp. 1–6, Nov. 2017, doi: 10.1109/BIGSARDATA.2017.8124934.
- [6] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” Apr. 2018, [Online]. Available: <http://arxiv.org/abs/1804.02767>
- [7] H. Mousazadeh *et al.*, “Developing a navigation, guidance and obstacle avoidance algorithm for an Unmanned Surface Vehicle (USV) by algorithms fusion,” *Ocean Engineering*, vol. 159, pp. 56–65, Jul. 2018, doi: 10.1016/J.OCEANENG.2018.04.018.
- [8] R. jian Yan, S. Pang, H. bing Sun, and Y. jie Pang, “Development and missions of unmanned surface vehicle,” *Journal of Marine Science and Application*, vol. 9, no. 4, 2010, doi: 10.1007/s11804-010-1033-2.
- [9] M. Whitty, “Robotics, Vision and Control. Fundamental Algorithms in MATLAB,” *Industrial Robot: An International Journal*, vol. 39, no. 6, 2012, doi: 10.1108/ir.2012.04939faa.005.
- [10] by J. Borenstein, Y. Koren, and S. Member, “THE VECTOR FIELD HISTOGRAM-FAST OBSTACLE AVOIDANCE FOR MOBILE ROBOTS,” 1991.
- [11] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems (8th Edition) (What's New in Engineering)*. Pearson, 2018. [Online]. Available: <https://www.amazon.com/Feedback-Control-Dynamic-Systems-Engineering/dp/0134685717?SubscriptionId=AKIAIOBINVZYXQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0134685717>
- [12] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” Apr. 2020, [Online]. Available: <http://arxiv.org/abs/2004.10934>
- [13] I. Loshchilov and F. Hutter, “SGDR: STOCHASTIC GRADIENT DESCENT WITH WARM RESTARTS”, Accessed: Jun. 29, 2022. [Online]. Available: <https://github.com/loshchil/SGDR>
- [14] H. C. Shin *et al.*, “Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning,” *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1285–1298, May 2016, doi: 10.1109/TMI.2016.2528162.
- [15] E. Jeong, J. Kim, S. Tan, J. Lee, and S. Ha, “Deep Learning Inference Parallelization on Heterogeneous Processors With TensorRT,” *IEEE Embedded Systems Letters*, vol. 14, no. 1, pp. 15–18, 2022, doi: 10.1109/LES.2021.3087707.

- [16] J. F. Kurose and K. W. Ross, *Computer networking : a top-down approach*, 7th ed. New Jersey: Pearson, 2017.
- [17] Stanford Artificial Intelligence Laboratory et al., “Robotic Operating System.” 2018. [Online]. Available: <https://www.ros.org>
- [18] M. T. LIU LING and ÖZSU, Ed., “Mean Average Precision,” in *Encyclopedia of Database Systems*, Boston, MA: Springer US, 2009, p. 1703. doi: 10.1007/978-0-387-39940-9_3032.
- [19] L. Yun, A. Bliault, and H. Z. Rong, “High speed catamarans and multihulls: Technology, performance, and applications,” *High Speed Catamarans and Multihulls: Technology, Performance, and Applications*, pp. 1–746, Oct. 2018, doi: 10.1007/978-1-4939-7891-5/COVER.
- [20] A. Bochkovskiy, C.-Y. Wang, and J. Redmon, “darknet,” *github*, 2020. <https://github.com/AlexeyAB/darknet> (accessed Mar. 30, 2022).
- [21] J. Jung, “tensorrt_demos,” *github*, 2021. https://github.com/jkjung-avt/tensorrt_demos (accessed Apr. 30, 2022).

LAMPIRAN

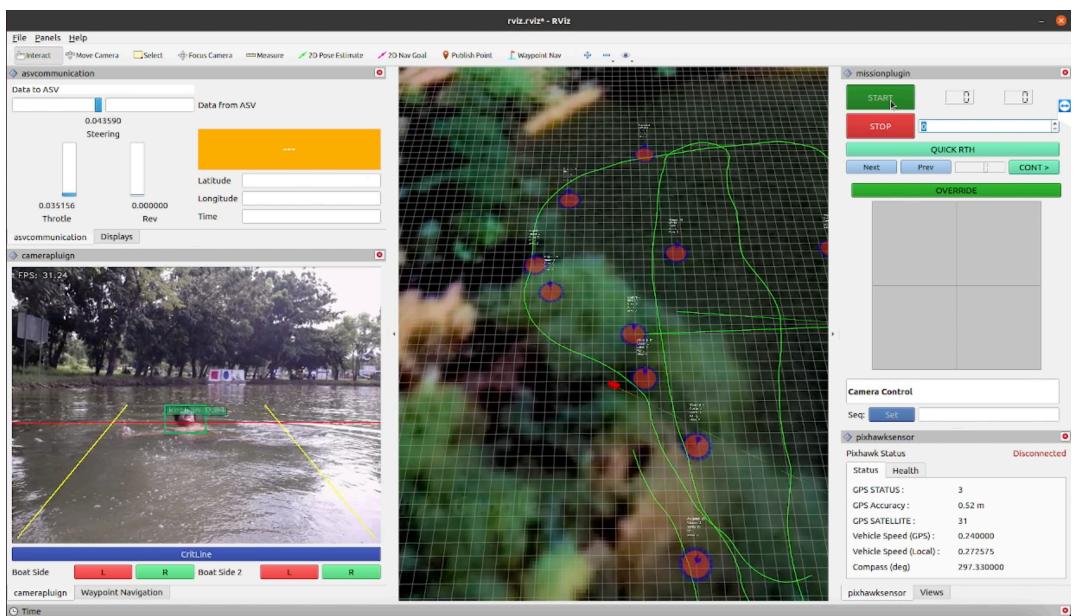
Berikut merupakan lampiran hasil uji coba



Gambar 5.1 ASV Beroperasi di Danau 8 ITS



Gambar 5.2 ASV Mendekat ke Korban



Gambar 5.3 Tampilan Antarmuka Saat Mendeteksi Korban

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS



Achmad Zidan Akbar, lahir pada tanggal 24 Juni 2000 di Lumajang. Penulis menempuh Pendidikan S1 di Departemen Teknik Informatika, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember, Surabaya. Penulis mengambil rumpun kuliah Komputasi Cerdas dan Visi (KCV). Penulis aktif dalam kegiatan riset robot, terutama di Unit Kegiatan Mahasiswa. Penulis merupakan programmer dari tim Barunastra ITS. Penulis telah mengikuti berbagai ajang kompetisi bersama

Barunastra ITS pada bidang Autonomous Surface Vehicle. Penulis berkesempatan terlibat pada riset – riset robot yang dilakukan oleh ITS, antara lain, Robot Medical Assistant ITS – Unair (RAISA), Intelligen Car ITS (I-Car ITS), dan Intelligent Boat (I-Boat ITS). Penulis dapat dihubungi melalui alamat email azidan.it@gmail.com.