Technical University of Crete, Greece

School of Electrical and Computer Engineering

Machine Learning for Enhancing Robotic Perception and Control

Dimitrios Chatziparaschis

Thesis Committee

Associate Professor Michail G. Lagoudakis (ECE)

Professor Aggelos Bletsas (ECE)

Associate Professor Panagiotis Partsinevelos (MRE)

Chania, November 2020

Πολυτεχνειο Κρητης

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Μηχανική Μάθηση για την Ενίσχυση της Ρομποτικής Αντίληψης και Ελέγχου

Δημήτριος Χατζηπαράσχης

Χανιά, Νοέμβριος 2020

Abstract

In recent years, there is an emerging need to use robotic systems to facilitate human missions, especially in Search-and-Rescue scenarios. Such systems may operate in cluttered and human-unfriendly environments, in which there may be no ideal circumstances to establish a remote control connection and delays may be detrimental due to the emergency of such scenarios. Henceforth, the most essential trait of these systems is their ability to deal with the uncertainty of the operating environment, in order to make appropriate decisions and accomplish their objectives autonomously. In this thesis, we utilize Machine Learning approaches to enhance robotic perception and control, namely vision and navigation, of a simulated Unmanned Aerial Vehicle (UAV) to be able to act fully autonomously in reconnaissance and rescue procedures. On the perception side, we use a custom Deconvolutional Neural Network trained on tailor-made loss functions to achieve autonomous visual target detection. On the control side, we applied Deep Reinforcement Learning using Deep Deterministic Policy Gradient, based on a custom lightweight training simulator, to obtain the appropriate autonomous navigation behavior in unknown worlds. The enhanced UAV system can navigate safely through an unknown environment, search and detect any existing humans in its surroundings with its onboard gimbal system, engage and take distance measurements from the acquired target, and georeference it to the global coordinate system. Thenceforth, the UAV pinpoints the positioned target in the generated map, shares it with the responding team, and proceeds with the exploration of the unmapped area to locate other individuals who may be in need. Throughout this study, each developed autonomous behavior of the UAV was thoroughly evaluated to demonstrate experimental results in various custom environments within the Gazebo robot simulation environment. The proposed system has been developed as a Robot Operating System (ROS) package and is deployable to both simulated and real UAV systems, as long as they meet the minimum proposed software and sensory requirements.

Περίληψη

Τα τελευταία χρόνια, υπάρχει μια αναδυόμενη ανάγκη χρήσης ρομποτικών συστημάτων για τη διευκόλυνση ανθρώπινων αποστολών, ειδικά σε σενάρια έρευνας και διάσωσης. Τέτοια συστήματα μπορεί να λειτουργούν σε ακατάστατα και μη-φιλικά προς τον άνθρωπο περιβάλλοντα, στα οποία ενδέχεται να μην υπάρχουν ιδανικές συνθήκες για τη δημιουργία απομακρυσμένης σύνδεσης τηλεχειρισμού και οι καθυστερήσεις μπορεί να είναι επιζήμιες λόγω της κρισιμότητας του σεναρίου. Επομένως, το πιο βασικό χαρακτηριστικό αυτών των συστημάτων είναι η ικανότητά τους να αντιμετωπίζουν την αβεβαιότητα του περιβάλλοντος όπου ενεργούν, προκειμένου να λαμβάνουν τις κατάλληλες αποφάσεις για να επιτυγχάνουν αυτόνομα τους στόχους τους. Στην παρούσα διατριβή, χρησιμοποιούμε προσεγγίσεις Μηχανικής Μάθησης (Machine Learning) για την ενίσχυση της ρομποτικής αντίληψης και ελέγχου, συγκεκριμένα της όρασης και της πλοήγησης, για ένα προσομοιωμένο μη-επανδρωμένο εναέριο όχημα (Unmanned Aerial Vehicle - UAV) που θα μπορεί να ενεργεί πλήρως αυτόνομα στις διαδικασίες αναζήτησης και διάσωσης. Στην πλευρά της αντίληψης, χρησιμοποιούμε ένα προσαρμοσμένο Deconvolutional Neural Network, εκπαιδευμένο πάνω σε ειδικά σχεδιασμένες συναρτήσεις κόστους, για να πετύχουμε αυτόνομο εντοπισμό οπτικού στόχου. Στην πλευρά του ελέγχου, εφαρμόσαμε Βαθιά Ενισχυτική Μάθηση (Deep Reinforcement Learning) χρησιμοποιώντας Deep Deterministic Policy Gradient, πάνω σε ένα ειδικά διαμορφωμένο και χαμηλής υπολογιστικής πολυπλοκότητας περιβάλλον εκπαίδευσης, για να αποκτήσουμε την κατάλληλη αυτόνομη συμπεριφορά για πλοήγηση σε άγνωστους κόσμους. Το βελτιωμένο σύστημα UAV μπορεί να πλοηγηθεί με ασφάλεια μέσα στο άγνωστο περιβάλλον, να αναζητήσει και να εντοπίσει ανθρώπους στη γύρω περιοχή με το ενσωματωμένο σύστημα gimbal, να εστιάσει και να λάβει μετρήσεις απόστασης από τον αναγνωρισμένο στόχο και να τον αναφέρει στο καθολικό σύστημα συντεταγμένων. Στη συνέχεια, το UAV εντοπίζει τον προσδιορισμένο στόχο στον παραγόμενο χάρτη της άγνωστης περιοχής, τον μοιράζεται με την διασωστική ομάδα και προχωρά στην εξερεύνηση της μη-χαρτογραφημένης περιοχής για να εντοπίσει άλλα άτομα που βρίσκονται σε κίνδυνο. Σε όλη τη διάρκεια της εργασίας, κάθε ανεπτυγμένη αυτόνομη συμπεριφορά του UAV αξιολογήθηκε διεξοδικά για να επιδείξει πειραματικά αποτελέσματα σε διάφορα προσαρμοσμένα περιβάλλοντα στο περιβάλλον προσομοίωσης ρομπότ Gazebo. Το προτεινόμενο σύστημα έχει αναπτυχθεί ως πακέτο Robot Operating System (ROS) και μπορεί να εφαρμοστεί τόσο σε προσομοιωμένα όσο και σε πραγματικά συστήματα, εφόσον πληρούν τις ελάχιστες προτεινόμενες απαιτήσεις λογισμικού και αισθητηρίων.

Acknowledgements

First of all, I am deeply and cordially thankful to my two research supervisors and professors that I had in both my undergraduate and graduate years, Michail G. Lagoudakis and Panagiotis Partsinevelos. Their abundant support, constant guidance, and unbounded trust even in my early steps in their laboratories, made me discover and focus on what I really like and be truly prepared for my future steps.

I strongly believe that their mentality and the common experiences that we had are the main reasons that I love what I do today, and I cannot really imagine myself if I would not have met them.

I would like to express my deep gratidute to the *Pancretan Endowment Fund*, which provided me the financial support to fullfil my graduate studies and research at the Technical University of Crete.

I am also grateful for being part of the Senselab and Kouretes research teams. I am grateful for being part in all these collaborative experiences, in each competition that we participated and every presentation that we performed, in all of these places here in Greece and abroad. I am grateful for every person that I have met and for all the cordially friends that I have made throughout my years in these teams.

I would like to thank all my friends, and especially Angelos Antonopoulos, Stelios Manoudakis, Dimitris Trigkakis, Manos Stefanakis, Ilias Tsichlis and George Vougioukas which were always by my side throughout this memorable trip.

Finally, and above all, I am deeply greatful to my mother Alexia, my father Antonis, and my uncle Kostis, for their abundant love and support since I remember me.

Contents

List of Figures

List of Algorithms

LIST OF ALGORITHMS

Chapter 1

Introduction

Over recent years, due to the emerging advance of robotics field, there is an increasing spectrum of applications that include robot usage, either for research purposes or for common everyday use-cases. A notable percentage of the latter category refers on Search-and-Rescue (SAR) scenarios, on which first-responder teams have to act rapidly and efficiently in an emergency occurence to provide assistance to people and animals that are in need.

Specifically, these emergency situations may be occurred in distant areas which may be destructed, inaccessible, and even hazardous for the first-responders, to conduct a safe operation. Also, these places can be characterized as dynamically changing, as they can change spatially while as the incident is still active (building collapses, extreme natural phenomena, etc.), which raises the complexity of the rescuers' work, and therefore increases their reaction time. For this reason, some rescuing teams have recruited robotic systems to facilitate their work, by providing a broader view of the current situation and also by helping them with their rescuing actions. In the meantime, establishing a teleoperation station to manually control these robots can be inefficient, and sometimes infeasible, due to the existing circumstances. Henceforth, an essential trait for the rescuing robots would be their ability to act semi-autonomously, or even better totally autonomously, to perform their tasks alone and act like a real independent rescuer.

The main problem which these robots have to face is the environment uncertainty. To deal with this matter, these robots are equipped with a plethora of sensors that enable them to obtain a belief about the spatial structure of their surrounding world. In many situations, since there is no prior or adequate map information of the current area, the robots have to create the map and continuously localize themselves in it, to explore and operate safely. Additionally, these robots should be able to decide about their upcoming actions (either in navigation or rescuing) autonomously, as there would be limited or no human intervention in their decisions. In this part, even though there are remarkable works that propose probabilistic approaches to decision making and acting, emerging applications utilize machine learning techniques to create more general and adaptive behaviors of such robots, by training them in simulated and real scenarios, and thus enabling them to deal with new and unforeseen environmental conditions.

At the same time, an essential characteristic of rescuing robots should be their ability to locate key objects in the unknown area, such as individuals that need help, throughout their operation. Indubitably, this procedure should be precise and trustworthy, since minor mistakes during the rescue might produce inaccurate rescuing intel and therefore may imply the delay of the first-responders' work. For this reason, since present robots have the ability to carry multiple onboard sensors and support sufficient computational resources for online processing, it is vital to utilize their online processing capabilities and combine information taken from different sensors, to enhance their detection ability, cross-validate their findings, and finally obtain an accurate belief about their surrounding emergency situation.

Overall, as the operation of a robot can be quite demanding, in such human-unfriendly environments, more general and robust approaches for its autonomy should be followed to ensure its efficient navigation, target detection, and decision making characteristics. Hence, the autonomous robotic rescuer should seize every available bit of captured information and onboard computational resource, in order to maximize its performance in the unknown field, and thus contribute actively to the mission, as a real rescuer would do.

1.1 Thesis Contribution

This thesis proposes an Unmanned Aerial Vehicle (UAV) system solution, which can autonomously navigate and locate individuals, for reconnaissance purposes in Search-and-Rescue (SAR) scenarios, in totally unknown environments. This system is designed and tested in the Gazebo simulator, and its behavior has been developed as a ROS software package, that allows its installation on any other UAV system (simulated or not), as soon as it meets the necessary structure and sensory requirements. The goal of this UAV is to navigate safely and autonomously in the unknown area, construct the area map, detect and localize every person or object of interest in the generated map, and proceed with the search of the unexplored areas. The UAV forms a map of the rescuing environment with the pinpointed interest areas and is able to transmit it via the established wireless network to the first-responders in order to plan their operation.

Specifically, the UAV system is equipped with a three-dimensional system, which integrates a dual-camera setup (optical and thermal unit) and a one-dimensional rangefinder. This gimbal system has been developed to act independently from the UAV body movement, since it can move in all three directions, and its main role is to search and localize object of interest in the searching area during the UAV navigation. Particularly, by obtaining distance measurements from a detected target, it performs a multilateration approach with applied custom spatial criteria and concludes with high accuracy target positionings.

In terms of target recognition, the UAV utilizes the image data from both the thermal and the optical camera. On the side of the thermography camera, it excludes and prioritizes image regions that contain high-temperature values and resemble human beings. On the side of the optical camera, a custom neural network has been implemented and trained on the image datasets that contain humans. This network has been initially tested on real-world datasets to test its performance in real SAR circumstances, and afterwards on generated Gazebo imagery data, and was successfully integrated into the proposed UAV solution. For the evaluation and improvement of the detector, a variety of loss functions and metrics have been implemented, combined, and tested for training, to conclude with the most appropriate detector setup for this problem. The architecture of this network has been suitably selected to be able to run online and onboard during either simulated or real UAV flight.

At the same time, a LiDAR system is rigidly mounted on the UAV body, by which it performs simultaneously localization and mapping procedures. Since the UAV has to operate autonomously, as there is no human intervention, it learned flight behaviors through a Deep Reinforcement Learning (DRL) approach, by using the Deep Deterministic Policy Gradient (DDPG) algorithm to learn the Q-function and the optimal policies. Specifically, the UAV agent utilizes only 6 distance measurements captured by its LiDAR sensor in specific directions to decide for the upcoming move. To accelerate the learning procedure, we have developed a dedicated OpenAI Gym environment that simulates the Gazebo world and UAV control specifications and can be used with or without User Interface (UI). Hence, we could perform numerous training and testing experiments, and obtain the ideal UAV behavior without the need to perform the training on the Gazebo simulator and be subject to its system requirements. Thenceforth, the pretrained models are transferred on the UAV system in the Gazebo simulator, and thus the UAV could maneuver safely and navigate by using its LiDAR measurements, in the unknown world.

Overall, this work demonstrates the use of Machine Learning (ML) in both vision and control of a simulated UAV model, to act autonomously in SAR scenarios. The selection of each sensor and the development of each detector system has been made concerning recent UAV computational onboard capabilities and optimal cost efficiency. The complete UAV solution has been tested and evaluated in a variety of randomly-generated Gazebo SAR environments and modified appropriately to complete successfully its reconnaissance missions. This work will be made available[1] as a complete ROS package and can be executed on both simulated and real UAVs, by applying proper adjustments, as long as they satisfy the required technical specifications.

1.2 Thesis Outline

In Chapter 2 we present all the background information needed for this thesis. We give an overview of the used framework for development, the simulator environment, and the used sensory specifications, which will be used throughout this study. Also, a brief description will be made around the fundamentals of recent Machine Learning (ML) approaches in both the robotic perception and the robotic control spectrum. In Chapter 3 we state the issues around the autonomous reconnaissance approach, from the aspect of a rescuing UAV robot, and also mention notable studies that have been made to address these problems. In Chapter 4 we describe in detail the implementation steps of the proposed solution, starting with the robot setup and required specifications and ending with the overall UAV behavior toward the human positioning and exploration problems. In Chapter 5 we present the results and remarks from the creation of the autonomous rescuing UAV. Specifically, we present the training and testing results of both UAV's

[1]https://github.com/jimcha21

human detection and navigation procedures, by noting important characteristics that aim to enhance the UAV's autonomous operation, and we perform rescuing experiments within the Gazebo simulator. This work is concluded in Chapter 6, in which future plans for extending our approach are presented.

Chapter 2

Background

In this section, we present background information that is essential to support the approach of this study. Specifically, we describe fundamental information about the used framework for development, the robotic simulator tool, and the proposed UAV model that will be used in this scenario. Also, a brief description will be made of the sensors' functionality and noise models, which will be required to resolve the Search-and-Rescue problem. In the end, concise details will be given about the Machine Learning field and recent applications, which are necessary to support our approach in both autonomous robotic perception and control.

2.1 Robot Operating System (ROS)

The Robot Operating System [1] is a framework that was developed by the Willow Garage robotics research lab, to introduce a common basis for robotic application development and maintenance. Specifically, it is a middleware that runs on an established network master device, on which, one or more robotic devices may be connected. Due to its distributed architecture, given a selected central node (computer system or robot system), all sensory or actuator messages can be monitored and accessed in real time, as they are published asynchronously from each robot system. Thus, this framework allows the implementation of high-end robotic applications that can run real-time on a robot depending on their onboard sensor units, despite their software development kit setup.

2.2 Gazebo Simulator

Figure 2.1: The Gazebo environment.

The Gazebo [2] simulator is an open source robot simulation tool, that is broadly used for robotic application development and testing scenarios. Many companies and associations, like *NASA*[1] and *RoboCup Rescue*[2,3], use this simulator in order to develop applications for specific simulated robots and then test it on real world conditions. It integrates a world physics engine and dynamics simulation tools, for realistic simulated scenarios. Figure 2.1 illustates a snapshot of a simulated world created in the Gazebo. This tool also includes many simulated models, of real and familiar robots, and they can be used and manipulated in the simulation environment. These robots may have sensor units (like cameras, distance sensors, and etc.) that are also usable and adjustable to work in this simulator software. Every simulated sensor within the Gazebo environment can be characterized by its own noise model, which resembles the captured noise in measurements as in real-world scenarios. The ROS framework is fully compatible with the GazeboSim, as each spawned robot has a developed ROS driver (ROS-Software Development Kit (SDK) package), which enables its control within the environment and provides access on

[1] https://www.nasa.gov/feature/space-robotics-challenge

[2] https://rescuesim.robocup.org/news/2018-robocup-rescue-virtual-robot.-simulation-competition/

[3] http://gazebosim.org/ariac

the information captured by its onboard simulated sensors. Thus, this makes it ideal for initial application development and testing, and then, due to ROS framework scalability and adjustability features, it can be installed on a real UAV system and be tunned and evaluated in real-world scenarios.

2.3 Sensors

Every robot needs a variety of sensors in order to adapt in the operation environment, deal with spatial uncertainty, and finally fulfil the requested objectives.

2.3.1 Camera Sensor

One of the foremost used sensors nowadays in robotic systems is the camera sensors. Due to the emerging advancement of the Machine Vision applications, these sensors are employed for various purposes, such as object identification, scene's depth estimation, visual localization and mapping, and more. In specific, the camera modules contain an image sensor, namely the imager, which is sensitive to light or electromagnetic radiation waves exposure, and through the captured energy it creates the image.

Generally, a conventional 2D camera model is based on the pinhole camera's functionality, which describes a box with a single and tiny hole, namely the aperture, by which only the correctly aligned light-rays pass through and "project" an inverted image on the opposite surface of the box, namely the projective plane. As an extension to this model, Figure 2.2 illustrates an object been projected upright on the camera's image plane, which plane represents the slice of all those light rays that end up passing through the pinhole's aperture.

In this way, every three-dimensional point P can be projected on the focal plane on its corresponding two-dimensional coordinates $P_c(x, y)$. To achieve this, we define the camera's initrincs matrix $\mathcal{K}$, as follows,

$$\mathcal{K} = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

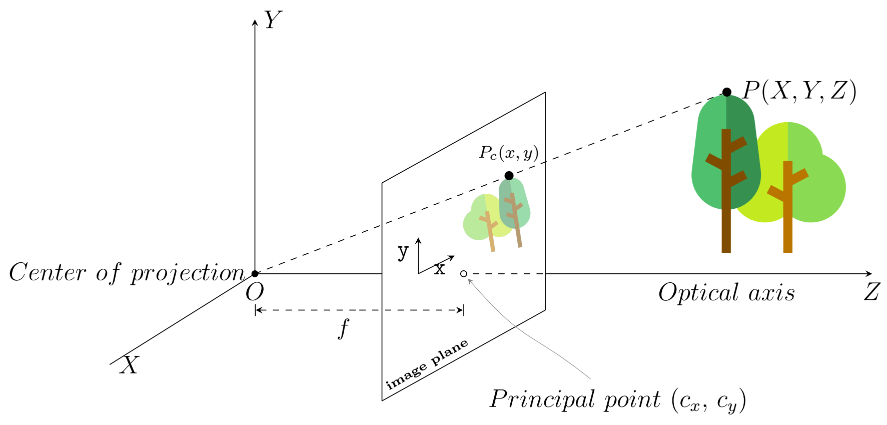


Figure 2.2: The pinhole camera model.

In specific, the f_x and f_y parameters represent the focal length values of each image plane axis, the s is the skew parameter and c_x and c_y the principal point coordinates, which indicate the intersection point of projection center and image plane.

Also, as cameras use lenses to focus the incoming light, their images are disposed of distortion effects. Hence, to suppress these consequences we define the distortion matrix $\mathcal{D}$, which contains the distortion coefficients that can transform and undistort the image to its original acquisition. Specifically,

$$\mathcal{D} = \Big[k_1, k_2, p_1, p_2, k_3, k_4, k_5, k_6\Big] \tag{2.2}$$

where the k_i's and the p_i's are the radial and tangential distortion coefficients respectively. Both $\mathcal{K}$ and $\mathcal{D}$ matrices can be approximated by a calibration procedure of a monocular camera, by using a measured calibration pattern board, like a chessboard grid.

Thermography Camera Sensor

As an extension of the color cameras, there are camera modules that capture information from the wavelengths of infrared radiation. In contrast to the visible light, the infrared wavelengths span from $0.75\mu m$ to $15\mu m$ (the long-wave infrared) and reach until $1000\mu m$ which is defined as the far-infrared wave region. In specific, the margin from $1\mu m$ to $14\mu m$ can be used for thermal measurements, and this is where the thermographic cameras are sensitive. In this way, the camera sensor captures the infrared energy, which is implied to the temperature value and forms a thermal image in which every pixel represents the captured temperature. Every thermography camera can be characterized by its camera matrix $\mathcal{K}$, like a color camera, and also is prone to distortion effects depending on its used lenses. Figures in 2.3 illustrate thermography images that are captured in real-world environments.

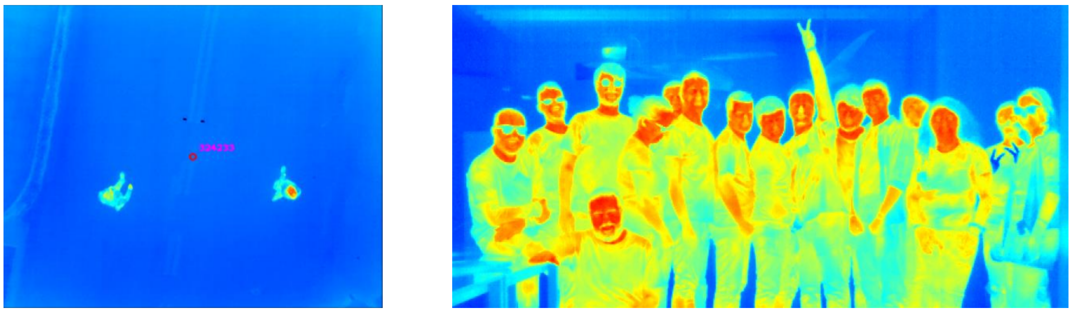


Figure 2.3: Thermographic images that depicts the temperature information of the viewing environment. The selected colormap is the "jet", which has been used to as a pseudocolor to represent the temperature values. Image courtesy of Angelos Antonopoulos and the SenseLab.

As it is apparent, the usage of such cameras can be ideal for human and generally living creature detection, as also can be used for other objects that can be distinguished by their particular temperature information. This fact can be crucial for a Search-and-Rescue operating robot, which performs in low-lighting or cold environments, as it directly locates any thermal-positive object in the viewing scene and thus proceeds with its investigation.

2.3.2 Distance Measurement Sensor

In many robotic applications, from autonomous cars to surveying UAVs, laser rangefinders are used to obtain a precise spatial belief of the current environment. These sensors are based on the method called "LiDAR", namely light detection and ranging, by which the system emits a pulsed laser light on single or multiple predefined directions and extracts the distance of the reflected surface by utilizing physical principles of operation, like the intensity of reflection, triangulation and time of flight. There is a variety of laser rangefinder sensors for selection, depending on the mapping scenario conditions, the desired measurement accuracy level, and the cost profile.

In particular, these sensors can be distinguished into three categories depending on their dimensional perception, namely the 1D, 2D, and 3D LiDARs. As each category indicate the physicality of the captured measurements, these sensors can create pointcloud data that are described with respect to the LiDAR's receptor sensor module. In this way, the robotic system can be instantly aware of the spatial condition of its operating environment, and so it can plan its actions accordingly. It is worth noting, these sensors are also prone to measuring noise, as they can be occurred due to the existing environment circumstances (dusty air, humidity, etc.) or by possible sensor malfunctions.

2.3.3 Global Navigation Satellite System Receiver

The Global Navigation Satellite System (GNSS) is defined as the procedure of utilizing time signals and positioning data from satellites, to determine the global position of a GNSS receiver. These positioning receivers are broadly used in robotic applications, when it is essential to know the exact position of the robots in the global coordinate system, to perform more accurate navigation and maneuvring corrections (along with information from onboard Inertial Navigation Systems) and also to utilize to perform georeferenced tasks.

One of the most common navigational systems is the World Geodetic System 1984 (WGS84), which is the reference coordinate system of Global Positioning System (GPS) sattelite navigation and is an earth-fixed, global ellipsoid model. The ROS middleware and the Gazebo simulator uses the WGS84 elipsoid as the standard to define the longitide and latitude coordinates.

Specifically, the Gazebo simulator initiates a simulated form of the WGS84 coordinate system to map the simulation world. In particular, by initializing a global coordinate system in the starting point of its world, it geo-references every object in respect with this coordinate system, as there is constant access on their global position (groundtruth) by the simulation system. Figure 2.4 shows the Gazebo world coordinate system origin, as is projected by a blue perpendicular line that represents the z-axis of the world's origin point.

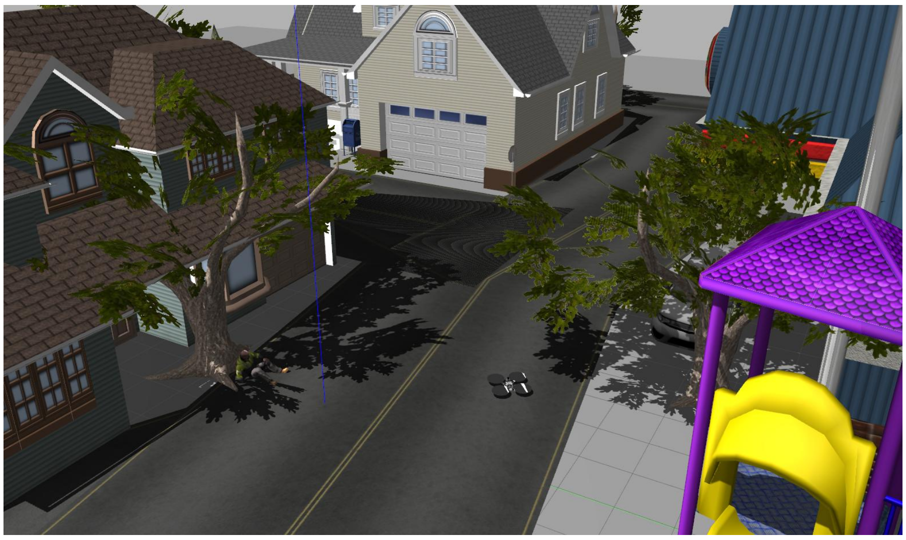


Figure 2.4: A snapshot of the simulated UAV system in the Gazebo world. The blue perpendicular line represents the z-axis of the world's origin point.

In our study, in order to map and utilize the captured positioning data, we to transform the captured GNSS data on the global Cartesian coordinate system. For this reason, we use the earth-centered earth-fixed geocentric coordinate system (ECEF), which expresses each point according to its position relative to the Earth's center of mass, with geocentric x, y and z coordinates. So, given the latitude (ϕ), longitude (λ) and height (h) values, the transformation into ECEF coordinates is performed by,

$$x = \Big(N(\phi) + h \Big) \cdot \cos\phi \cdot \cos\lambda \tag{2.3}$$

$$y = \Big(N(\phi) + h \Big) \cdot \cos\phi \cdot \sin\lambda \tag{2.4}$$

$$z = \Big(\frac{b^2}{a^2} N(\phi) + h \Big) \cdot \sin\phi \tag{2.5}$$

where,

$$N(\phi) = \frac{a^2}{\sqrt{a^2 \cos^2\phi + b^2 \sin^2\phi}} \tag{2.6}$$

and a, b parameters be the equatorial radius and the polar radius of the Earth, respectively. In this way, every global position can be referenced in the Cartesian ECEF geographic coordinate system, and thus can be described relatively to the Gazebo's environment coordinate system.

2.4 Machine Learning

2.4.1 Semantic Segmentation Metrics

The Semantic Segmentation in Machine learning fields, defines the procedure of extracting image regions that contain similar informatio/context. In this way, the image is "segmented" into separate parts that depict different objects, which can reveal the contextual meaning of the current captured scene.

First of all, in binary classification over the input image X, we assume that the network output Y will be a binary annotated image with the same size as X, which indicates the possible areas of the trained object's depiction. In specific, the image Y forms a probability distribution of the class instance appearance on the image frame, and thus can be represented as a heatmap.

During the learning procedure, the network tries to approximate on the groundtruth image Y by making the estimation of $\hat{Y}$ image. As we assume that images Y and $\hat{Y}$ are formed by pixel units y_i and $\hat{y_i}$ for $i \in [1, M]$, which indicate the classification of pixel value x_i in X input image, we can define parameters that can evaluate the precision of the estimation $\hat{Y}$ in respect of groundtruth annotated image Y. Specifically, True and False Positives (TP and FP), and True and False Negatives (TN and FN) quantities are essential in semantic segmentation scenarios, as they indicate the amount of correct and incorrect predictions over the image according to the groundtruth annotated image Y. Specifically, their definitions are,

$$\text{True Positives } (TP) \ = Y \cap \hat{Y} \tag{2.7}$$

$$\text{False Positives } (FP) \ = Y \cup \hat{Y} - Y = Y^c \cap \hat{Y} = \hat{Y} - TP \tag{2.8}$$

$$\text{True Negatives } (TN) \ = Y^c \cap \hat{Y}^c, \text{ where } Y^c = 1 \text{ - } Y. \tag{2.9}$$

$$\text{False Negatives } (FN) \;= Y \cup \hat{Y} - \hat{Y} = Y \cap \hat{Y}^c = Y - TP \tag{2.10}$$

Furthermore, we define the confusion matrix $\mathcal{C}$, which is a table that contains the aforementioned values,

$$\mathcal{C} = \begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix} \tag{2.11}$$

Given the definitions in the confusion matrix of Equation 2.11, we proceed with the definition of metric functions, that are used as the basis to evaluate detector accuracy in semantic segmentation.

Specifically, the *Sensitivity* parameter, also known as the *Recall* parameter and True Positive Rate (TPR), depicts the number of correct positives values derived by the number of total positive predicted areas,

$$Sensitivity\ (TPR) = \frac{TP}{TP + FN} \tag{2.12}$$

Similarly, to evaluate the ratio of correct negatives compared with the mount of the total negatives, we define the $Specificity$ parameter, known also as the True Negative Rate (TNR).

$$Specificity\ (TNR) = \frac{TN}{TN + FP} \tag{2.13}$$

Also, the *Accuracy* metric in semantic segmentation is defined as,

$$Accuracy = \frac{TP + TN}{N}, \text{ where } N \text{ is the number of pixels } y. \tag{2.14}$$

Additionally, it is worth defining the *Precision* parameter which shows the percentage of correct positive predictions over the image in respect of the total amount of positive predictions. This proportion can be calculated as follows,

$$Precision = \frac{TP}{TP + FP} \tag{2.15}$$

Since we have stated the fundamental metric parameters, we will describe essential semantic segmentation metrics which are used to evaluate the performance of such object detectors.

Mean Squared Error One of the most common loss and metric functions in machine learning is the Mean Squared Error (MSE), which is defined as follows,

$$MSE(Y,\hat{Y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2. \tag{2.16}$$

Mean Absolute Error Similarly, with the Mean Squared Error, the Mean Absolute Error (MAE) is defined as follows,

$$MAE(Y,\hat{Y}) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|. \tag{2.17}$$

Binary Cross Entropy A widely used loss function in binary classification problems, is the Binary Cross Entropy (BCE) metric, known also as the Log-Loss function, which is formulated as,

$$BCE(Y,\hat{Y}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i) \tag{2.18}$$

2.4.2 Batch Normalization

In 2015, Ioffe S. *et al.* [3] presented the Batch Normalization mechanism which is can mitigate the covariance shift that occures during the training procedure of a convolutional neural network. This mechanism can be applied on the sub-parts of the network and the layers, and aids to minimize internal-covariate-shift effect of convolutional layers. In specific, input distributions in each layer change over iterations during training as the parameters of the previous layers are updated. So, by applying batch normalizations on convolutional layer outputs, we normalize the input distributions of every layer to the standard Gaussian distribution, and so we avoid situations of poor local optimum during the training. This procedure leads to significant improvements in the speed of the training

2.5 Reinforcement Learning

Reinforcement Learning (RL) is a part of Machine Learning scientific field, which aims on learning through action performing and cumulative reward maximization as provided by the environment. In particular, the learner discovers the optimal policy (strategy) of its actions to achieve a specific goal, by evaluating the accumulated rewards during its experimentation, as they are formed by the environment. In the early steps of learning, a trial-and-error approach is followed as there is no prior guidance to make appropriate decisions, and afterwards, there may be a balance between further exploration (continuation of random moves) and utilization of prior knowledge to achieve short- or long-term optimal results.

First, we consider an agent interacting within an environment $\mathcal{E}$ which can be inspected at any time t by forming an observation o_t. Also, agent actions are denoted as a_t, and can be taken from pre-defined and known set of K actions, namely the $\mathcal{A} = (1, ..., K)$. Hence, each time t in which an action a_t is executed, the environment emits the observation o_t as an outcome of this event.

In the meantime, in order to determine the optimal agent's move in the current environment, we define the state s_t as the information used to make the action selection at time-step t. Specifically, there are two approaches of state definition which are distinguished by the current environment's observability. In one hand, there may be partial observation of the environment, and so the history of prior observations and actions is required to describe the current state. Thus, the state s at time-step t is,

$$s_t = (o_1, a_1, ..., o_{t-1}, a_{t-1}, o_t) \tag{2.19}$$

On the other hand, in case of fully observed environments there is no need to retain information from previous observations and states, since the current observation is adequate to decide about the upcoming action.

$$s_t = o_t \tag{2.20}$$

Henceforth, due to this distinction we can assume all states s_t (sequences of actions and observations) can form a finite Markov Decision Process (MDP), in which each state is represented at time t.

Furthermore, given an action execution, the agent receives a reward value r_t according to the affection of the environment due to this action. The goal of the learner is to interact with the environment by selecting proper actions in such way to maximize the future rewards. For this reason, it is considered that the future rewards are discounted by a factor γ per time-step, and the future discounted *return*, at time-step t, is formed as,

$$R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r_t, \text{ where } \gamma \in [0, 1). \tag{2.21}$$

Overall, the future reward function is denoted as the accumulation of state rewards from time t until a terminate state T. Meanwhile, the agent's behavior is controlled by a policy π, which forms a probability distribution over the available actions in $\mathcal{A}$, given the current state S.

$$\pi : S \rightarrow P(\mathcal{A}) \tag{2.22}$$

In order to approximate the optimal agent's behavior, we define the action-value function $Q^\pi(s, a)$, which gives the *return* of a given strategy π, after being in state s, and have taken an action a.

$$Q^\pi(s, a) = \mathbb{E}_\pi \big[R_t | s_t = s, a_t = a\big] \tag{2.23}$$

Also, the value function $V^\pi(s)$ evaluates the presence of the agent in state s, and is defined as,

$$V^\pi(s) = \mathbb{E}_\pi \big[R_t | s_t = s\big] \tag{2.24}$$

The optimal policy can be approximated through $Q^*(s, a)$, which is defined as the maximum expectation of the aforementioned *return* values over all policies π, such as,

$$Q^*(s, a) = \max_\pi Q^\pi(s, a) = \max_\pi \ \mathbb{E}_\pi \big[R_t | s_t = s, a_t = a\big] \tag{2.25}$$

The optimal action-value function, as appeared in Equation 2.25, satisfies the Bellman optimality equation, by which implies the following recursive relationship,

$$Q^*(s,a) = \mathbb{E}_\pi\big[r_t + \gamma \max_{a'} Q^*(s',a')|s_t = s, a_t = a\big] \tag{2.26}$$

The purpose of many RL algorithms is to estimate the $Q(s,a)$ function by using the Equation 2.26 as an iterative update. For example, the value and policy iteration approaches utilize the Bellman optimality equation into an update rule, and thus approximate the optimal $V^*(s)$ and $Q^*(s,a)$ functions, respectively, after a finite number of iterations. Nonetheless, in action-value function approximation for every sequence, it is implied that there is lack of generalisation during learning. For this reason, many RL approaches utilize a function approximator, of either linear or non-linear format, to estimate the optimal function. In the latter approaches, the neural network is utilized as the Q-network.

2.5.1 Deep Reinforcement Learning

Due to the emerging usage of deep learning techniques to extract high-level features, there was a need to apply such approaches to maximize agents' behavior performance in reinforcement learning applications. In 2013, Mnih V. *et al.* [4] presented a deep learning model for reinforcement learning based on Q-learning, namely the Deep Q-Learning (DQN), by showing its outstanding capabilities on seven Atari 2600 computer games only by using raw pixel information. In particular, this method outperformed in six out of seven Atari 2600 games in prior RL methods and also achieved better scores than expert human players in three of them.

Specifically, the DeepMind team proposed the usage of a convolutional neural network with weights θ as the function approximator $Q^*(s,a)$, instead using a linear formated estimator, as appeared in many RL approaches. In specific, the Q-network (as a neural network) is trained by minimizing the loss functions $L_i(\theta_i)$ that changes at each iteration i.

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim p(\cdot)}\big[(Q^*(s,a;\theta_{i-1}) - Q(s,a;\theta_i))^2\big] \tag{2.27}$$

The $Q^*(s,a;\theta_{i-1})$ is aqcuired from Equation 2.26 and it resembles the target in iteration i. The $p(s,a)$ is a probability distribution over the sequences s and actions a, defined as *behavior* distribution. Thus, by differentiating the loss function in Equation 2.27 with respect to the weights, the gradient $\nabla_{\theta_i} L_i(\theta_i)$ is occured, which can be approximated by optimizing the loss function by stochastic gradient descent. Overall, this approach is considered as $model\text{-}free$, since it does not estimate the environment $\mathcal{E}$, as it takes environment samples from the emulator directly. Also, it is characterized as *off-policy*, similarly to Q-learning, as we assume that a greedy policy is followed, with probability $1 - \epsilon$, during the estimation of $return$ value of action-state pairs.

Additionally, a vital feature of DQN approach is the integration of *experience replay* technique. Each time-step an experience is gained by the agent, namely the $e_t = (s_t, a_t, r_t, s_{t+1})$, this information is stored in an array D, which holds experiences over many episodes in to a *replay memory*. Hence, each time there is an inner iteration of the DQN approach, Q-learning or minibatch updates are implied on random experiences which are sampled from the D array. Given that, the agent follows the ϵ-greedy policy, by selecting the most greedy action given the current state. Overall, this method offers greater data efficiency compared with the Q-learning approach, it reduces the variance of the updates, by breaking the correlations between samples as it performs randomization, and finally it can avoid local minimum selections as it averages over many of its previous states and mitigates divergence in parameter approximation. Algorithm 1 presents the main body of the proposed DQN approach.

The DeepMind team performed preprocess steps to the Atari frames, before building the main body. These colored frames are initially transformed into 84×84 grayscale images to be sucessfully inserted in the 2D convolutional layers. Also, despite other works, this network took only the state representation as an input and resulted to the Q-values for every available action as its output. The main body of the presented neural network, contained 2 consequetive hidden convolutional layers of 16 and 32 filters, separately, and a fully-connected layer in the end, which had a single output for each action. This approach was evaluated and compared with state-of-the-art implementations, and presented significant advances among the vast majority of Atari games.

Algorithm 1: Deep Q-learning with Experience Replay

```
1  Initialize replay memory $D$ to capacity $N$;
2  Initialize action-value function $Q$ with random weights;
3  for  episode = 1, M do
4      Initialise sequence $s_1$=$x_1$ and preprocessed sequenced $\phi_1$=$\phi(s_1)$;
5      for  t = 1, T do
6          With probability $\epsilon$ select a random action $a_t$,
7          otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$;
8          Execute action $a_t$ in emulator and observe reward $r_t$ and image $o_{t+1}$;
9          Set $s_{t+1} = s_t, a_t, o_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$;
10         Store transitions $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$;
11         Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$;
12         Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
13         Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$;
14     end
15 end
```

Chapter 3

Problem Statement

3.1 Autonomy in Search-and-Rescue Robots

Search and Rescue situations can be quite demanding in terms of responders' operation, as there may be on hazardous areas, obstructed places, and remote locations, in which the responding team cannot perform immediate rescuing actions. For this reason, many responding teams have recruited robotic systems to facilitate the rescuing mission and ensure fast operation. Nonetheless, in such emergency scenarios, it may be inapplicable to successfully deploy and teleoperate rescuer robots (either ground or aerial), and thus there is a need for these systems to be able to fulfill their assigned missions autonomously.

Initially, the rescuing robots are equipped with a plethora of sensors, which make them "perceive" their surrounding world and plan their future moves. In reconnaissance, such systems have to search and find individuals that are in need and obtain important optical facts, that could enhance the rescuers' approach. For this reason, it is quite important for a robot system to successfully detect the requested objects, and avoid the inclusion of false information towards the rescuers. Henceforth, it is vital that the operating robot can fuse information from more than one onboard sensors, in order to validate its findings, and thus proceed with their exploration.

In addition, an important fact in such scenarios and environments, is the robot's movement availability. Search-and-Rescue scenarios may be occurred in inaccessible areas, in which there may be no sufficient ways to establish teleoperation with the operation robot, to monitor and intervene in its actions. As many emergency situations contain uncertainty in terms of area structure, because they are continuously changing, the operating robots should be able to quickly adapt to environment variations, and select the appropriate and safest moves along its path. For this reason, as the environments can differ, it is quite important that the deployed robot can make immediate decisions by itself and plan its path in the unknown environment.

Considering that, these robotic systems have limited specifications and operation time, each implemented algorithm should be able to run onboard and in real-time, while it consumes the minimum computational resources. Lastly, it is worth noting that, developed behaviors like this should be designed to be easily adaptable and installable on other robotic systems, which meet the necessary structure and sensory requirements, in order to help other responding teams setup and enhance their robotic co-worker's rescuing behavior.

3.2 Related Work

During the past decade, some notable studies and applications utilize robot systems to act in human-related operations. In many studies, ground robots [5, 6, 7] and aerial robots [8, 9, 10] act semi-autonomously in the operation environment, as they are monitored and teleoperated through an established control station, in order to fulfill a specific mission. Search-and-Rescue situations are one of the foremost reasons to employ robotic systems, like bushfires situations [11], in underwater scenarios [12], and even nuclear disasters like in Fukushima in 2011 [13] in which a plethora of different robots have been employed to investigate the nuclear accident areas. Also, there are studies, like Khasawneh A. *et .al* [14], which investigate the latency between the human responses and the UAV actions in a Search-and-Rescue scenario and suggest a set of measures to increase the trust between them. Meanwhile, these robots have been proposed and tested to operate in such scenarios under the control of a human team, to provide a broader view of the operation area. However, such scenarios may be at places that teleoperation is inapplicable or ineffective, due to the signal interferences, and thus, the robots have to be more autonomous and independent to proceed with their missions, despite the communication malfunctions with its human or robot cooperators.

Additionally, visual detection ability is vital for all robotic systems, as they should be able to interpret the viewing scene and recognize if any object of interest is in their view, regardless of the environment circumstances. In recent years, many visual detectors utilize convolutional neural networks (CNNs) to classify an image to the trained predicting categories [15, 16, 17] and also to apply object detection [18, 19, 20] within the captured frame, to approximate the rectangle occupied area of the image plane. Nonetheless, in many situations we need pixel-wise object detection [21, 22], which can be enhanced with batch normalisation [3] techniques and more a directive learning procedure, in order to develop more extensive and detailed vision applications that can be fused with other sensory data.

At the same time, it is also important to have a flexibility in the inspection of the surrounding world. There are publications that propose UAV systems with onboard gimbal units, equipped with sensory modules, in order to minimize its flight planning complexity. Specifically, Zhang *et al.* [23] develops a vision-based 3D geolocation method that calculates 3D coordinates of a target, using a stereo vision approach. Also, Sun *et al.* [24] developed an all-in-one camera-based target detection and positioning system, for fixed-wing UAVs and Search-and-Rescue scenarios, and evaluated it through simulation experiments. Even though the aforementioned research studies achieve target detection and spatial approximation in a global coordinate system, they highly depend on the used image processing approaches and corresponding camera unit specifications. For this reason, it is essential to utilize more precise sensors, combined with the acquired visual data, in order to perform target detection and positioning more efficiently.

Chapter 4

Our Approach

4.1 Simulated Robot Model and Sensors

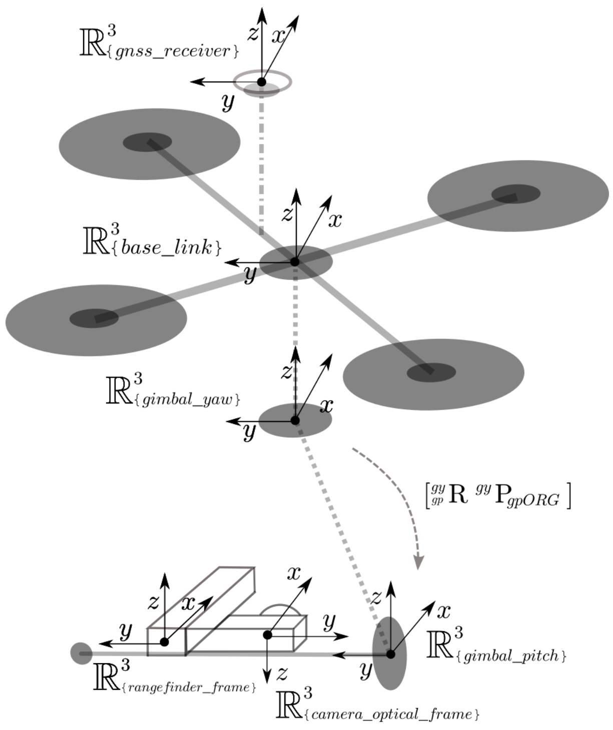
In this section, the proposed UAV system architecture will be described along with the onboard sensory modules, which are required to build the autonomous Search-and-Rescue quadcopter system. Specifically, an extended report for the onboard coordinate systems and their relations will be made, and details about the specifications of the used sensors will be presented.

4.1.1 Onboard Coordinate Systems and Spatial Transformations

A UAV, as a complex and rigid robotic system, can be described through a group of coordinate systems, namely frames. Moreover, each equipped sensor defines its coordinate system origin position, on which every spatial measurement can be referenced. For this reason, it is mandatory to define unique coordinate system for each onboard sensor. Despite that, UAV body points like its center of mass or a joint point of two relatively-connected moving parts can have their coordinate frame.

Assuming that $\{\mathcal{G}\}$ is the global coordinate system and $\{\mathcal{A}\}$ a local frame which indicates the UAV's body center point, the latter is expressed as,

$$\{\mathcal{A}\} = \{{}^{\mathcal{G}}_{\mathcal{A}}R, {}^{\mathcal{G}}P_{\mathcal{A}_{ORG}}\} \tag{4.1}$$



$\mathbb{R}^3_{\{gnss\_receiver\}}$
$\mathbb{R}^3_{\{base\_link\}}$
$\mathbb{R}^3_{\{gimbal\_yaw\}}$
$[{}^{gy}_{gp}\mathrm{R}\ {}^{gy}\mathrm{P}_{gpORG}]$
$\mathbb{R}^3_{\{gimbal\_pitch\}}$
$\mathbb{R}^3_{\{rangefinder\_frame\}}$
$\mathbb{R}^3_{\{camera\_optical\_frame\}}$
x
y
z

Figure 4.1: The UAV's onboard coordinate systems sketch.

where, ${}^{\mathcal{G}}_{\mathcal{A}}R$ is the rotation matrix that describes frame $\{\mathcal{A}\}$ relative to $\{\mathcal{G}\}$, and ${}^{\mathcal{G}}P_{\mathcal{A}_{ORG}}$ is the vector $[x, y, z]^T$ that localizes the origin of the frame $\{\mathcal{A}\}$ in the $\{\mathcal{G}\}$ global coordinate system.

Thus, in order to map a point P which was spatially described by the frame $\{\mathcal{A}\}$ to the frame $\{\mathcal{G}\}$, we define the transformation matrix,

$$
{}^{\mathcal{G}}_{\mathcal{A}}T = \left[\begin{array}{ccc|c} \multicolumn{3}{c|}{{}^{\mathcal{G}}_{\mathcal{A}}R} & {}^{\mathcal{G}}P_{\mathcal{A}_{ORG}} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{4.2}
$$

So,

$$
{}^{\mathcal{G}}P = {}^{\mathcal{G}}_{\mathcal{A}}R \cdot {}^{\mathcal{A}}P + {}^{\mathcal{G}}P_{\mathcal{A}_{ORG}} \Leftrightarrow
$$
$$
\Leftrightarrow {}^{\mathcal{G}}P = {}^{\mathcal{G}}_{\mathcal{A}}T \cdot {}^{\mathcal{A}}P \Leftrightarrow
$$

$$
\overset{(4.2)}{\Leftrightarrow} \begin{bmatrix} {}^{\mathcal{G}}P \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} \multicolumn{3}{c|}{{}^{\mathcal{G}}_{\mathcal{A}}R} & {}^{\mathcal{G}}P_{\mathcal{A}_{ORG}} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \cdot \begin{bmatrix} {}^{\mathcal{A}}P \\ 1 \end{bmatrix} \tag{4.3}
$$

The point ${}^{\mathcal{G}}P$ is described in global coordinates in frame $\{\mathcal{G}\}$. Also, to revert on to the initial coordinate system, the inverse of the homogeneous transform is used,

$$ {}^{\mathcal{A}}_{\mathcal{G}}T = {}^{\mathcal{G}}_{\mathcal{A}}T^{-1} = \left[\begin{array}{ccc|c} \multicolumn{3}{c|}{{}^{\mathcal{G}}_{\mathcal{A}}R^{T}} & -{}^{\mathcal{G}}_{\mathcal{A}}R^{T} \cdot {}^{\mathcal{G}}P_{\mathcal{A}_{ORG}} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{4.4}$$

In this way, every captured spatial measurement can be described in respect of any other UAV onboard coordinate system, as long as there is a direct transformation relation (connection) between them. Figure 4.1 illustrates a sketch of selected onboard coordinate systems that will be furtherly described in succeeding sections, with their main role within the scenario approach.

4.1.2 Simulated UAV Model and the Onboard Sensor Units

In our tests, we use the DJI Matrice 100 (M100) Gazebo model, which is based on the Hector Quadrotor model[1] [25]. This UAV model features quadcopter flight dynamics inside the Gazebo environment, and supports control commands (wrench, velocity and pose waypoints) as a real UAV system. Figure 4.2 shows the M100 model operation in the Gazebo world environment.

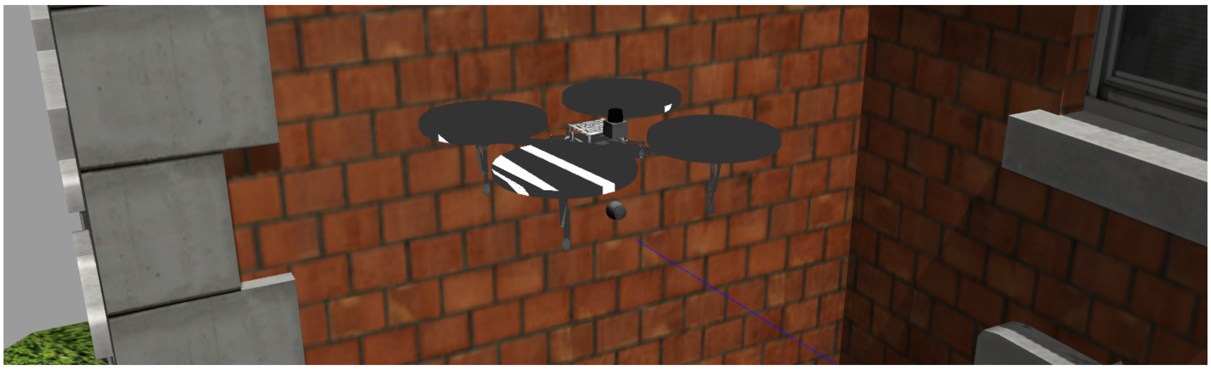


Figure 4.2: The DJI M100 quadcopter inside the Gazebo world.

Particularly, this model consists on various coordinate systems, like the $\{base\_link\}$ (center of mass) and $\{base\_stabilized\}$ (pose stabilized center of mass), and also there

[1]https://github.com/tu-darmstadt-ros-pkg/hector_quadrotor

is conterminous access to its groundtruth spatial relations through the Gazebo tool. Therefore, the accuracies of the mapping, detection, and global positioning procedures can be evaluated, as they can be cross-checked with the groundtruth values. Figure 4.3 illustrates the existing onboard frames of the UAV model, at a given moment, within the RViZ tool[1].

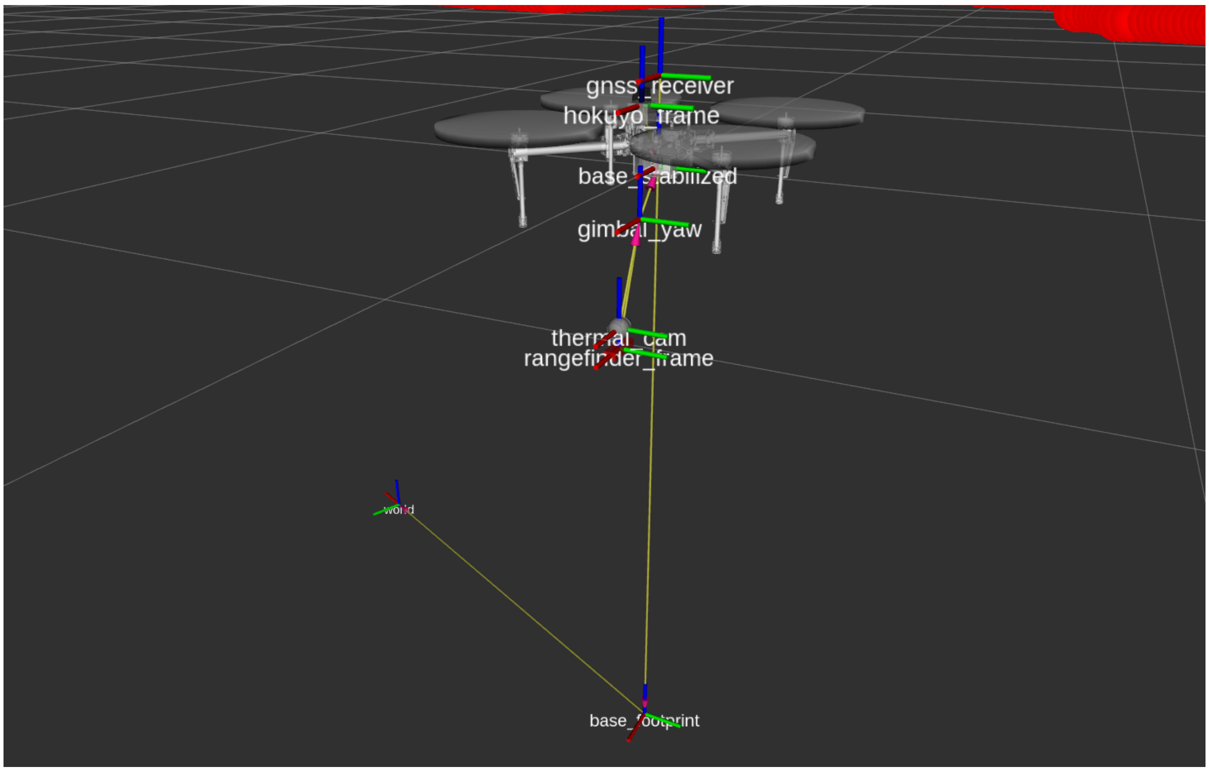


Figure 4.3: The simulation world and UAV coordinate systems, with their spatial transformation relations, as appeared in RViZ visualization tool.

Meanwhile, the Gazebo simulator features sensor plugin creation and development, by which we can simulate real sensor models in the simulator world. Hence, we included the appropriate sensors for the Search-and-Rescue approach in the UAV and gimbal simulated system models and setup their technical specifications, and noise models.

[1] https://github.com/ros-visualization/rviz

4.1.2.1 2D Laser-Scanner Module

To perform planar localization and mapping methods a two-dimensional LiDAR device is required, mounted onboard on the UAV system. By capturing surrounding distance measurements in reference with the ranging sensor coordinate system and geo-referencing them in respect of the UAV body frame, a local map of the area can be constructed and the position of the UAV can be approximated. For our scenario, we utilized the Hokuyo UTM-30LX-EW [26] sensor model, and we defined its coordinate system as the $\{hokuyo\_frame\}$. Figure 4.2 shows the simulated onboard 2D LiDAR sensor on the UAV body, and Figure 4.3 depicts the $\{hokuyo\_frame\}$ coordinate system inclusion.

4.1.2.2 GNSS Receiver Module and the Navigational Coordinate System

Additionally, in order to acquire global positioning data within the Gazebo environment, we selected the Hector Quadrotor GNSS receiver model that has been developed as a Gazebo sensor plugin tool. Specifically, this sensor plugin publishes navigation satellite fix data, based on $sensor\_msgs :: NavSatFix$[1] ROS message structure, which includes longitude, latitude and altitude information, along with the position's covariance matrix, the fix covariance type category and the Header field, which describes the measurement timestamp, and coordinate system. The fix data are defined using the World Geodetic System 84 (WGS84) reference ellipsoid.

For this reason, we included the global positioning sensor plugin in the UAV's Unified Robot Description Format file [2] (URDF), and placed it on a static position on its body frame ($0.2m$ above the $\{base\_link\}$ frame origin). The coordinate system of the GNSS receiver will be referred under the name of $\{gnss\_receiver\}$, as shown in Figure 4.4. Having that, the taken measurements can be spatially described and geo-referenced in respect of the selected navigational global system, like the *GPS*, *Galileo*, etc., depending on the running scenario.

Furthermore, the Gazebo tool uses by convention a right-handed coordinate system, with x,y values referring on the plane, and z defining the altitude value. Also, Euler angles are used to indicate objects' rotation in the three-dimensional space and thus their orientation in the field. Given that, each time a simulated world is loaded, the Gazebo

[1] http://docs.ros.org/api/sensor_msgs/html/msg/NavSatFix.html
[2] http://wiki.ros.org/urdf

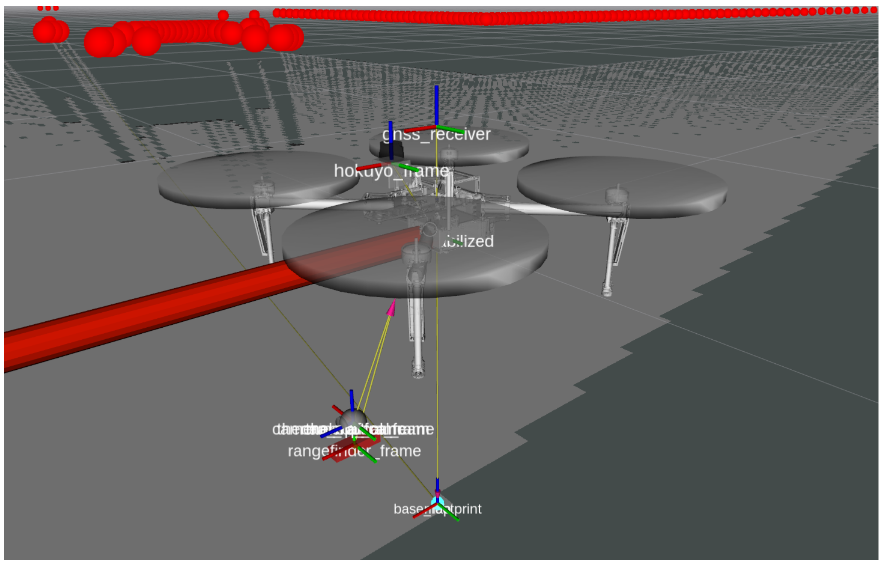


Figure 4.4: The DJI M100 quadcopter inside the Gazebo world.

initializes the global coordinate system by defining the the rigid *xyz* origin position and orientation, and maps it directly on the selected navigational coordinate system. Consequently, each point in the simulated world can be either be described in the Cartesian or the desired navigational system.

4.1.3 Simulated 3D Gimbal Frame and the Embedded Sensors Units

Meanwhile, since our purpose is to develop an autonomous UAV system that can easily deploy and act in a rescuing occurrence, the foremost feature is to ensure its adaptability in the new and unforeseen environment conditions. Therefore, to achieve sufficient area coverage and freedom during its detection and localization of objects of interest, we propose a gimbal unit to be attached underneath the UAV body, which will be equipped with the essential sensors to fulfill this purpose.

4.1.3.1 3D Gimbal Architecture and Specifications

Specifically, we developed a gimbal mechanism within the Gazebo environment, that has three degrees of freedom and is based on the *dji_m100_ros* GitHub organization model[1]. During the flight, the gimbal unit can perform 3-axis orientating and searching movements, and can either rotate independently of the UAV heading orientation or follow its movement heading, according to the current UAV searching status and priorities. The gimbal movements are motorized through the simulator environment and are limited to the gimbal model's architecture specifications.

Specifically, roll, pitch, and yaw (RPY) gimbal joints are created, which resemble the gimbal subsystem movements. A stable point is defined in the front of the UAV body as the gimbal's mounting point, namely the $\{gimbal\_yaw\}$ frame, and we denoted the RPY frames with their relations in order to construct the three-dimensional simulated gimbal module. The transformation between the $\{gimbal\_yaw\}$ and the $\{base\_link\}$ frames indicates the relative orientation of the gimbal frame with respect to the UAV heading global orientation. Figure 4.5 show the gimbal sub-frames relations, as appeared in RViZ visualizer tool. This illustration also depicts the $\{gimbal\_pitch\}$ frame, which corresponds to the second motor of the gimbal and defines the gimbal pitch axis.

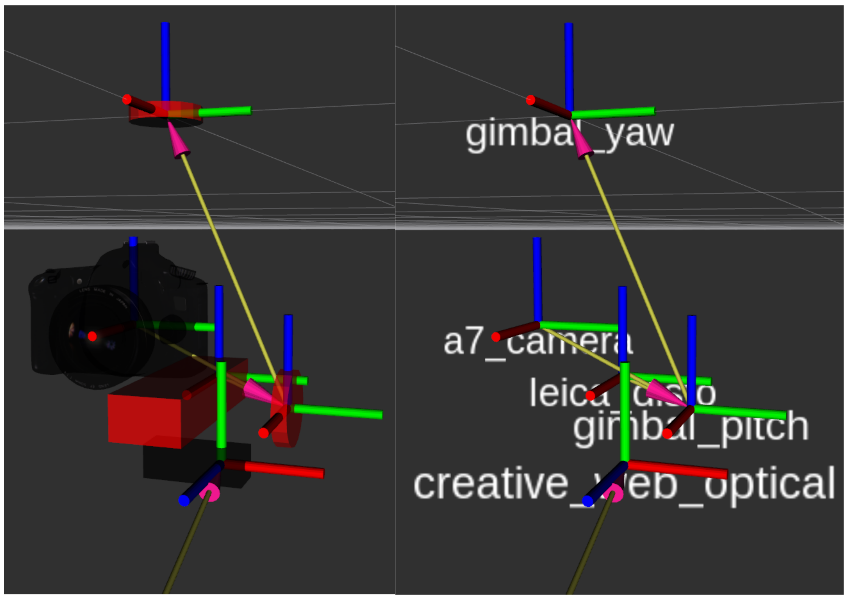


Figure 4.5: Gimbal System embedded frames relations as appeared in RViZ tool. Despite the illustration of the gimbal motor frames, a selection of embedded sensor frames is also included.

[1]https://github.com/dji-m100-ros

According with the current development, the gimbal *roll* movement is locked as there is no utilization of it during the target positioning procedure, and the *pitch* and *yaw* limitations are set to $[-\frac{\pi}{4}, \frac{\pi}{2.3}]$ and $[-\frac{\pi}{4}, \frac{\pi}{4}]$ radians, individually. Those values are selected according to real limitation ranges of common three-dimensional gimbal units on quadcopters. Thus, every embedded sensor is positioned on the same horizontal axis of the gimbal frame and their relative pose remains unchangeable during the gimbal axis orientation movements. Consequently, each spatial measurement of every embedded sensor, can be transformed to any other embedded sensor coordinate system, and also to any UAV's body frames or other equipped sensor frame (like the GNSS receiver), as long as there is a direct transformation between selected frames.

Henceforth, to detect existing objects of interest in the surrounding surveying area, we propose a gimbal frame equipped with two cameras, an optical- and a thermal-type respectively, and a 1D laser rangefinder module unit. The selection of these sensors is made mainly because they offer a cost-effective solution for real-world implementation, since there is no a particular requirement for their specifications, and also due to their sufficient combination to approach the detection and localization procedures.

Overall, the main behavior of the UAV gimbal system is to recognize requested objects in the simulation field, center and engage on the acquired target, and perform distance measurements to approximate the relative position and geo-reference the target in the rescuing map's coordinate system.

4.1.3.2 Color Camera Unit

Given the developed UAV's URDF model file, we included a camera module within the gimbal body. The selected camera module integrates a typical pinhole camera sensor, that captures RGB imagery with 1280×768 resolution and provides 30 frames per second (fps). The camera's main body coordinate system is initialized to the $\{camera\_frame\}$ frame, and the two-dimensional coordinate system on which every image plane detection is referenced, is defined as the $\{camera\_optical\_frame\}$.

Since the camera is attached to the gimbal frame, it is calibrated with a chessboard pattern object inside the Gazebo world. By applying a calibration procedure, the intrinsic camera parameters $\mathcal{K}$ and the distortion coefficients matrix $\mathcal{D}$ are acquired, as defined in Equations 2.1 and 2.2, respectively, which help us reference every 3D spatial data described in respect of the camera's body frame into their 2D image plane projection (image coordinates). Figures 4.6, 4.7 and 4.8 show moments during the UAV flight for the calibration of the color camera.

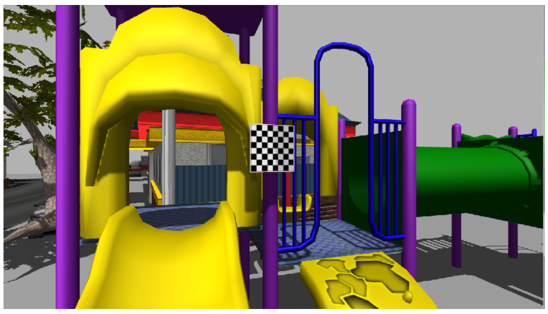


Figure 4.6: A chessboard snapshot using the color camera.

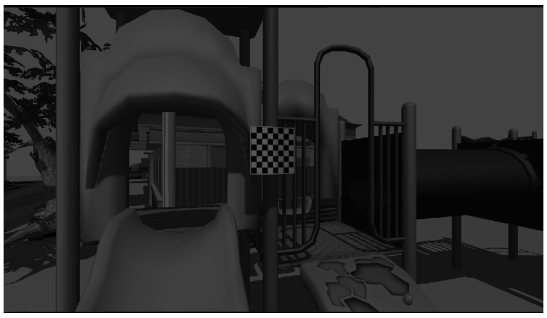


Figure 4.7: A chessboard snapshot using the thermography camera.

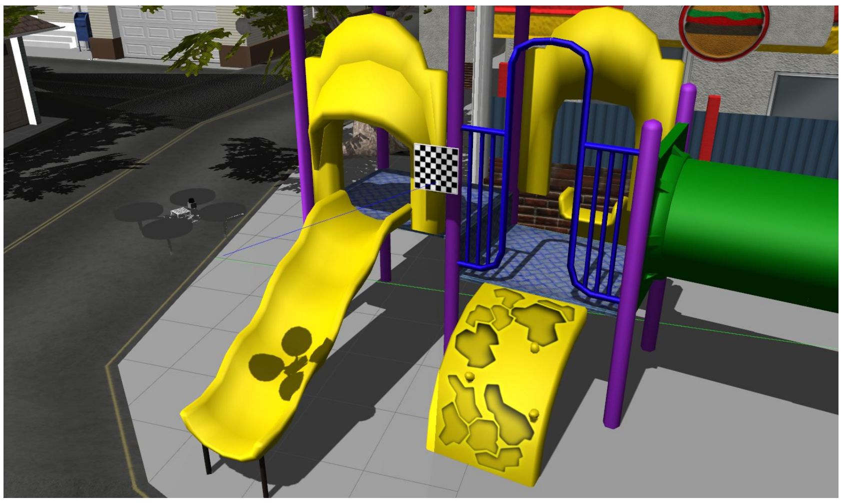


Figure 4.8: Third-person view of UAV flight for the cameras calibration procedure.

4.1.3.3 Thermal Camera Unit

As the UAV is operating in a Search-and-Rescue scenario, one of its primary goals is to locate individuals. For this reason, we enriched our gimbal system's "perception" capability with the addition of a thermography camera module, based on the Hector

Gazebo thermal camera plugin[1].

Similarly with the aforementioned color camera plugin, the thermal imaging camera module is defined as a pinhole camera model, which provides single-channel image data with the captured pseudo-temperatures of the Gazebo environment. The specifications of this camera module selected to be the same as the color camera, as it features 1280×768 resolution at 30 fps.

Nevertheless, since there is no temperature information within the Gazebo simulator objects, the thermal camera receives their warmth condition according to the percentage of red color into their body meshes. For this reason, we designed dedicated human models to contain the necessary red color in their meshes, and thus to be able to recognized as positives (high in temperature) objects by the simulated thermography camera module. Figures in 4.9 and 4.10 illustrate thermal objects in the Gazebo environment and their appearance in the simulated thermography raw-data.

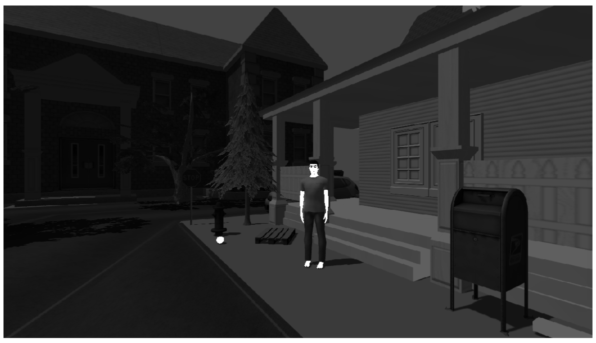


Figure 4.9: Thermography raw data.

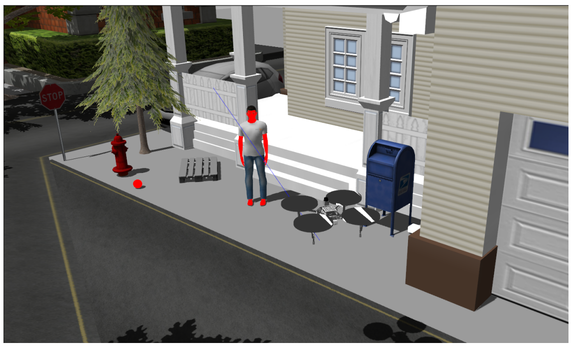


Figure 4.10: Thermal objects placed in the Gazebo world.

As a pinhole camera module, it is characterized by the $\{thermal\_cam\}$ and $\{thermal\_optical\_cam\}$ frames, on which every captured positive area is referenced, in 3D and 2D respectively. Also, a specific calibration procedure was followed for this camera module to obtain its intrinsic and distortion coefficient matrices, with the use of a dedicated custom chessboard pattern board as illustrated in Figure 4.7.

[1]http://wiki.ros.org/hector_gazebo_thermal_camera

4.1.3.4 1D Laser-Scanner/ Rangefinder Module

For the target positioning procedure, a one-dimensional laser rangefinder is utilized in order to measure UAV distances from the detected target. In specific, a common noise model is used, based on the *Leica DISTO D*110 sensor, which assumes a measurement tolerance of $\pm 1.5mm$ with a confidence level of 95%, in smaller than $5m$ distance measurements, and $\pm 3mm$ in other cases.

The coordinate system of this ranging unit is defined as the $\{rangefinder\_frame\}$, in which every distance measurement will be referenced.

4.1.3.5 Frames' Relations and the Transformation Tree

Overall, since the aforementioned frames' relations are developed and calculated, they are included in the transformation library's server, namely the tf [27], in order to be easily accessible during the UAV execution when a frames' relation is needed. Also, each transformation carries timestamp information, that indicates the exact moment in which this transformation was updated. This library is selected due to its plethora of handy transformation tools and its integration in the ROS environment. The total tf tree with its frame components, is depicted in Figure 4.11.

4.2 Autonomous Navigation and Obstacle Avoidance

In this section, the development of the UAV's autonomous navigation behavior will be discussed, along with noteworthy tools that were implemented to enhance its overall efficiency.

As we propose an autonomous Search-and-Rescue UAV system, a significant percentage of its autonomy refers to being able to move in the unknown area without any human supervision and intervention. For this reason, we focused on achieving such an autonomous behavior by using Deep Reinforcement Learning (DRL) approaches. Meanwhile, we developed a special OpenAI environment world to facilitate the development, modification, and training of our agent, and thus, we describe the transition and the deployment of the trained model into the Gazebo environment.

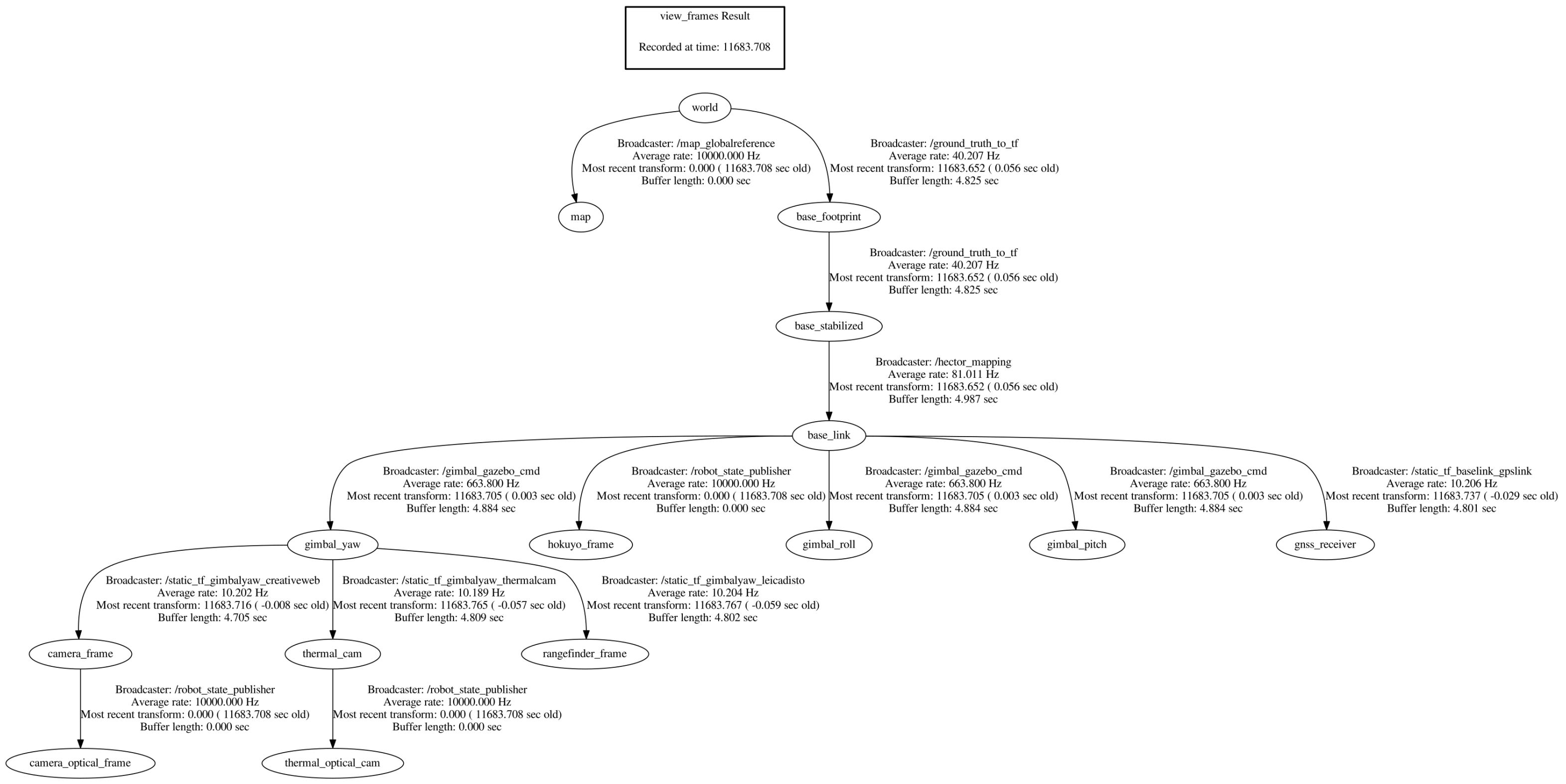

view_frames Result
Recorded at time: 11683.708
world
Broadcaster: /map_globalreference
Average rate: 10000.000 Hz
Most recent transform: 0.000 (11683.708 sec old)
Buffer length: 0.000 sec
Broadcaster: /ground_truth_to_tf
Average rate: 40.207 Hz
Most recent transform: 11683.652 (0.056 sec old)
Buffer length: 4.825 sec
map
base_footprint
Broadcaster: /ground_truth_to_tf
Average rate: 40.207 Hz
Most recent transform: 11683.652 (0.056 sec old)
Buffer length: 4.825 sec
base_stabilized
Broadcaster: /hector_mapping
Average rate: 81.011 Hz
Most recent transform: 11683.652 (0.056 sec old)
Buffer length: 4.987 sec
base_link
Broadcaster: /gimbal_gazebo_cmd
Average rate: 663.800 Hz
Most recent transform: 11683.705 (0.003 sec old)
Buffer length: 4.884 sec
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (11683.708 sec old)
Buffer length: 0.000 sec
Broadcaster: /gimbal_gazebo_cmd
Average rate: 663.800 Hz
Most recent transform: 11683.705 (0.003 sec old)
Buffer length: 4.884 sec
Broadcaster: /gimbal_gazebo_cmd
Average rate: 663.800 Hz
Most recent transform: 11683.705 (0.003 sec old)
Buffer length: 4.884 sec
Broadcaster: /static_tf_baselink_gpslink
Average rate: 10.206 Hz
Most recent transform: 11683.737 (-0.029 sec old)
Buffer length: 4.801 sec
gimbal_yaw
hokuyo_frame
gimbal_roll
gimbal_pitch
gnss_receiver
Broadcaster: /static_tf_gimbalyaw_creativeweb
Average rate: 10.202 Hz
Most recent transform: 11683.716 (-0.008 sec old)
Buffer length: 4.705 sec
Broadcaster: /static_tf_gimbalyaw_thermalcam
Average rate: 10.189 Hz
Most recent transform: 11683.765 (-0.057 sec old)
Buffer length: 4.809 sec
Broadcaster: /static_tf_gimbalyaw_leicadisto
Average rate: 10.204 Hz
Most recent transform: 11683.767 (-0.059 sec old)
Buffer length: 4.802 sec
camera_frame
thermal_cam
rangefinder_frame
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (11683.708 sec old)
Buffer length: 0.000 sec
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (11683.708 sec old)
Buffer length: 0.000 sec
camera_optical_frame
thermal_optical_cam

Figure 4.11: Coordinate systems' transformation and relation tree.

4.2.1 UAV Teleoperation and Action Server

In order to achieve exploration in the unknown operating area, the UAV system has to support teleoperation within the simulator environment. Throughout our work, we have utilized two ways of controlling the UAV system; either by publishing velocity commands to the simulated model or by using an action server that calculates the corresponding command controls to achieve a final UAV pose.

4.2.1.1 Velocity Control

The DJI M100 simulated model, as a Hector quadrotor model, supports velocity commands within the Gazebo environment, which are distinguished by the linear velocity vector and the angular velocity vector.

For this reason, we developed a *joystick* ROS node which takes input from the buttons and switches input of a computer gamepad, and according to the selected profile, it

```
$ rosmsg show geometry_msgs/Twist

geometry_msgs/Vector3    linear
    float64    x
    float64    y
    float64    z
geometry_msgs/Vector3    angular
    float64    x
    float64    y
    float64    z
```

Figure 4.12: The *geometry_msgs* :: *Twist* ROS message variable fields.

Linear or angular velocity commands can be applied on the UAV model along with its x,y and z axis, under the structure of the *geometry_msgs/Twist* message type which appears in Figure 4.12.

sends appropriate teleoperation commands to the simulated UAV. The main purpose of this node is to provide teleoperation access of the UAV control during its autonomous navigation, in case there is a need for an immediate action to avoid a certain situation or to command the UAV system to follow the desired path.

4.2.1.2 Action Pose Server

In the meantime, since our goal is to create an autonomous controlled UAV system, we attached the Hector quadrotor action tool[1] on the simulated DJI M100 drone, which features UAV pose command execution. As this tool's functionality is based on the action server of the ROS Navigation stack, this package features UAV velocity and wrench calculation given its current pose, in order to obtain a desired final pose. The desired UAV poses must be described in the $\{map\}$ or $\{world\}$ coordinate system, and thus the UAV has to know its position and orientation in respect to these frames.

Specifically, the action server captures a *hector_uav_msgs/PoseActionGoal* ROS message, which contains the new pose spatial description with respect to a coordinate system, its unique identification number, and the timestamp information of the requested command. Figure 4.13 shows the format and variable fields of a *PoseActionGoal* ROS

[1] http://docs.ros.org/kinetic/api/hector_quadrotor_actions/html/classhector_
_quadrotor__actions_1_1PoseActionServer.html

```
$ rosmsg show hector_uav_msgs/PoseActionGoal

std_msgs/Header   header
    uint32   seq
    time   stamp
    string   frame_id
actionlib_msgs/GoalID   goal_id
    time   stamp
    string   id
hector_uav_msgs/PoseGoal   goal
    geometry_msgs/PoseStamped   target_pose
        std_msgs/Header   header
            uint32   seq
            time   stamp
            string   frame_id
        geometry_msgs/Pose   pose
            geometry_msgs/Point   position
                float64   x, y, z
            geometry_msgs/Quaternion   orientation
                float64   x, y, z, w
```

Figure 4.13: The *hector_uav_msgs/PoseActionGoal* ROS message variable fields.

message. Each time a message is captured, the corresponding velocity commands are calculated and executed by the simulated UAV to obtain the desired position and orientation values. The server makes pose corrections under predefined criteria, like the relative distance of the UAV and the goal position, or a pose timeout parameter. In the end, if any of the restrictions are enabled, or in contrast, the pose has been acknowledged and acquired, the server provides a *actionlib_msgs/GoalStatus* response which describes the current situation. The message format can be distinguished in the following status categories,

(i) PENDING : The goal has yet to be processed by the action server.

(ii) ACTIVE : The goal is currently being processed by the action server.

(iii) PREEMPTED : The goal received a cancel request after it started executing and has since completed its execution (Terminal State).

(iv) SUCCEEDED : The goal was achieved successfully by the action server (Terminal State).

(v) ABORTED : The goal was aborted during execution by the action server due to some failure (Terminal State).

(vi) REJECTED : The goal was rejected by the action server without being processed, because the goal was unattainable or invalid (Terminal State).

(vii) PREEMPTING : The goal received a cancel request after it started executing and has not yet completed execution.

(viii) RECALLING : The goal received a cancel request before it started executing, but the action server has not yet confirmed that the goal is canceled.

(ix) RECALLED : The goal received a cancel request before it started executing and was successfully cancelled (Terminal State).

(x) LOST : An action client can determine that a goal is LOST. This should not be sent over the wire by an action server.

4.2.1.3 Automated/Preplanned Flights

As an extent to the UAV's pose execution functionality, as mentioned in Section 4.2.1.2, we created a separate node that sends sequential pose commands on the UAV system taken from a list of poses. Specifically, in the target positioning procedure, it is essential to be able to repeat flight trajectories, in order to further evaluate the precision and efficiency of the positioning approaches. Hence, the developed ROS node can process a *.csv* file which includes distinct UAV poses and can parse them to the simulated drone system to perform them consecutively. The given poses can be written in respect of the following formats,

$$\bullet\ x, y, z, \text{Euler}(roll, pitch, yaw) \tag{4.5}$$

$$\bullet\ x, y, z, \text{Quaternion}(x, y, z, w) \tag{4.6}$$

The position data are expressed in meter units and the orientation Euler angles in degrees. Also, through this script, we can adjust the action server parameters like the smoothness or the accuracy of the UAV's positioning procedure, as well as the speed of flight. During the execution of the sequential poses, joystick controlling intervention is permitted, in case there is a need to take manual control of the drone or to perform a correction.

Also, an automated GeoGebra [28] tool has been developed which can calculate the desired amount of UAV poses based on the drawn trajectory within the tool. Initially, the number of the desired UAV poses is selected, which is desired according to our scenario and target position, and thus the script computes the poses and creates a *.csv* file with them in respect of the aforementioned saving formats, in Equations 4.5 and 4.6. Figure 4.14 shows 20 UAV poses, named with letters from B to U alphabetically, that are calculated by selecting a circular trajectory around the given marker, which is located at (-3,2,1). As it is apparent from the GeoGebra script, there can be also more straight or hyperbolic-lined calculated trajectories.

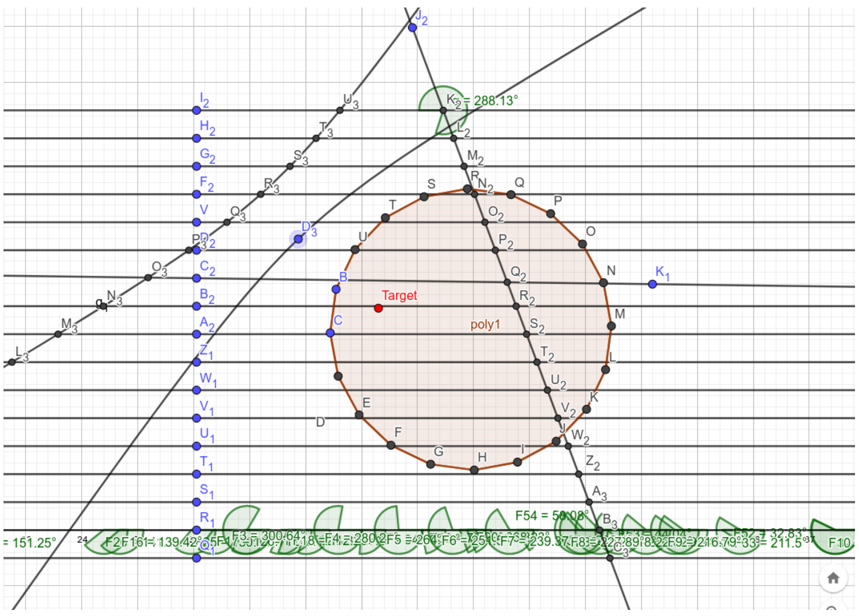


Figure 4.14: GeoGebra tool script for extracting UAV poses *.csv* files given a target position.

4.2.2 UAV Control and OpenAI Gym Environment

The GazeboSim tool, as a fully operational robot and physics simulator, requires moderate to high computational resources during its runtime. This fact makes it ineligible for deep reinforcement learning training and agent development purposes, as it may be time ineffective. For this reason, we developed a lightweight OpenAI environment that resembles the control of the UAV in the Gazebo simulation, to facilitate and accelerate the agent's behavior development and training procedures.

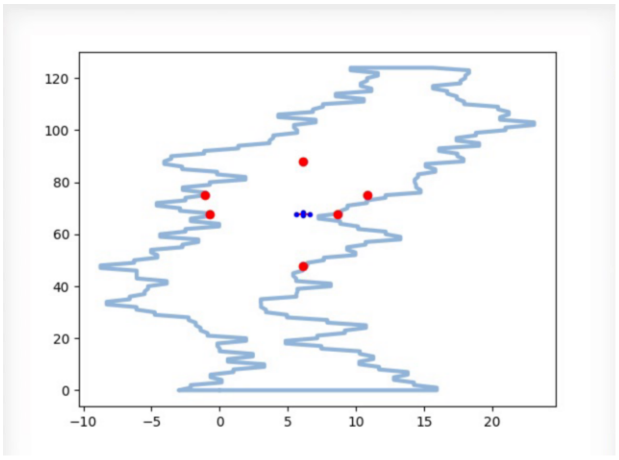
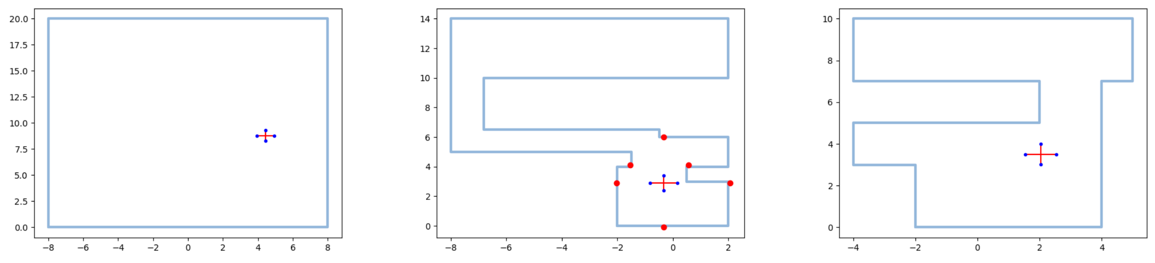


Figure 4.15: The simulated UAV and the maximum range distances from its onboard LiDAR.

Specifically, given the initiative of A. Sakai in "drone_trajectory_following"[1] repository, we constructed a 2D UAV simulator world as appeared in Figure 4.15. The current UAV model supports only translation commands in both x and y dimensions, without any orientation movements. Henceforth, as we want to develop an autonomous navigation behavior for the UAV, we included map addition capabilities inside the python simulator tool. The maps must be formed in polygon shapes, as their boundaries (walls) are formed by their shape coordinates. Figures in 4.15, 4.16, and 4.17 show instances of created maps.

Additionally, in order to achieve safe navigation in the generated environment map, the UAV had to capture distance measurements from its surrounding area. For this rea-

[1] https://github.com/AtsushiSakai/PythonRobotics/tree/master/AerialNavigation/drone_3d_trajectory_following



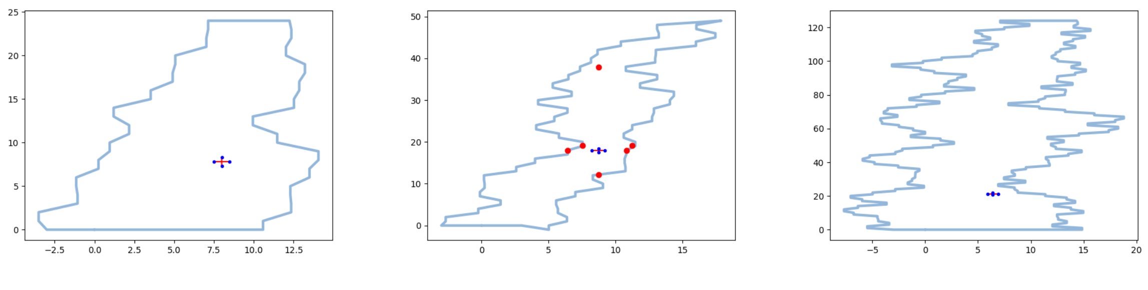
(a) (b) (c)

Figure 4.16: Examples of polygon maps.

son, we developed a simulated LiDAR behavior for the UAV system, that gathers ranging information in 6 specific directions. Particularly, by assuming that the UAV heading orientation is $\theta = 0°$, distance measurements are captured at $\theta = [0°, \pm45°, \pm90°, 180°]$ angles. So each time new LiDAR measurements had been obtained, the intersection points of the laser raycasting projections with the polygon map were approximated, and the distances of the end-points were calculated. Given that, these distance measurements were referenced in respect to the UAV center of mass, which is the exact UAV 2D position, and a white Gaussian noise is also included in their values, with standard deviation of 0.01m. Figures 4.15, 4.16(b), and 4.17(b) show the red dots which illustrate the LiDAR's distance measurements in various UAV positions. Also, each distance measurement was bounded by an upper limit, in order to further reduce the agent's perception abilities and simulate real LiDAR performance. Figure 4.15 illustrates a maximum detected range in the front direction of the simulated UAV, as there is no existing obstacle.

The reason of selecting a limited amount of distance measurements was to evaluate the UAV autonomous navigation efficiency in narrowed perception capabilities. Also, the plethora of the front distance measurements (as only one is made in the back) was to focus on the ability of the UAV to perform forward movements.

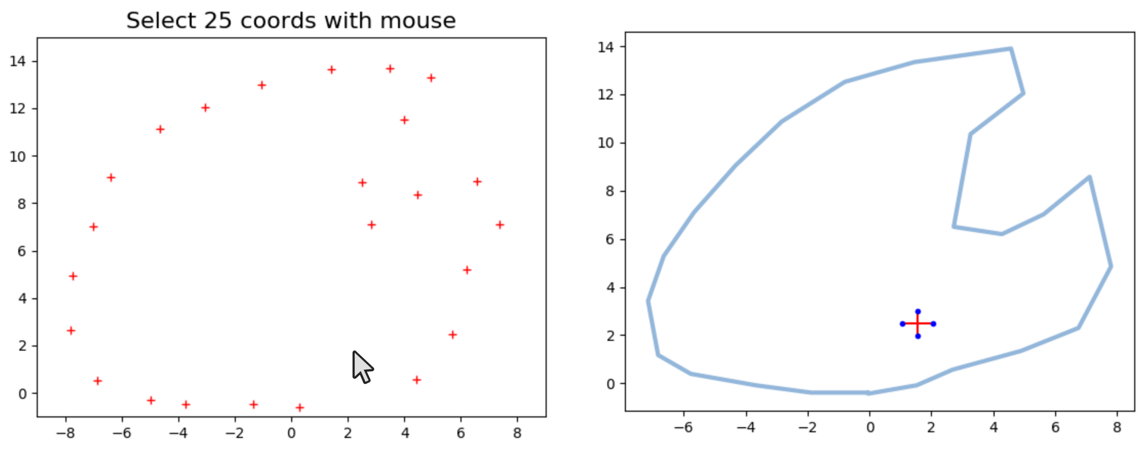
Meanwhile, to enhance the agent's learning capability, a random maze generator library was also implemented and included in the simulator tool environment. Through this application, a new maze world can be constructed, based on the desired length size along y-axis and the desired lengths of the narrowest and widest path along its structural skeleton. By this way, we generated new maps along the progress of the training procedure, in order to enrich the generality of its behavior. Figures in 4.17 show some



(a) (b) (c)

Figure 4.17: Autogenerated polygon maps, with various spatial properties (length, narrowness, width).

examples of autogenerated maps, under various building preferences. Given the requested spatial properties, the maps are generated randomly by choosing wall positions based on samples taken from a normal distribution.



(a) (b)

Figure 4.18: Drawing a custom polygon map of 25 edge points.

Also, a drawing functionality has been included to support custom-maps creation. The user selects the amount of the map edge points and can click on their positions on an empty plot as appeared in Figure 4.18(a). Since all the points have been selected, the map is formed (as depicted in Figure 4.18(b)), stored locally, and thus, can be fully-utilized as an OpenAI Gym environment.

In the meantime, since our goal is to develop an autonomous UAV behavior for the Gazebo simulator, we had to include such map objects in the simulator world. Thus, we expanded the functionality of this library to construct polygon maps in respect of the Gazebo building structure format. In this way, 3D map objects can be created according to autogenerated or custom-generated OpenAI Gym maps, and be imported into the GazeboSim. Figure 4.19 shows a variety of polygon maps that are created and added in the Gazebo environment, along with the simulated UAV model.

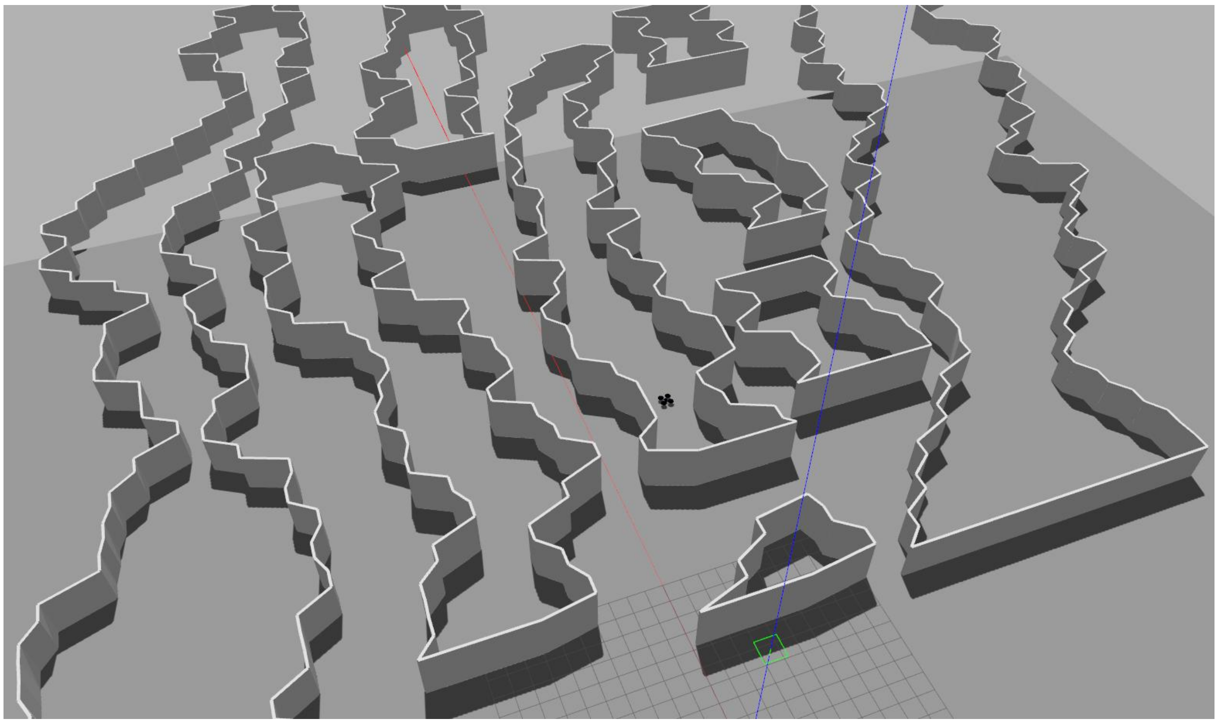


Figure 4.19: Autogenerated 3D maps for the GazeboSim under various specifications, in terms of length (size) and narrowness levels.

4.2.3 Creating the Autonomous Navigation Behavior

As mentioned in Section 2.5, Reinforcement Learning is broadly used in the development of autonomous agent behaviors. Due to the emerging advancement of embedded platforms, there is a plethora of commercial and developer UAVs that are equipped with onboard Graphical Processor Units (GPUs), as on the DJI Matrice series, which can facilitate online processing. Henceforth, in order to develop the autonomous flight behavior, we deploy the Deep Deterministic Policy Gradient (DDPG) [29] approach, which is a model-free and off-policy algorithm based on the Deep Q-network approach, as described in Section 2.5.1, and is used to learn policies in continuous action spaces. This approach utilizes an actor-critic scheme and it is capable of learning policies through low-dimensional observations as in our scenario.

Deep Deterministic Policy Gradient

DDPG algorithm combines Q-learning and Policy Gradients techniques, through an actor-critic approach, as it forms two distinct neural networks to learn the policy (actor) and the Q-function (critic). The goal of the actor is to extract an appropriate action for the agent given the current state. Thus, it takes as an input a flattened vector that contains the UAV observations, namely the 6 LiDAR measurements, and approximates the UAV actions. The network's output is a two-dimensional vector that holds information about x and y dimension movement, in the value range of $[-1, 1]$. On the other hand, the critic network is utilized to evaluate a certain action execution, by knowing the current UAV state. Henceforth, as this network takes the concatenation of the current observations and the selected action from the actor's network, it approximates the outcome Q value for this input.

In terms of architecture, both neural networks contain 3 same-sized dense layers with ReLU activation functions and batch normalization modules in between them. The final layers for both networks have linear activation function, but they result in different sizing outcomes as mentioned before. In our tests, we utilized various architectures for further system evaluation.

In the meantime, during the training procedure DDPG uses experience replay, similarly to DQN in Section 2.5.1, to learn from accumulated experiences that have gained from the past. Additionally, even though the observations are only distance measurements, which are clipped in predefined value ranges, we use normalization techniques to facilitate DDPG training and application procedures. Batch normalization is included within the networks' architecture to mitigate the covariance shift during training by ensuring that each layer receives a normalized input. It is also noteworthy, since DDPG is an off-policy algorithm it can isolate the exploration problem from the learning algorithm. By adding generated noise on the actor policy, sampled by a correlated normal distribution generated by an Ornstein-Uhlenbeck process, it achieves more efficient exploration.

Reward Function and Behavior

A vital requirement for the agent's training was the design of the reward function. In our approach, the UAV has an onboard gimbal system that can act and rotate independently from the main body. In particular, during the exploration, the UAV should be able to detect possible targets and thus rotate towards them. In case of the extreme left or right positions of the engaged targets relative to the UAV heading direction, it performs rotation movements to center its heading direction towards the given target location. Having that, the UAV approaches the target and localizes it during its movement towards it. Hence, as it may be clear, we impel a forward movement for the UAV towards the target location, since one is detected and the reward function is formed with respect to this purpose.

Particularly, the UAV system is positively rewarded every time it performs a successful one-meter forward movement (passes a checkpoint) in the map because it is considered that it further approaches the target. Meanwhile, in order to ensure its safe operation, we penalize flights close to walls, depending on the captured range distances from the LiDAR system. Also, each time the UAV collides on a wall or even gets out-of-bounds from the current polygon map, it gets the maximum negative reward with a terminal state. In this way, the UAV learns to be in safe distances from surrounding walls and thus it learns to avoid occurring obstacles while it tries to go forward.

Furthermore, each generated map has a unique area zone that is considered as the goal terminal state. This zone is defined according to the map length, which is always located 3 meters before its endpoint (for example in Figure 4.17(b) is for $x = 45$ for $map\_length = 48$). Thus, as soon as the UAV manages to pass this point, it is awarded the maximum reward, and the episode is terminated successfully. It is worth noting, each time the UAV performed a backward movement it gets a reduced reward in order to prefer forward exploration.

Throughout our experiments, we will conclude to the most appropriate actor-critic network setup, by training under various loss functions, network sizes, number of episodes, and learning environment specifications.

4.3 Autonomous Target Detection and Gimbal Aiming

In this part, the machine vision procedures will be described to perform autonomous target detection in the viewing field. As the UAV is equipped with both optical and thermography cameras, there will be a detailed report about their data manipulation and also their fusion, in order to recognize effectively any observed target. Alongside this report, the gimbal's autonomous aiming and engaging behavior will be presented according to the detected target's situation.

4.3.1 Target Recognition through Optical Data

In order to perform an effective, on-board, and real-time object detection in SAR environments, we implemented a custom neural network detector and tested it under both real-world and GazeboSim datasets. Initially, since our future goal is to develop a real UAV system to operate in emergency situations, the developed autonomous detector was initially trained and tested under real imagery data. Therefore, as we conclude with the appropriate network architecture and characteristics, the system was trained from scratch on a custom dataset that was created within the Gazebo software and tested in various simulation environments.

4.3.1.1 Real-World Dataset

One of the most broadly used datasets for building and training object detectors is the Microsoft Common Objects in Context (COCO) [30], which contains over 200K labeled images on over 80 classes. Every image have been pixel-wise labeled with the appearing class instances and contains additional contextual information for further scene understanding. In our case, in order to design and test the robustness of our detector, we utilized the two major COCO dataset releases, from 2014 and 2017, which contained over

250K labeled images with appearing people in everyday scenes. Since our primary goal is to search for humans, it is vital for our detector to be effective in terms of localizing every appearing person in the scene and also to be precise, by avoiding the inclusion of false-positive or false-negative areas during the detection.

4.3.1.2 Image Normalization and Standardization

A common issue with many open imagery datasets is the variations of the photographed scene conditions and object visualizations. Many datasets contain images taken from different cameras and lenses, under various lighting conditions, and therefore same objects may appear slightly or notably different among these instances. This fact worsens the training procedure and affects the detector's precision, which thus may delay its learning convergence. Hence, it is essential to process appropriately every image prior to its utilization in the training or testing procedure, in order to avoid such implications.

Primarily, in order to mitigate the divergence of the brightness and the color intensities among the images, we standardize the images to form a Gaussian distribution with zero mean and standard deviation equal to one. Particularly, each time before using a new dataset, we extract the overall mean and standard deviation over all images for each channel separately. This procedure is performed independently for each image channel, which in our scenario indicate the three color values (RGB), and are formed by pixel values $x_i = [r_i, g_i, b_i]$ where $i \in [1, number\_of\_pixels]$.

$$\mu_{\{r,g,b\}} = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \text{and} \qquad \sigma^2_{\{r,g,b\}} = \frac{\sum_{i=1}^{N}\left(x_i - \mu_{\{r,g,b\}}\right)^2}{N} \tag{4.7}$$

Since these values are acquired, each image pixel $x_i = [r_i, g_i, b_i]$ is formed as,

$$x'_i = \left[\frac{r_i - \mu_r}{\sigma_r}, \frac{b_i - \mu_g}{\sigma_g}, \frac{b_i - \mu_b}{\sigma_b}\right] \tag{4.8}$$

Additionally, an essential procedure after the standardization is the normalization. The normalization procedure is essential for the neural networks, in order to keep the same scale during weight updates. Specifically, during training, if every weight update has a different scale, the neural network would need a different learning rate to achieve convergence. Figures in 4.20 illustrate three raw images along with their standardized transformation, as was applied over their RGB channels.

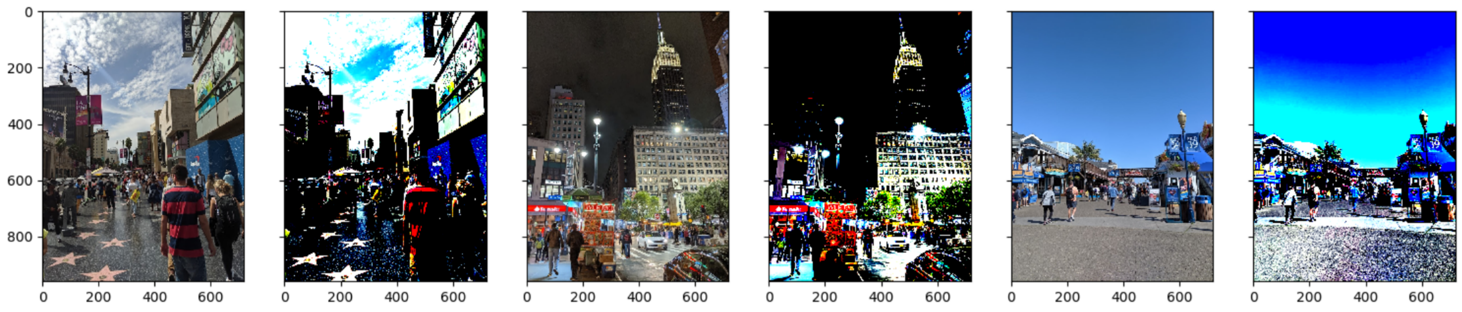


Figure 4.20: RGB images with their standardized depictions.

4.3.1.3 Neural Network Architecture

Initially, our idea in the network architecture is inspired by the VGG16 net [31] convolutional architecture, due to their ImageNet [32] challenge distinctions in 2016 and 2017. The VGG16 network utilizes 16 weight layers with 3×3 kernel-sized filters and three fully-connected layers in the end, with the last having a softmax layer of 1000 channels for the object classification task. At the same time, approaches like Hyeonwoo N. *et al.* [33] add deconvolutional layers and perform unpooling techniques to retrieve bigger feature maps with smaller depths. The goal of these approaches is to obtain high-resolution semantic maps, to determine in pixel-wise way areas that belong to specific classes. In 2015, Ronneberger O. *et al.* [21] presented the U-Net, a fully convolutional network for biomedical purposes, which had the shape of an autoencoder along with forward connections between the convolution and deconvolution parts. This network structure could learn how to upscale the reduced-sized information, appropriately, by selecting features from the convolutional part, and thus increase the resolution of the output and conclude to the input-sized prediction map.

In our approach, we developed an autoencoder-alike convolutional deep network inspired by the aforementioned publications, equipped with a custom convolution and deconvolution part along with skipping connections for feature map enhancement. Figure 4.21 illustrates the detailed configuration of the entire deep network. Specifically, we utilize a 9 layer convolutional network without any fully-connected layer, and we use upsampling techniques through retrieved skipping connections from the convolutional part, to concatenate feature maps information and deconvolute to obtain the final input-sized

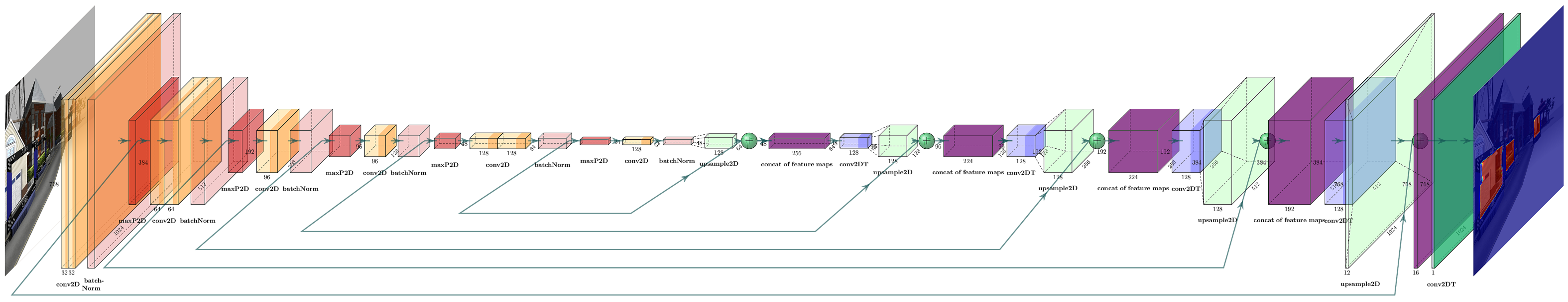


Figure 4.21: **The Network Architecture**. The network takes the color camera's raw data as an input, and through feature extraction and the deconvolution procedure retrieves a heatmap array that contains the trained semantic map.

object semantic segmentation.

Pooling and batch normalization, as described in Section 2.4, operations are performed between the convolutional layers. The latter method is applied to minimize internal-covariate-shift effect of convolutional layers. In the deconvolution network part, feature maps concatenations are applied with nearest window interpolation upsampling, instead of unpooling techniques, to let the network determine the useful spatial information to achieve the requested heatmap array activations. Since we combine spatial information from previous normalized convolutional layers with the approximating the semantic map activations, we allow the network itself to extract the vital data features which are occurred from the higher resolution imagery information and the spatial information.

4.3.1.4 Advanced Metrics and Loss Functions for Semantic Segmentation

Meanwhile, neural networks that are used in semantic segmentation approaches, focus mainly on image segmentation which is based on detected context through pixel-wise classification. For this reason, there is a plethora of metric functions to evaluate the precision and accuracy of such image areas, as they are formed by the detector's prediction. Hence, we developed numerous broadly-used metrics as Keras metrics modules, to extensively monitor the training procedure, and also as loss functions, to use them for training the models and targeting specific detector behaviors.

As described in Section 2.4.1, there is a variety of important metrics to evaluate the performance of a semantic segmentation detector. Given these references and the confusion matrix definition in Equation 2.11, we utilize in our approach notable and broadly used loss functions and metrics, that can handle TN and TP amounts differently, as they enhance the ability of the detector in either detecting positives or negatives more accurately. In specific,

F-1 Score The F_1 Score metric, also known as the Dice Loss (DL), is a broadly used function to measure the overlap of the prediction segmented image and the groundtruth. This metric is based on the *Precision* and *Sensitivity*(*Recall*) parameters of the test.

$$F_1 = \frac{2 \cdot |Y \cap \hat{Y}|}{|Y + \hat{Y}|} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} = \frac{2 \cdot \mathrm{TP}}{2 \cdot \mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \tag{4.9}$$

Fβ Score Similarly to the F_1 Score, the F_β score utilizes a real factor β to adjust the *Sensitivity* parameter importance compared to *Precision*. This metric is formulated as,

$$F_\beta = (1+\beta^2) \cdot \frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall} = \frac{(1+\beta^2) \cdot \mathrm{TP}}{(1+\beta^2) \cdot \mathrm{TP} + \beta^2 \cdot \mathrm{FN} + \mathrm{FP}} \tag{4.10}$$

As it is obvious, in case of β parameter equal to 1, the F_β score becomes the F_1 score.

Tversky Loss The Dice loss function, as appeared in 4.9, weights and penalizes evenly the FP and FN proportions. In 2017, Seyed Sadegh Mohseni Salehi *et al.* [34], used the Tversky features [35] and presented a loss function in order to weight FN more than the FP, to achieve robust detection of small lesion areas in medical images. Hence, by initializing the magnitude parameters α and β, he proposed the loss layer based on Tversky index,

$$\text{Tversky Loss} = \frac{Y \cap \hat{Y}}{Y \cap \hat{Y} + \alpha(Y^c \cap \hat{Y}) + \beta(Y \cap \hat{Y}^c)} = \frac{TP}{TP + \alpha(FP) + \beta(FN)} \tag{4.11}$$

Thus, by adjusting the parameter α and β, this formulation can adjust the trade-off between FP and FN. It is noteworthy, if α and β values are equal to 0.5, the Tversky metric becomes the F_1 score. Also, by assuming that $\alpha + \beta = 1$, each pair implies the F_β score formulation.

Intersection over Union (IoU) The Intersection over Union [36], is also a common metric which is used in semantic image segmentation, which is written as,

$$\mathrm{IoU}(Y,\hat{Y}) = \frac{Y \cap \hat{Y}}{Y \cup \hat{Y}} = \frac{TP}{TP + FP + FN} \tag{4.12}$$

The Equation 4.12 can be expressed in respect of the Dice loss metric, as follows,

$$\frac{A \cap B}{A \cup B} = \frac{TP}{TP + FP + FN} = \frac{Dice}{2 - Dice} \tag{4.13}$$

and the mean IoU (for two classes) is defined as,

$$\text{Mean IoU}(Y,\hat{Y}) = \frac{\mathrm{IoU}(Y,\hat{Y}) + \mathrm{IoU}(Y^c,\hat{Y}^c)}{2} \tag{4.14}$$

Likewise, in case of $\alpha = \beta = 1$, the Tversky score resembles the IoU metric.

Throughout our experiments, we will evaluate the performance of the optical detector as described in Section 4.3.1.3, by training on various semantic loss functions and a common imagery dataset, to perform human detection. In specific, we utilize a plethora of the aforementioned loss and metric functions, and we create combinations of them to furtherly check the detection efficiency. In this way, we conclude with the final models of the trained detectors and we examine their diverse performances on TP and TN detection and their overall behavior.

4.3.1.5 Gazebo Human Models and Dataset Creation

As the detector has been tested under the COCO dataset on real imagery data, we had to re-train it based on data captured from the Gazebo simulator, in order to attach it to the simulated UAV system and perform object detection in the Gazebo environment. Thus, we had to create a dataset with the same format of the raw and annotated images as the COCO dataset, with the new data captured from the Gazebo simulator.

First, we use human models from the Gazebo's open-source model database, as appear in Figure 4.22, to simulate the real SAR scenarios. As it may be clear, the training procedures on Gazebo's data may be notably better than the real-world testing, as they are less realistic and detailed in terms of textures and lighting conditions. However, this is the main reason that we have previously evaluated it on a real-world dataset, in order to ensure its efficiency.

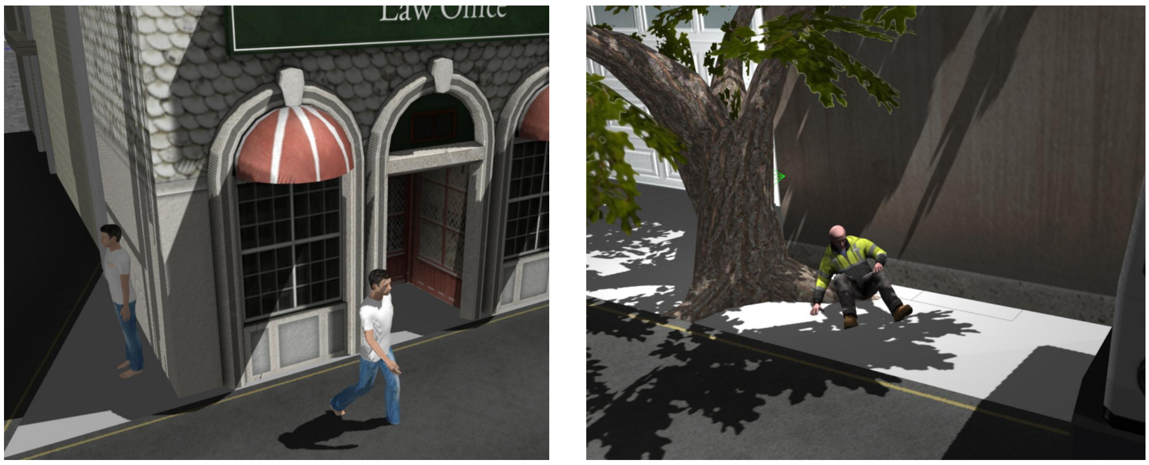


Figure 4.22: Human models in Gazebo world.

In specific, as the UAV and the human objects are considered as rigid bodies inside the Gazebo world, they are described by their local coordinate system. Thus, for each used model within the Gazebo world, such as the human objects, we define their local coordinate system in respect of the simulator's global coordinate system and broadcast it on the tf tree to be viewable within the ROS environment. In this way, we can spatially correlate the existing objects, according to their current status in the simulated world. Figure 4.23 shows coordinate system relations, as appeared in the RViZ tool, between the UAV and a human object.

In order to capture raw images and their annotated instances which indicate the areas of the depicted objects, we utilized the ground-truth spatial information of the existing Gazebo objects. Since the UAV has an integrated camera which is calibrated, we can instantly project any 3D world point on its image plane and extract its x, y pixel position. Nonetheless, in contrast with the COCO pixel annotations, we mark a predefined surrounding area of the human object as the positive area, by utilizing four coordinates in certain positions with respect to the object's local coordinate system. Specifically, we define four points at $\pm 1m$ in z and $\pm 0.7m$ in y direction, with respect to

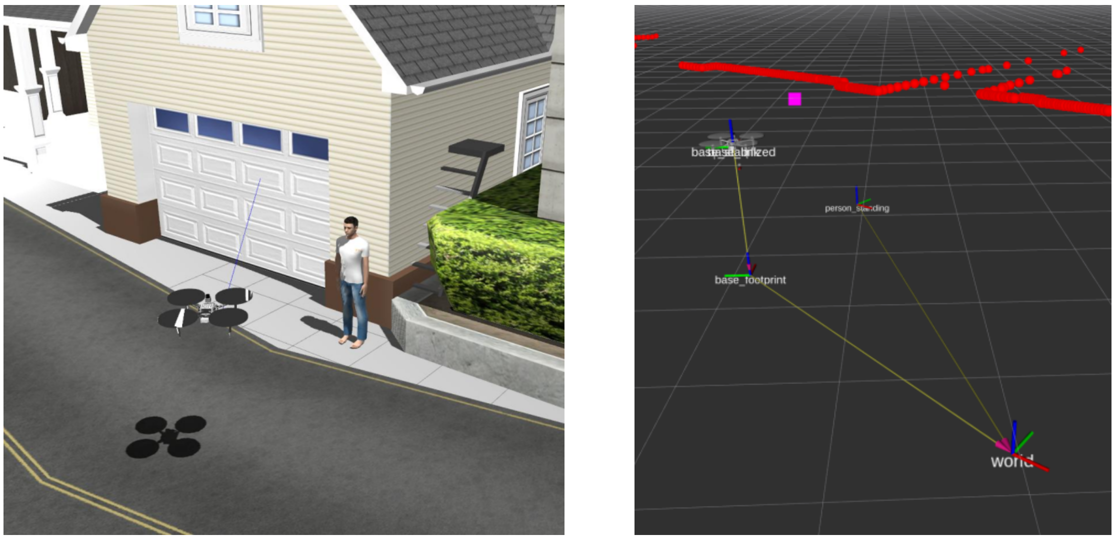


Figure 4.23: Gazebo environment and the corresponding coordinate system relations.

the object origin point, and thus a rectangle around the object is formed. Henceforth, by transforming the 3D points in respect of the camera's frame, and by using undistortion and perspective transformation, we can extract the four coordinates projections on the image plane. Figures in 4.24 depict the projected four points and the corresponding annotated images.

In this way, the detector will learn to localize humans and extract bigger areas, which will be correlated with the thermal-positive regions, as captured from the thermography camera. Consequently, we generated imagery datasets by flying above the human objects, with the UAV system, under various Gazebo environmental and lighting conditions.

Meanwhile, we create a $.json$ file, along with the images, which contains the ground-truth pixel coordinates from each of the four coordinates. The $.json$ file has the following format,

$$\begin{aligned} \Big\{ \text{“IMG\_XXX0.JPG”} &: [x^0_{lu}, y^0_{lu}, x^0_{ru}, y^0_{ru}, x^0_{rd}, y^0_{rd}, x^0_{ld}, y^0_{ld}], \\ \text{“IMG\_XXX1.JPG”} &: [x^1_{lu}, y^1_{lu}, x^1_{ru}, y^1_{ru}, x^1_{rd}, y^1_{rd}, x^1_{ld}, y^1_{ld}], \\ &\ldots \\ \text{“IMG\_XXXX.JPG”} &: [x^X_{lu}, y^X_{lu}, x^X_{ru}, y^X_{ru}, x^X_{rd}, y^X_{rd}, x^X_{ld}, y^X_{ld}] \Big\} \end{aligned}$$

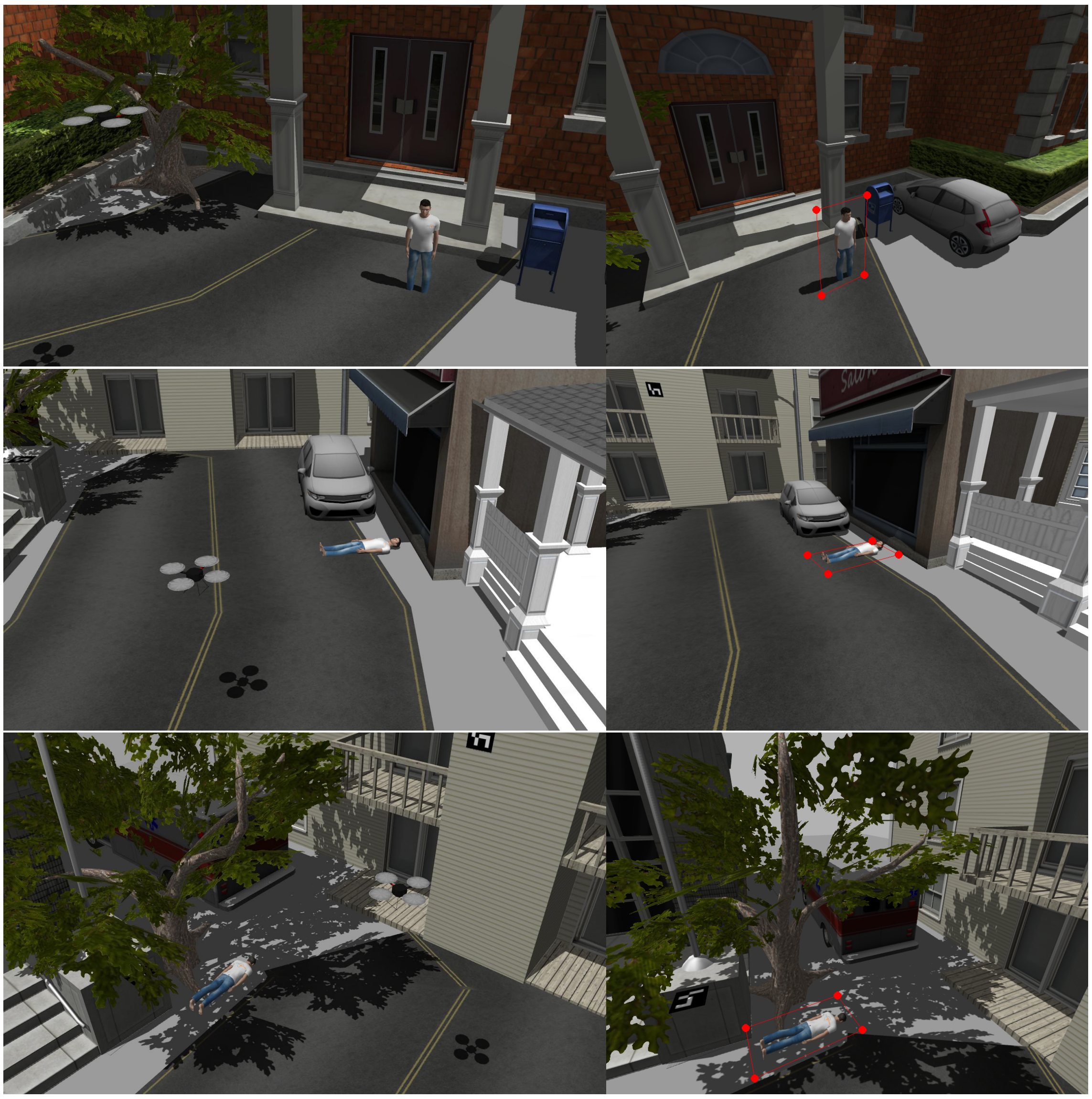


Figure 4.24: Moments during the dataset creation. The images of the right column, depict the positive areas of the current captures.

which has the picture name and its four x, y positive region coordinates, starting from the $left-up$ corner to the $left-down$ corner, in clockwise direction. Also, in case that not all corners points can be projected simultaneously on the image frame (they are out of current field-of-view), this image is discarded from the creating dataset. Figures 4.25 and 4.26 illustrate captures of a person model, by the UAV's camera within the Gazebo environment, along with the generated label images.

Thenceforth, similarly, with the case of the real-imagery COCO dataset, we created a custom Gazebo dataset and we trained the neural network on virtual images. Throughout these procedures, we evaluate the different behaviors of the concluded detectors by comparing their accuracy results and underscoring important remarks about their ability in human detection in the Gazebo environment conditions. It is noteworthy, this process can be applied in any other object of the Gazebo environment, by inserting the corresponding object's coordinate frame in the dedicated node, along with the four 3D framing points. In this way, we can create multiple datasets of various objects, that exist in the Gazebo simulator.

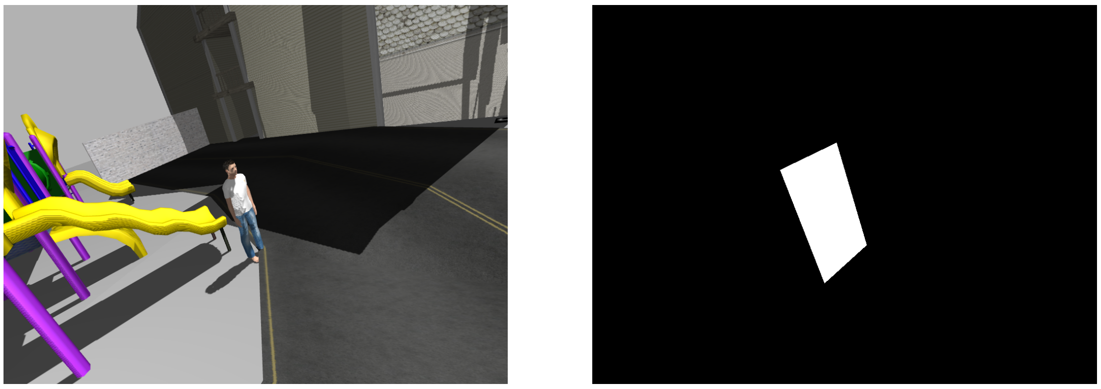


Figure 4.25: The captured RGB image and the auto-generated annotation image.

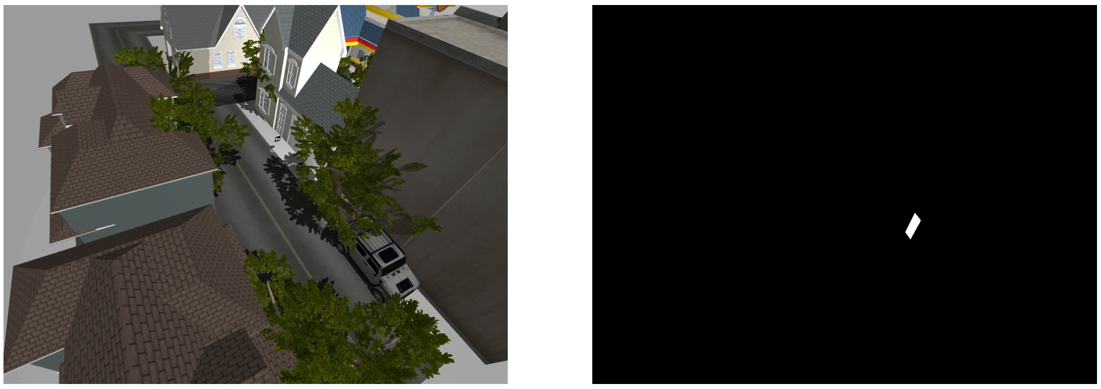
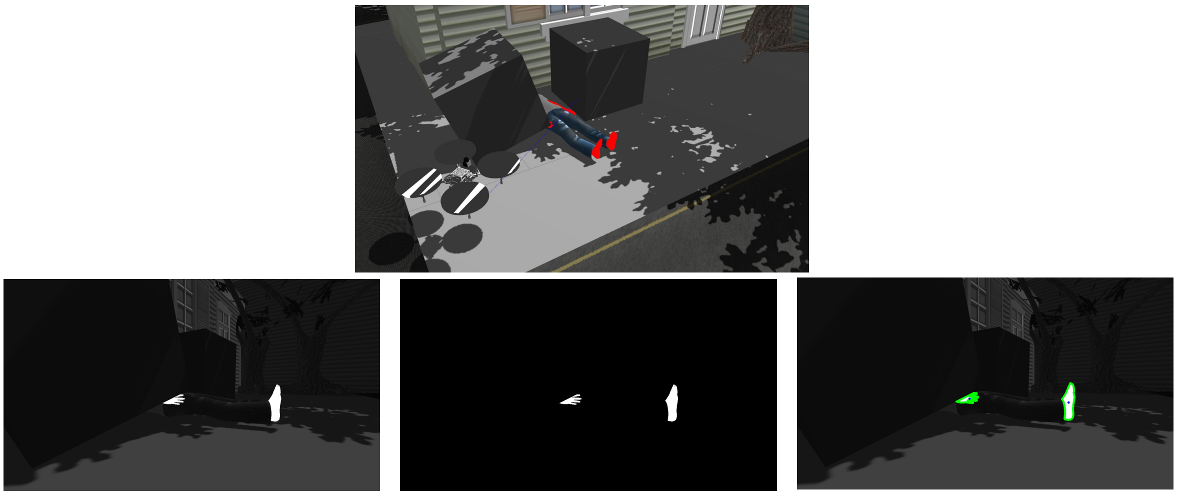


Figure 4.26: Distant image with its corresponding label image.

4.3.2 Target Recognition through Thermography Data

First, as mentioned in Section 4.1.3.3, the thermography camera captures single-channel imagery information that contains the temperatures of the viewing scene. Hence, we initially filter out lower temperatures by applying a fixed binary threshold and thus extract the hotter areas of the thermal image. Having that, the contour regions of the thermal-positive areas are calculated by using the topological structural analysis algorithm of S. Suzuki *et al.* [37] of OpenCV library. In specific, the outer border following approach is utilized to capture the extreme outer contour coordinates of the appearing regions, as they offer adequate information about the localization of the area. Figures in 4.27 show the raw captured footage of the thermography camera and the outcome of the outer contour extraction algorithm.



(a) Input thermography image.

(b) Binary thresholded image.

(c) Extracted thermal regions with their moment coordinates.

Figure 4.27: Thermal-Target detection.

By having the positive areas, we extract their moment coordinates which indicate their center of mass projected on the image plane. Finally, these regions are published through a specific ROS topic, to be furtherly processed by the rest of UAV behavior nodes. These areas are published in respect of the *PositiveAreas* ROS message structure, similarly to the optical-positive areas, as described in Section 4.3.1.

4.3.3 Fusion of Optical and Thermal-Positive Areas

As the detection results are published independently from their corresponding nodes, these data can be combined by correlating them with respect to their captured timestamp. As mentioned before, both detectors produce 24 fps approximately, which are more than sufficient to cross-validate and perform data fusion. However, in the case of having an onboard computer that has lower specifications, the correlation procedure can be simply modified to adapt to the new and decreased publishing rates.

First, since both detectors publish positive areas with respect to *PositiveAreas* ROS message structure, we spatially compare their occupied areas to locate possible matches and overlaps. As long as there are common references, this implies that a living individual is depicted, and thus it can be marked as a possible target for further examination. In the case of having a confident and stable prediction of a human from the optical detector, despite the lack of temperature information from the thermal detector, we also include these situations as possible targets. Therefore, as the positive areas are approximated by the combination of both aforementioned detectors, a fused image of possible individuals depiction is generated.

In the meantime, the resulted predictions may include outliers and deformations in their areas' shape generation, which will aggravate the final area approximation. Figures 4.28(a-b) show moments during the UAV's target detection and the approximated positive areas. To suppress such occurrences we apply convex hull methods to obtain optimal convex areas. In specific, we utilized the J. Sklansky approach [38], in finding the convex hull of each generated polygon shape. Thenceforth, by employing the OpenCV topological structural analysis module [39], we obtain the contour regions of the given thresholded binary image. As it can be seen from Figures' column 4.28(c), the positive areas contain discontinuities that form smaller-sized positive area, that can be considered as distinct targets. Nonetheless, by applying the convex hull region formulation and by forming a rectangle to cover the existing black areas with respect to the occurred convex areas, the nearby areas are grouped and form a united area, and the majority of the outliers are mitigated. Thence, by extracting the convex set plane points, we calculate the generated polygon moments which will describe the center points of the depicted targets. Figures 4.28(d) illustrate the concluded detected targets.

Thus, these coordinates are posted and published as the projected target coordinates, which will be used by the gimbal system to perform the aiming procedure. However, in

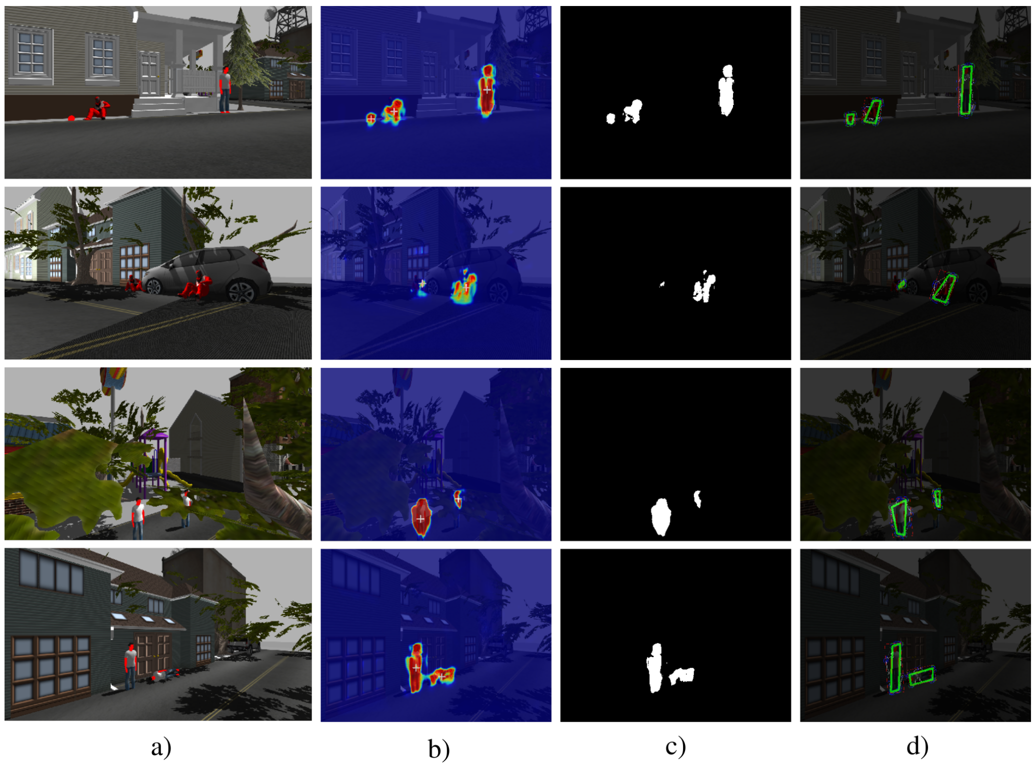


Figure 4.28: The convex-hull procedure applied on target-positive areas. a) Input image. b) The probability map of the located targets. c) Binary representation of the detection, after adaptive thresholding. d) Convex hull post processed detections.

case of more than one viewing target, the UAV follows a prioritization behavior depending on the scenario's current status.

4.3.4 Gimbal Scanning Behavior and Aiming

Meanwhile, to achieve sufficient area scanning, the UAV system is enhanced by mounting a three-dimensional gimbal underneath its body frame. In specific, this gimbal system will be equipped with the cameras and the rangefinder sensor, in order to be utilized for the targets' detection and localization procedures. In this way, this system can be independent of the total UAV movement, as it can easily rotate, scan the surrounding environment and engage on a possible target, while the drone system will execute its navigation trajectory. Hence, the gimbal behavior has been enhanced to support gimbal commands and perform specific scanning and engaging movements, depending on the current target detection status.

Particularly, the developed gimbal unit contains inner joints and links which can receive forces, and achieve gimbal movements and pose corrections. In our implementation, we developed a gimbal controller which awaits for the communication bridge of the ROS environment and the Gazebo simulator and then supports simulated force and torque commands broadcasting to the UAV's gimbal system unit. This system captures the image plane targets, namely coordinates, that are published by the fusion of both optical and thermal detectors, and by integrating a software Proportional-Integral-Derivative (PID) module, achieves smooth and fast aiming on the desired target, and engages until the next target is acquired.

Figures in 4.29 and 4.30 depict gimbal engaging tests on an ArUco [40, 41] marker, which resembled our target, in order to evaluate the smoothness and efficiency of the UAV's gimbal control. In specific, the main purpose of the gimbal aiming behavior is to orientate and engage accordingly, and aim the rangefinder measuring ray on the target's projected body coordinates. However, since the cameras' positions differ from the rangefinder measuring point ($7cm$ below their optical frames origin), the gimbal system aims given a slightly translated crosshair, to make the rangefinder be on target. The software implementation of the PID logic that satisfies following formulation,

$$u(t) = K_p * e(t) + K_i \int_0^t e(t')dt' + K_d \frac{de(t)}{dt}$$

This software implementation was fine-tuned until the gimbal performed sufficient and fast aiming commands. The three control terms of the PID controller, depending on the UAV model mass and Gazebo environment gravitational forces, were set as the $K_p = 11$, $K_i = 1$, and $K_d = 50$, for the proportional, the integral, and the derivative influence, respectively. Specifically, we used a small value for the integral parameter to reduce model overshooting and settling time, and a bigger value for the derivative tuning parameter to increase the latter. Also, we chose $dt = 0.1$ as the time sampling parameter, which was adequate in terms of speed aiming.

Figures in 4.29 show sequential moments of marker detection and gimbal-unit aiming, during a UAV's flight in the Gazebo world. Figure 4.30 depict a more distant aiming procedure of the simulated UAV system. In this way, by providing coordinates of the viewed-detected target, the onboard gimbal unit is able to perform appropriate movements, and thus engage on target without being affected by the UAV's body movement.

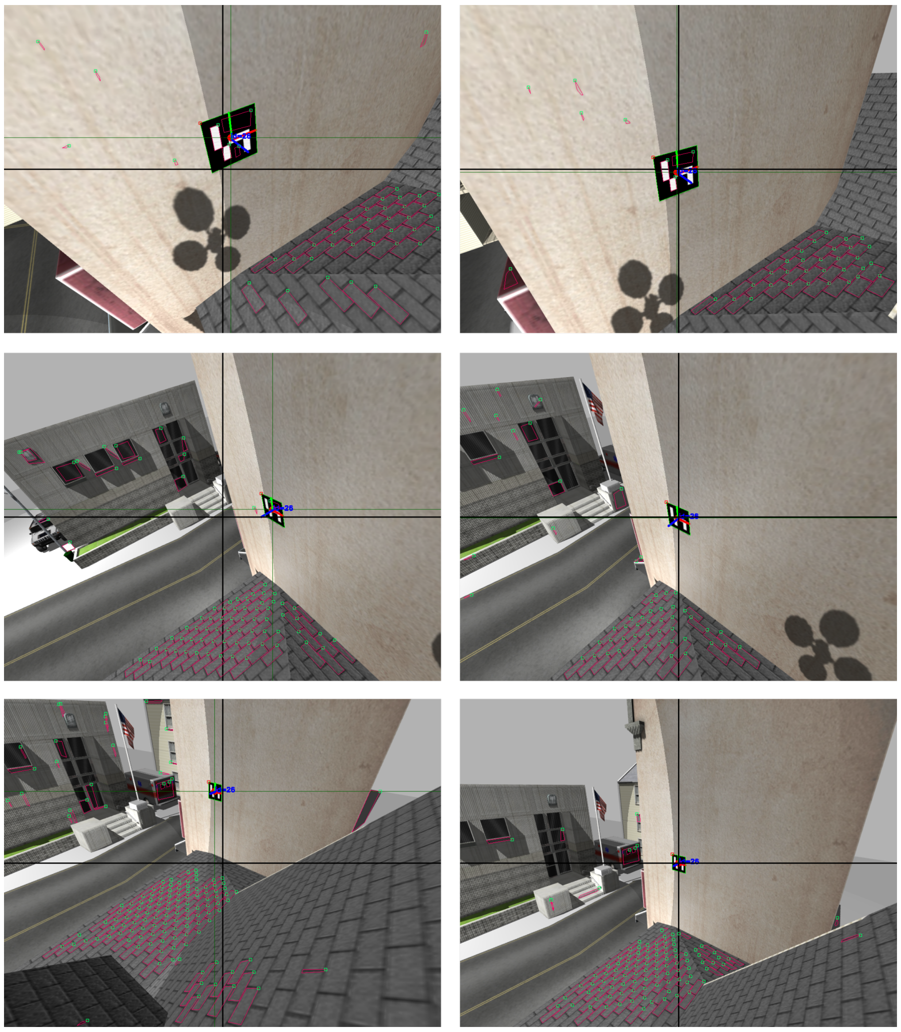


Figure 4.29: Gimbal-unit engage on located target, during simulated UAV's flight.

4.4 Search-and-Rescue UAV Behavior

In this section, UAV behavior characteristics will be described which are essential for its Search-and-Rescue robust operation in the unknown environment. In particular, notable UAV behavior features will be reported, in terms of creating the SAR map, prioritizing and positioning detected targets, and finally deciding about unknown areas exploration, which completes the total autonomous behavior of the rescuing UAV robot.

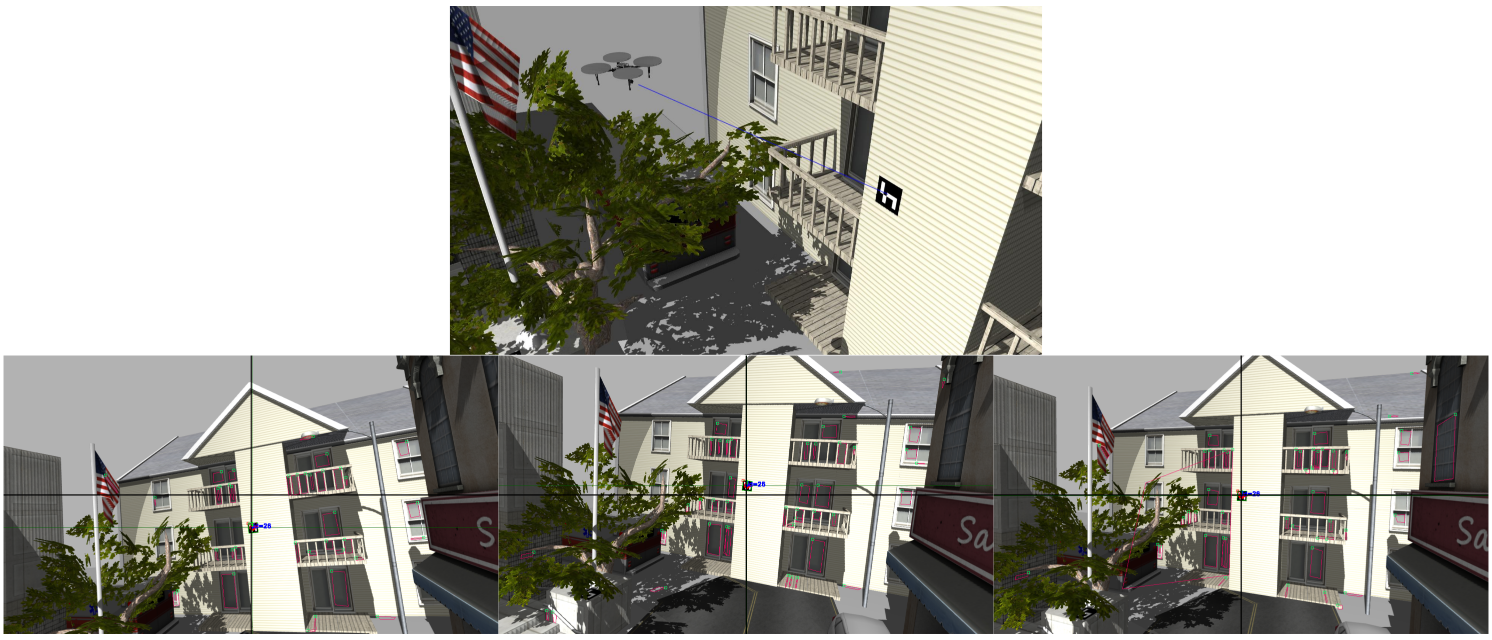


Figure 4.30: Gimbal aiming on distant target, based on optical frame center point.

4.4.1 2D Simultaneous Localisation and Mapping

Initially, to ensure sufficient area coverage and provide a local map with geo-referenced objects and areas of interest for the first-responders, the UAV performs simultaneous localization and mapping procedures during its flight in the unknown environment.

In crisis scenarios, rescuers, and more importantly the UAV system, can not fully rely on available area maps as there may be significant spatial variations due to the occurred incident in the area. Hence, the UAV needs to construct the map of its surrounding environment and also localize itself in it, to adapt to environment changes and focus on unexplored areas without any human intervention. At the same time, every detected target should be positioned and pinned with respect to this map, and so the rescue team will have details about the operating area and will be able to plan their rescuing approach, accordingly.

For this reason, the Hector SLAM system [42] is utilized, which provides full 6DoF robot pose estimation by solving the Online SLAM problem. The Hector system combines two subsystems, a 2D SLAM system based on the integration of laser scans in a planar map and an integrated 3D navigation system based on inertial measurement unit, to achieve precise localization and mapping. During the navigation, the captured distance ranges are expressed in the LiDAR's frame the $\{hokuyo\_frame\}$. The measurements are spatially transformed through quadcopter pose estimation and the tf tree information, into the quadcopter's stabilized base frame coordinate system, namely the $\{base\_stabilized\}$ frame, to be expressed relative to the body frame.

The Hector SLAM uses an occupancy grid map [43], to store the belief about the current environment form. Due to the occupancy grid map discrete form, sub-grid cell approximation can not be performed, as there is a limitation to the map resolution parameter. For this reason, an occupancy value and gradient approximation are applied by using the information from the four closest point positions, to achieve accurate estimations of occupancy probabilities and derivatives. Figures in 4.31 show moments during a SLAM procedure of the UAV in the Gazebo environment. Additionally, as mentioned previously in Section 4.2.3, the UAV navigates autonomously by keeping a stable altitude level, and thus the resulted 2D map and the searching procedure is done in reference to this.

Meanwhile, each time the mapping procedure is reset, the new map is initialization is based on the current UAV global position and orientation. Therefore, for each point on the planar map, we instantly extract its global coordinates which are described in the $\{world\}$ frame. Figures in 4.32 show two separate $\{map\}$ frame initializations relative to the corresponding UAV global pose (described in $\{world\}$ frame).

4.4.2 Target Identification and Spatial Correlation

One of the most essential key features of the UAV search behavior is its ability to determine if a viewed target has been already mapped or not, to proceed with the gimbal engagement, make distance measurements and thus localize it. For this reason, it is vital to locally store information each time a target is positioned by referencing it on the generated map, to avoid duplicated inspections and measurements, and thus make the UAV system to focus on unpositioned and unlogged detected targets.

Henceforth, as the UAV system has the information of previously measured targets, it utilizes their spatial information in relation with its corresponding body and gimbal frame pose and determines if any new targets are appearing in its current viewing field. The structure of the information, which is stored each time a target is registered in the UAV system, is presented in Figure 4.33. Particularly, this struct contains variables which indicate the global position of the target in the environment field, the category of

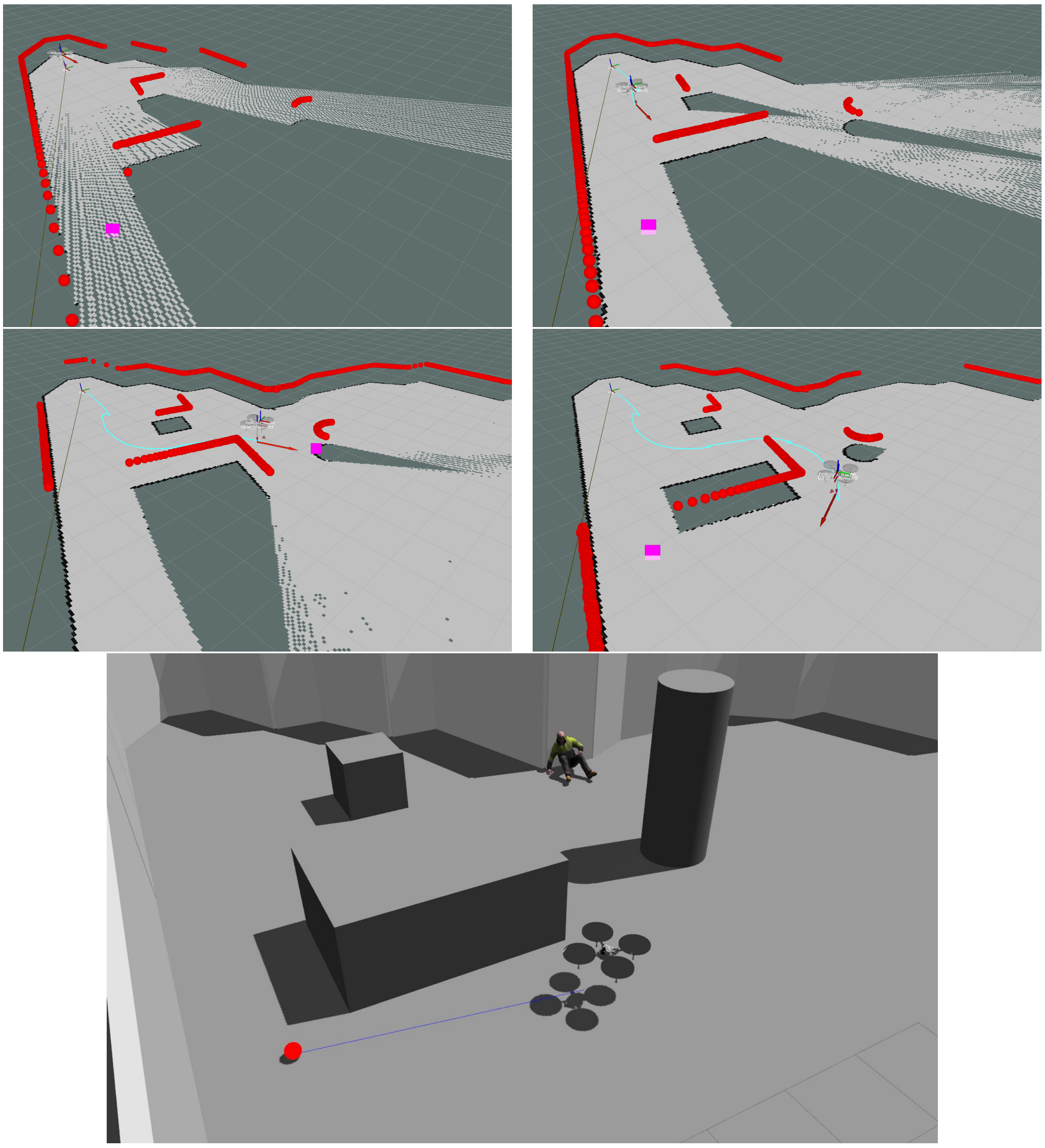


Figure 4.31: Moments of SLAM procedure during the UAV flight in unknown Gazebo environment.

the target (human, object, etc), and an identification number.

Additionally, since the UAV holds information about the targets global three-dimensional

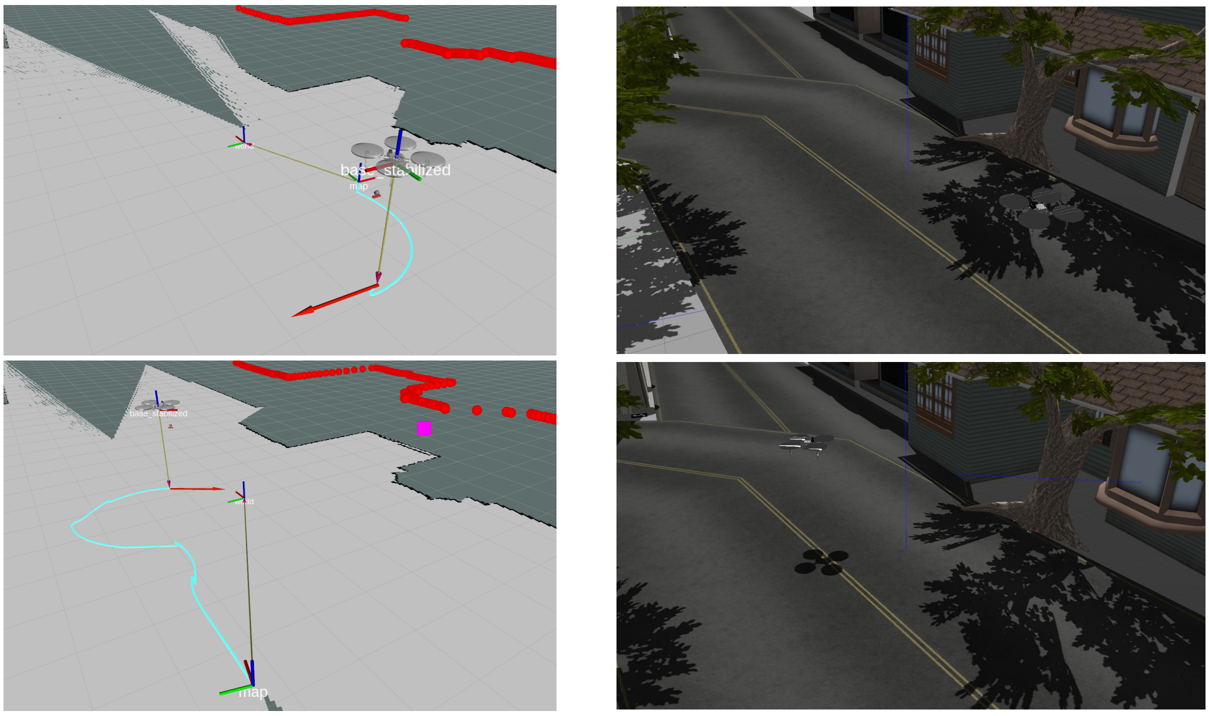


Figure 4.32: Map initializations according to the corresponding UAV global pose and in respect of the $\{world\}$ frame.

position it can spatially describe them in relation to its body frame, namely the $\{base\_link\}$. Moreover, since there is a transformation connection between the $\{base\_link\}$ frame and the onboard systems, and more importantly towards the embedded gimbal sensors, the targets can be spatially described in respect of the cameras' coordinate systems. The Equation 4.15 shows the transformation of the target position P_{target} in respect of the UAV's color camera frame.

$$^{camera\_optical\_frame}P_{target} = {}^{camera\_optical\_frame}_{gnss\_receiver}T \quad \cdot \quad {}^{gnss\_receiver}P_{target} \tag{4.15}$$

Also,

$$^{cof}_{gr}T = {}^{cof}_{cf}T \cdot {}^{cf}_{gy}T \cdot {}^{gy}_{bs}T \cdot {}^{bs}_{gr}T \tag{4.16}$$

where, cof=$\{camera\_optical\_frame\}$, cf=$\{camera\_frame\}$, gy=$\{gimbal\_yaw\}$, bs=$\{base\_link\}$ and the gr=$\{gnss\_receiver\}$.

Algorithm 2: UAV Behavior on Target Identification, Prioritization and Enganging

```
load localized_target_areas;
define max_pixel_margin, viewing_target_areas, target_id = 0;
while new captured_positive_areas do
    active_areas=[ ];
    for all captured_positive_areas $\mathcal{P}$ do
        find $\mathcal{P}$ center point;
        define ignore_this_area = False;
        for all localized_target_areas $\mathcal{T}$ do
            if $\mathcal{P}$ center point $\in \mathcal{T}$ and isin_FOV$(\mathcal{T})$ then
                ignore_this_area = True;
                break;
        if ignore_this_area then
            continue;
        for all viewing_target_areas $\mathcal{V}$ do
            find closest region/s $\mathcal{V}$ with index/ces $idx$ and distance/s $d$;
        if $d >$ max_pixel_margin or # of matched regions != 1 then
            define $idx$ = target_id++;;
            initialize viewing_target$(\mathcal{P}$.x, $\mathcal{P}$.y, Area, $idx)$ ;
            add viewing_target in $\mathcal{V}$;
            active_areas.push_back($idx$)
        else
            $idx = idx$ of matched region;
            save old 2D position ($\mathcal{V}$.x, $\mathcal{V}$.y);
            update $\mathcal{V}$ $($ $\mathcal{P}$.x, $\mathcal{P}$.y, Area, $idx)$;
            active_areas.push_back($idx$)
    erase_inactivated_areas $(\mathcal{V}$, active_areas$)$;
    visualize_areas_of_interest_2DImagePlane$(\mathcal{V})$;
    find minimum id from $\mathcal{V}$, namely current_target_id;
    if # of ignored_areas = # of captured_positive_areas or current_target_id=-1 then
        No new target detected.
        publish Empty Target; publish(enable_exploration= $True$);
    else
        publish $\mathcal{V}$(current_target_id); publish(enable_exploration= $False$);
```

```
std_msgs/Header    header
    uint32    seq
    time    stamp
    string    frame_id
geometry_msgs/Point    position_in_map
    float64    x, y, z
std_msgs/Int64    id
std_msgs/Int64    category
std_msgs/Bool    isinUAVview
geometry_msgs/Point[8]    cube_area_cameraplane
    float64    x, y, z
geometry_msgs/Point    circle_center_cameraplane
    float64    x, y, z
geometry_msgs/Point    circle_radius_cameraplane
    float64    x, y, z
```

Figure 4.33: The registered target description fields.

Since both cameras are calibrated and the spatial relation of their optical frames is known relative to their body frames, every object that is described in respect of the latter frames in three dimensions can be projected on to their image plane surfaces. Hence, every registered target can be projected in real-time on to the cameras' image plane, and thus they can be correlated with the captured positive areas and determined if they are excluded from the running detection. The two-dimensional pixel position of the given targets can be acquired by using a perspective transformation, by utilizing the intrinsic and distortion camera matrices.

Specifically, assuming that we have $P_{target} = (X, Y, Z)$ described in $\{camera\_optical\_frame\}$ frame and the transformation,

$$ {}^{cof}_{cf}T = \left[\begin{array}{ccc|c} \multicolumn{3}{c|}{{}^{cof}_{cf}R} & {}^{cof}P_{cf\,ORG} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{4.17} $$

is known, it is implied that,

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = {}^{cof}_{cf}R \cdot P_{target} + {}^{cof}P_{cf_{ORG}} \tag{4.18}$$

and

$$x' = x/z \text{ and } y' = y/z \tag{4.19}$$

Meanwhile, since the camera lenses contain distortion, it is mandatory to calculate the undistorted coordinates,

$$x'' = x' \cdot \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1 x' y' + p_2(r^2 + 2x'^2) \tag{4.20}$$

and

$$y'' = y' \cdot \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_1(r^2 + 2y'^2) + 2p_2 x' y' \tag{4.21}$$

where, $r^2 = x'^2 + y'^2$. The parameters k_1, k_2, k_3, k_4, k_5 and p_1, p_2 are the distortion coefficients, extracted from the distortion matrix $\mathcal{D}$. Thus, by using the camera's intrinsics parameter matrix $\mathcal{K}$, the undistorted projection of the three-dimensional point P_{target} in the image plane, is given by the coordinates,

$$u = f_x \cdot x'' + c_x \text{ and } v = f_y \cdot y'' + c_y \tag{4.22}$$

The point (u, v) is expressed in the camera's optical frame, depending on the used camera sensor ($\{camera\_optical\_frame\}$ or $\{thermal\_optical\_cam\}$), and expresses the pixel coordinates of the target on the image plane. Additionally, given the two-dimensional projection of the target, we define a cube three-dimensional area that includes the target, in which we also obtain their coordinate projections and store them in the *cube_area_cameraplane*. In this way, we can project the three-dimensional area which is occupied by the target on the image plane. In parallel to that, the variables *circle_center_cameraplane* and *circle_radius_cameraplane* hold the projection points circular imaginary area.

Thenceforth, each time a target is positioned globally, the UAV updates its corresponding viewing plane with the aforementioned areas, and thus excludes prior detected

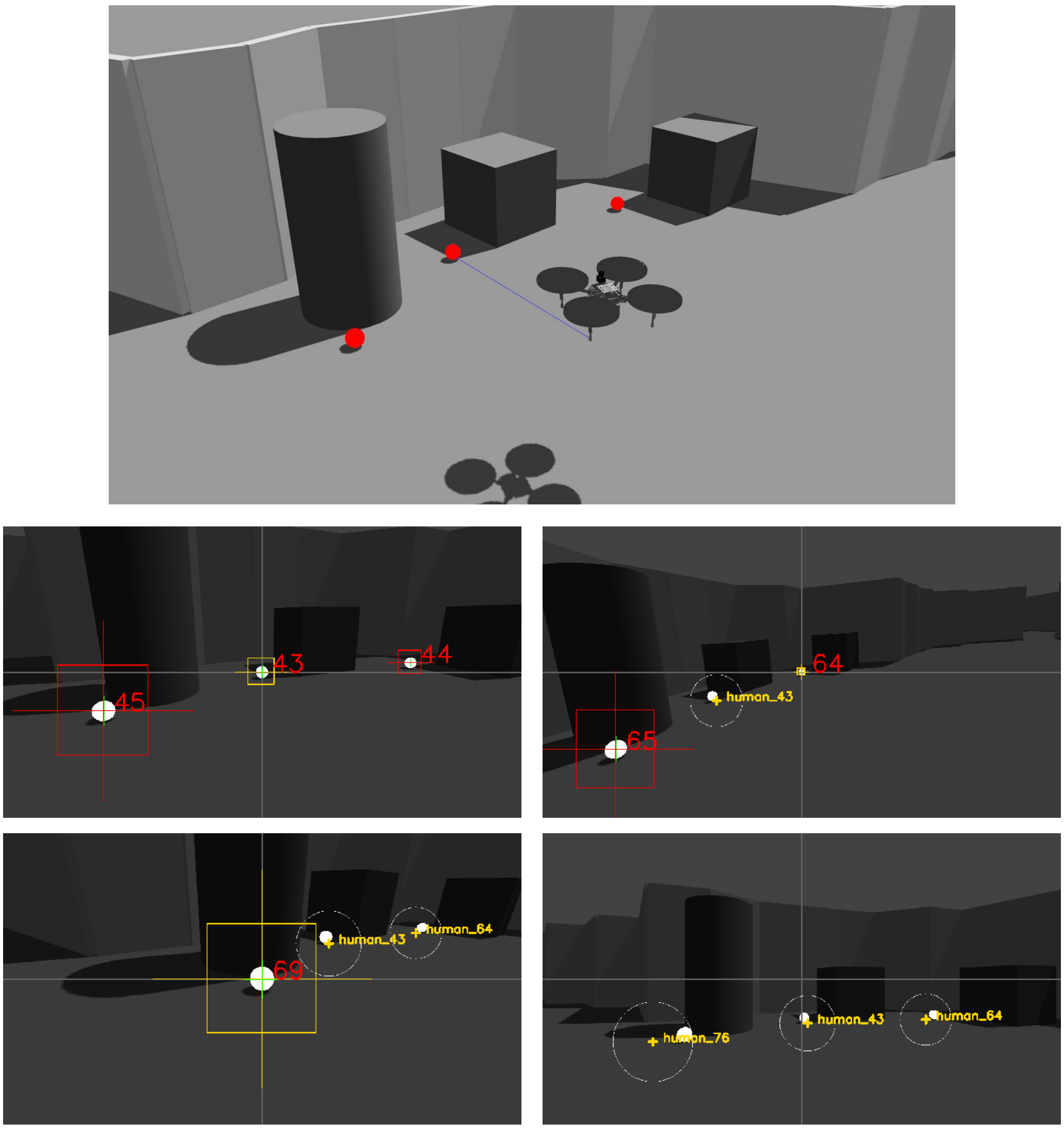


Figure 4.34: Snapshots during the UAV targets' positioning procedure. Each time a target is globally positioned, its sphere ROI is projected on the image plane to prohibit the area scanning.

areas from inspection. Figures in 4.34 show the addition of an appeared thermal target and the inclusion of the ROI after its positioning, in order to be ignored in future measurements.

Meanwhile, as the targets are projected on the cameras' image plane, it is crucial to determine if their position is in or out of the cameras' viewing field. As the perspective transformation can project the three-dimensional target on the image plane, there is no further information if this target is in the front or the back of the camera's heading orientation. For this reason, before the projection of a given target area, we check its relative position in respect to the origin of the camera's body frame. In case that the target's x coordinate value is negative, it denotes that the target is positioned behind the camera's field of view, and thus must not be projected and considered in the current viewing plane. On the other hand, the perspective transformation result will indicate the exact two-dimensional position of the corresponding target. Finally, the result of this examination is stored in the $isinUAVview$ variable field.

4.4.3 Target Prioritization and Gimbal Engagement

Since the UAV system has excluded the positive areas that belong to already mapped target areas, it focuses on the remaining unlogged viewing positive regions. For this reason, a prioritization behavior is implemented to achieve specific gimbal engaging behavior to position the unregistered targets, and thus to include them in the Search-and-Rescue map and the local database.

As the UAV captures imagery data (optical and thermal) in real-time, it has to correlate the corresponding unknown positive areas with the prior detected regions which were captured from previous frames. Specifically, each time there is a new frame with available positive contour areas for positioning, we define separate regions of interest (ROIs), based on their contour moment point coordinates (center of mass) and their occupation size of the image plane surface. In this way, there the image contains distinguished labeled areas, that need to be positioned. Therefore, each time a new frame is captured, the new unidentified regions are correlated with the existing, as long as they belong on their ROIs areas. Figure in 4.35 and 4.36 show a plethora of detected and unpositioned targets, along with their detection/positive ROIs. As it is clear, despite the UAV circular movement the identified areas remain unchangeable, and thus, the UAV can keep its priority and focus on the foremost acquired target.

However, in the case of abrupt UAV movements, there may be notable region translations on the image plane which may cause the new detections to be out of their corre-

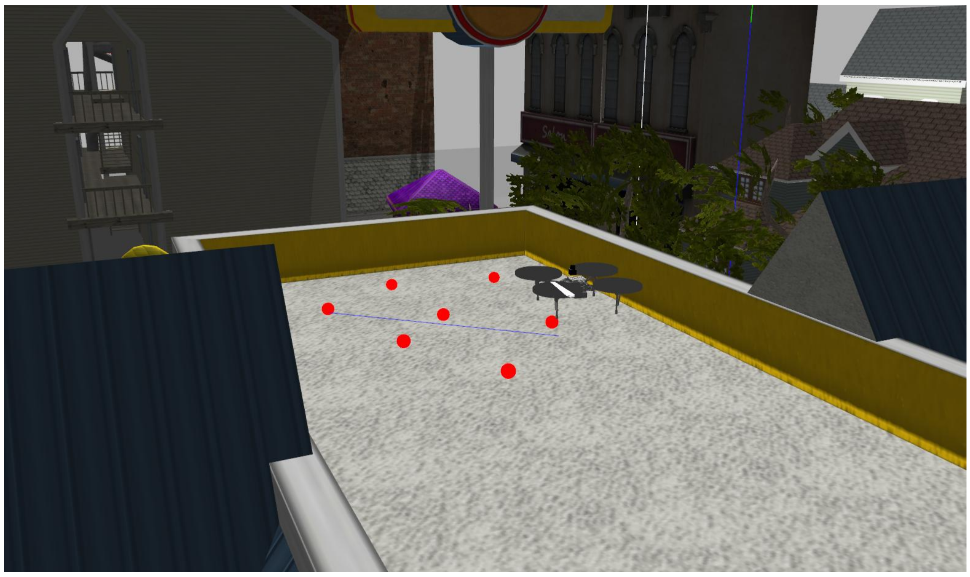


Figure 4.35: Unregistered targets prioritization scenario.

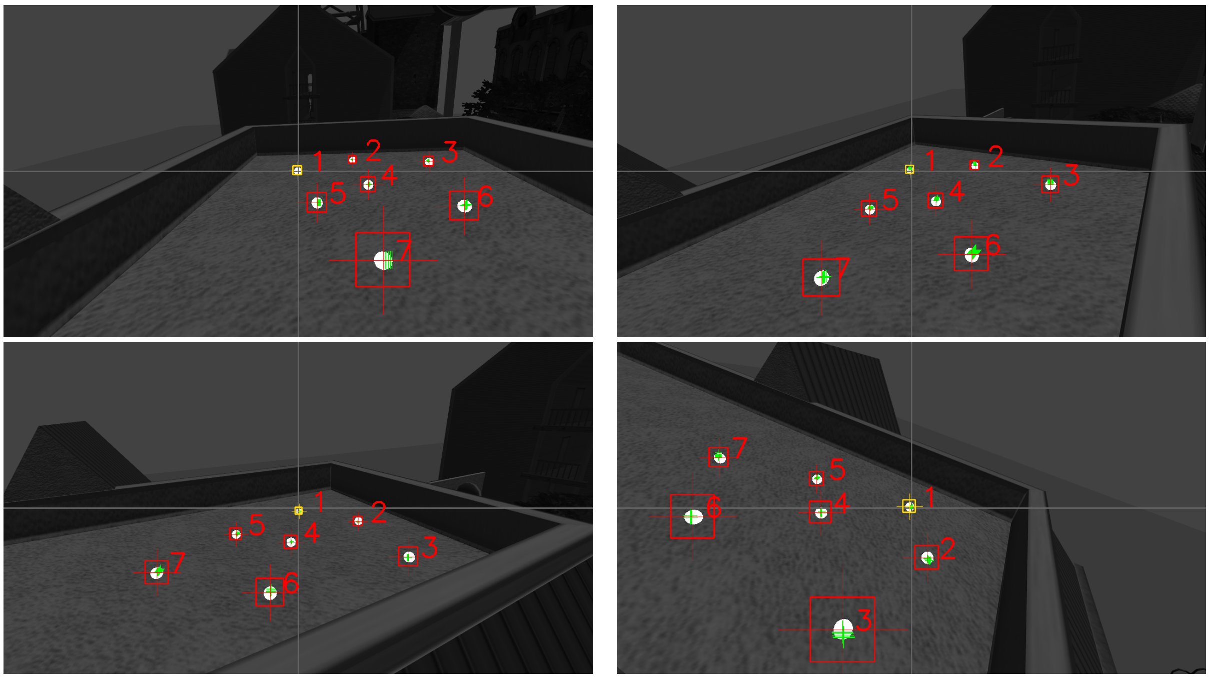


Figure 4.36: Target regions along with their generated ROIs.

sponding ROI sections. In this situation, the region is assigned to the closest group as long as their relative distance is less than the *max_pixel_margin* parameter value. Henceforth, in case of positive area appearance in a new image plane segment, it is considered as a new region for examination, and thus it is registered with a new identification number.

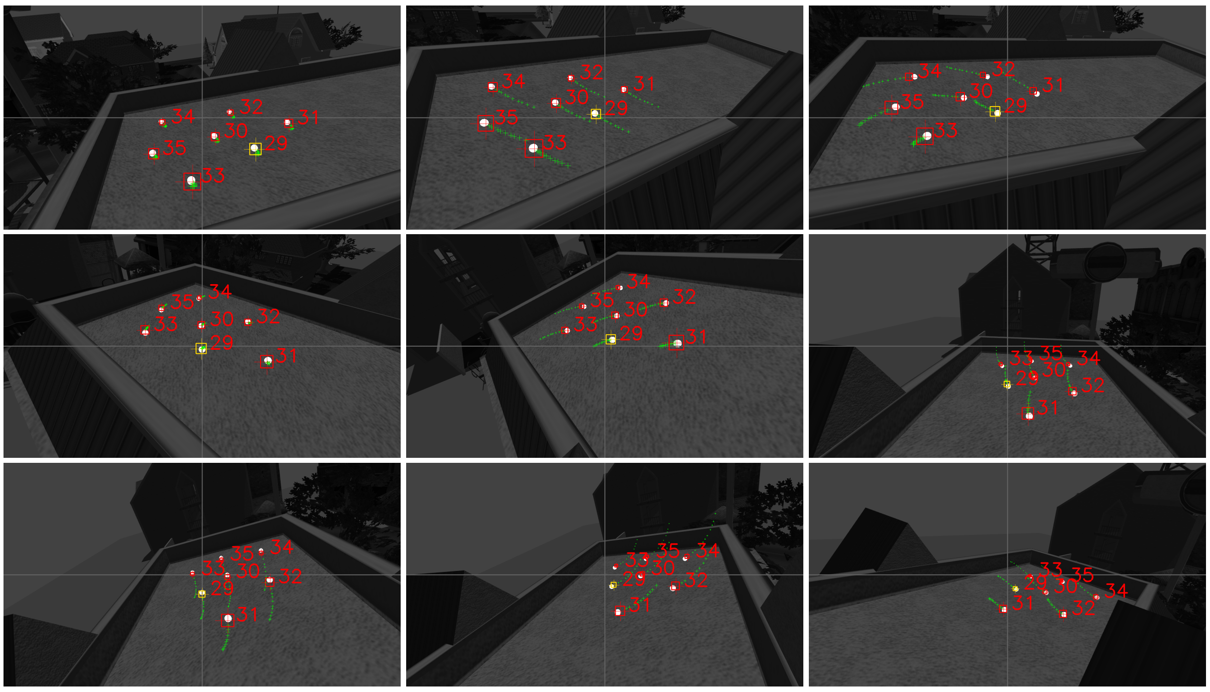


Figure 4.37: Target areas detection during abrupt UAV movements. The green cross markers indicate the previous center point positions of the targets' detection ROIs.

Meanwhile, during the frames update, we keep the preceding 10 two-dimensional positions of the region's center points, to show their trace along with the time progression. Figure 4.37 show the robustness of the area identification and prioritization algorithm in sudden UAV movements. Also, Figure 4.38 illustrates a special occasion, where the UAV performs a low flight above an area that has plenty of unpositioned targets.

As long as there are available target regions for detection, the UAV picks the smaller identification numbered target, and publishes its (x, y) camera plane coordinates for the gimbal aiming node. Throughout the gimbal aiming and engagement procedure on the selected target, the rangefinder captures a predefined number of distance measurements, while it orientates accordingly to keep the target in aiming position. Whenever the target positioning is finished, the UAV updates its localized targets database, pins the positioned target in the constructed map visualization, and proceeds with the successor

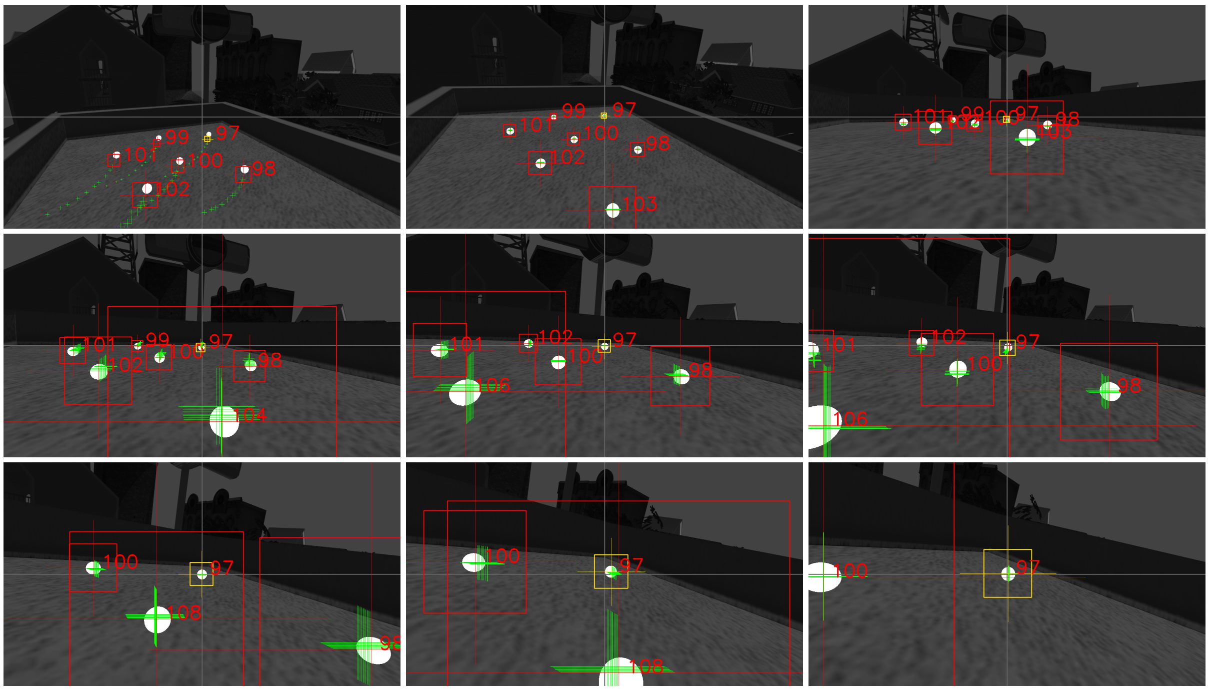


Figure 4.38: Gimbal engagement on yellow target 97. The gimbal aiming and engaging behavior remains unaffected even with a close UAV flight above from an area that is crowded with available and unlocalized targets.

target region aiming and mapping procedures (if there is any unpositioned target) or proceeds with further room exploration to search for new targets.

4.4.4 Target Global Positioning

Given our current simulated world and the UAV's sensors setup, we had to develop an approach for global positioning the appeared targets in the environment area. In particular, the positioning procedure will be performed by utilizing the captured ranging measurements of the embedded rangefinder sensor, while the gimbal unit is engaged on the selected recognized target. However, even though we use the gimbal yaw information, relative to the UAV heading orientation, to project the targets on the UAV's viewing plane, we exclude it during the target positioning approach. In specific, in this way we can not directly spatial determine the detected target's position relative to the UAV body, and thus, we had to approximate its value through a different perspective. Hence, we approach this problem by utilizing captured distances from the located target and the corresponding UAV's global positions, to create an overdetermined system, that has the target's position as the solution. For this reason, we propose a least-squares multilateration approach[1] to solve the non-linear overdetermined system, which is formulated by the UAV's captured global positions and the georeferenced distance measurements from the targets, and to conclude with the approximated target's global 3D position.

During the flight, the UAV system can capture multiple target distance measurements with its onboard rangefinder sensor unit. Nonetheless, those range measurements are taken with respect to the rangefinder's coordinate system, namely the $\{rangefinder\_frame\}$, and they are not directly related to the onboard GNSS receiver frame. So, the exact time that a single measurement is captured, the UAV system has to spatially describe and transform it to the global onboard frame, namely the $\{gnss\_frame\}$, and thus extract the target distance from the GNSS receiver module, according to the corresponding inner-frames' transformation relations.

However, we assume that there is no at least accurate information about the relative heading of the UAV body and the attached gimbal frame, so there is no direct transformation between the $\{rangefinder\_frame\}$ and the $\{gnss\_frame\}$. As described in Section 4.1.1, the rangefinder sensor is rigidly mounted on the gimbal horizontal axis, and so, it is statically described with respect to the gimbal's $\{gimbal\_pitch\}$ frame. So, each captured distance can be re-calculated with respect to the $\{gimbal\_pitch\}$ coordinate system, due to their direct static spatial relation. Also, the $\{gimbal\_pitch\}$ frame position is known, relatively to the $\{gimbal\_yaw\}$ frame, except their relative yaw orientation. For this purpose, we assume that their relative angle difference is zero, as, during the gimbal yaw movement, the onboard distance sensors are rotating around the gimbal yaw axis, which is the z-axis of the $\{gimbal\_yaw\}$ frame. Given that, the corresponding angle between the UAV body and the gimbal frame does not affect the captured the distance measurements, and so they can be spatially described to the rest of the frames that are located on the UAV's body.

Thus, the distance measurement is described in the UAV's gimbal mounting point, which is also statically placed relative to its onboard GNSS receiver frame. Hence, by obtaining the transformation matrix ${}^{\{gnss\_receiver\}}_{\{rangefinder\_frame\}}T$, which assumes zeroed relative

[1]This research has been partially supported by the European Union and Greek national funds 577 through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH 578 - CREATE - INNOVATE: T1EDK-03209.

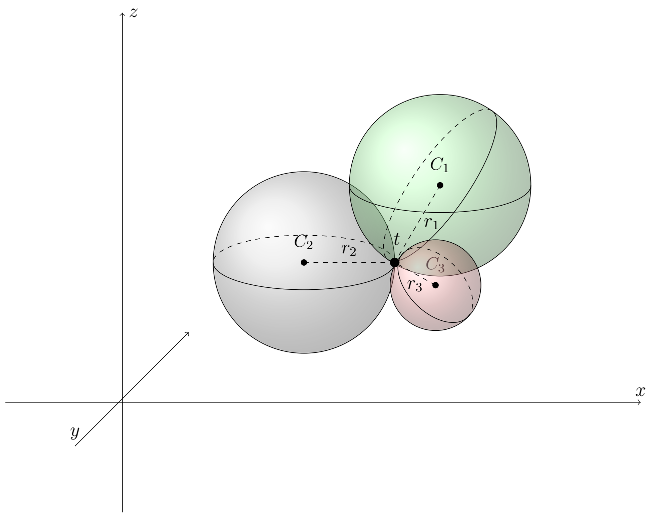


Figure 4.39: Trilateration method on t point positioning. Given the three sphere centers c_i with their radius lengths r_i, we can approximate their intersection point t, in the 3D space. The sphere centers represent UAV system positions in 3D space and the radius lengths are the corresponding estimated ranges from the target marker.

heading between the UAV body and the gimbal frame, the distance range of the viewing target and the global positioning receiver module are calculated. Given that, since we have constant access on the frames transformation matrices and status as they are updated with each gimbal movement at high and fixed rate, we can spatially describe every distance measurement to the $\{gnss\_receiver\}$ coordinate system. So, given a measured 3D point described in the ranging sensor frame it can be globally described as a distance from the GNSS receiver by using,

$$^{gnss\_receiver}P_{target} = {}^{gnss\_receiver}_{rangefinder\_frame}T \cdot {}^{rangefinder\_frame}P_{target}$$

where,

$$^{gnss\_receiver}_{rangefinder\_frame}T = \left[\begin{array}{ccc|c} & {}^{gnss\_receiver}_{rangefinder\_frame}R & & {}^{gnss\_receiver}P_{rangefinder\_frame_{ORG}} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

Hence, the target distance in respect of the GNSS receiver frame is given by the Euclidean distance between the transformed target's center point and the GNSS frame's origin point. This distance measurement is further stamped with the corresponding measuring time and stored along with the most recent captured UAV's global position.

4.4.4.1 Multilateration Positioning Approach

For the target positioning approach, we propose a true range multilateration method, applied on the UAV's captured spatial data. Through this method, the target xyz position can be estimated by solving a spheres' intersection system, that is formed by the estimated ranges and UAV positioning data.

According to this approach, three (trilateration) or more (multilateration) ranges with their corresponding positions, are required to solve the spheres' intersection equation system. However, in contrast with the example demonstrated in Figure 4.39 which has a unique intersection point solution, in real-world scenarios the captured spatial measurements include uncertainty due to noisy measurements and signal interference. For this reason, the specific system can not be solved in closed-form, and thus we use a non-linear least-squares optimization approach to estimate the target's optimal position.

4.4.4.2 Non-linear Least Squares Approach on Position Approximation

Since our goal is to approximate the x, y, and z coordinates of the detected target, we used the non-linear least-squares estimator on the overdetermined spheres equation system, which is formed by the captured ranging measurements and the UAV global positions. Firstly, the form of the measurement equations appears in (4.23), as it defines a sphere equation with center point (x_0, y_0, z_0) and radius r_0. Specifically,

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r_0^2 \tag{4.23}$$

where the point (x_0, y_0, z_0) is the corresponding UAV's position (center of the sphere) with the measured range r_0 (radius of the sphere). Thus, we define the residuals function as follows,

$$f(x, y, z, \boldsymbol{\beta}) = (\beta_0 - x)^2 + (\beta_1 - y)^2 + (\beta_2 - z)^2 \tag{4.24}$$

with $\boldsymbol{\beta}$ representing the vector of the adjustable parameters.

Although, since there is no closed-form solution, the parameters $\boldsymbol{\beta}$ are refined iteratively, by approximating,

$${\beta_j}^{k+1} = {\beta_j}^{k} + \Delta\beta_j \tag{4.25}$$

where the superscript k is the iteration number and the $\Delta\beta_j$ the vector of increments. This procedure requires also, an initial guess of the final parameter values, and thus we use the zero vector $\vec{v} = (0, 0, 0)$ in all our experiments. The non-linear least-squares problem is solved with the Gauss-Newton approach, which approximates the optimal position values by minimizing the sum,

$$S = \sum_{i=1}^{n} \Big(r_i^2 - f^k(x_i, y_i, z_i, \boldsymbol{\beta}) \Big)^2 \tag{4.26}$$

where n is the number of the given measurements - sphere equations and the $f^k(x_i, y_i, z_i, \boldsymbol{\beta})$ the linearized approximation of the residuals model function (4.24), at k iteration.

4.4.4.3 Measurement Restrictions and Spatial Criteria Algorithm

The multilateration approach approximates a target positioning problem by solving the intersection problem of a set of spheres, that are created by the UAV ranging data and its body global positions. However, the spheres intersection approach may lead to infinite solutions, depending on the type of the UAV flight path. For this reason, we developed a criteria software module, that filters out any newly taken measurement that may expand the equation system solution space or may lead to infinite solutions, and thus adversely affect least-squares convergence.

Initially, assuming that we have taken the first two measurements, their positions define a three-dimensional line, as they form two spheres that have a circular intersection plane. So, as Figure 4.40 illustrates, after getting the first two measurements (spheres A and B) the third measurement must not lie on the C_AC_B three-dimensional line, and also it must intersect the formed circular plane, with center M and radius r_m. Otherwise, if the sphere C is tangent or is not coinciding with the aforementioned plane, it will be discarded from the equation system. The aforementioned figure, shows a sphere C that satisfies the above criteria, as it defines two intersection points, the I and J respectively. In this way, two possible areas have been initialized which contain our target position.

Thenceforth, we define the middle plane of the I and J points, which is perpendicular to the vector $\vec{IJ}$. Every new measurement, like the sphere D that is illustrated in Figure 4.41, is acceptable if its measuring position does not belong on the formed middle plane. In this way, we prevent having a new sphere that is tangent to our current intersection system, as we restrict our system on particular solution areas, achieving faster

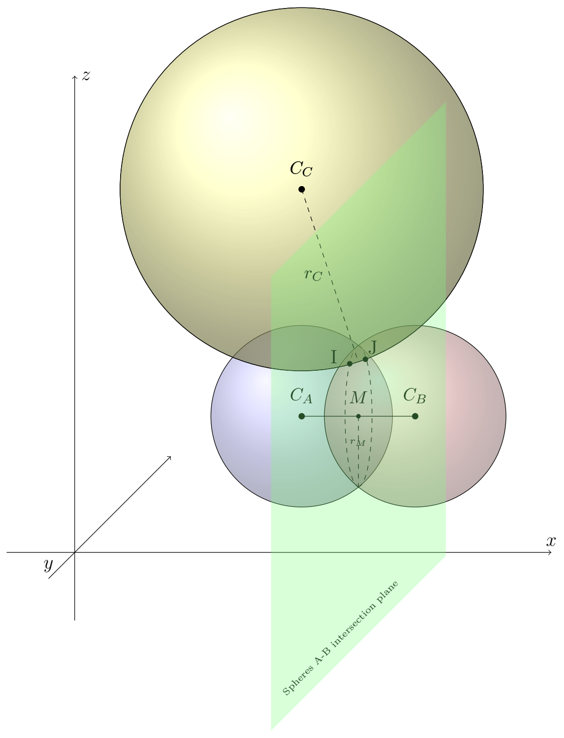


Figure 4.40: Illustration of the first three ranging measurements, in the world area.

convergence on our least-squares approach. The detailed measurement validity criteria algorithm is presented on Algorithm Table 3.

4.4.4.4 Voxel-Based Filtering on UAV Measuring Positions

Meanwhile, to avoid close range capturing positions and thus enrich the measuring spatial sparseness, we suggest a voxel-based spatial filter applied in realtime during the UAV flight. In specific, we consider the flying area map as a 3D voxelized grid map, in which a single distance measurement can be acquired in each predefined voxel area.

During the flight operation in the surveying environment, the UAV can autonomously detect and distance measure each target object that may be observed along its trajectory. Each time a range measurement occurs, it is correlated with the current UAV global position and stored locally in the UAV's system. By applying the proposed voxel-based filter, the containing voxel areas are updated each time a distance measurement is added within their area and they are marked as occupied. Hence, the voxel-based filter disallows future UAV measurements within the preoccupied voxel areas, as they already contain

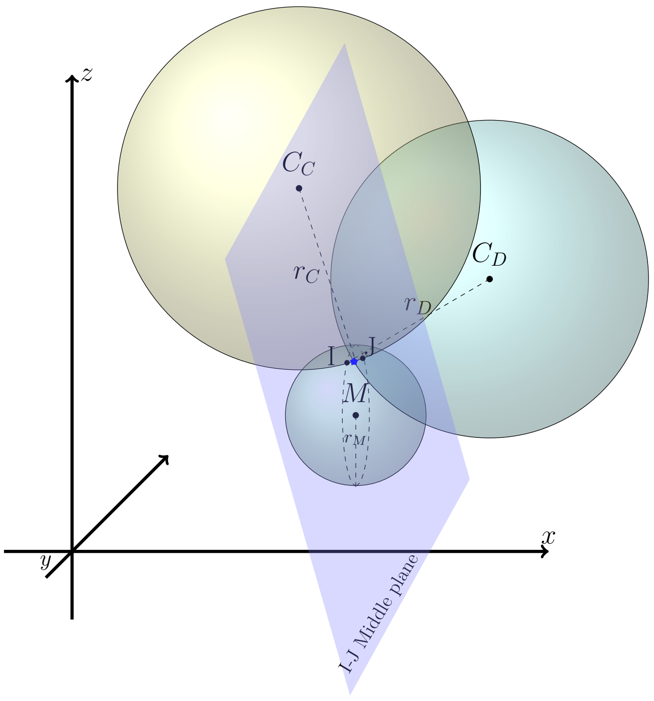


Figure 4.41: The addition of the fourth ranging measurement.

spatial information, and thus it indicates neighboring blank voxel areas for visitation.

Henceforth, the UAV captures distance measurements from the viewing ground target as sparse as desired, according to the selected voxel-filter size, and thus spatially enhances the spheres' equation system. Figure 4.42 illustrates the three-dimensional world area, as it is devised by a variety of voxel cubes in size.

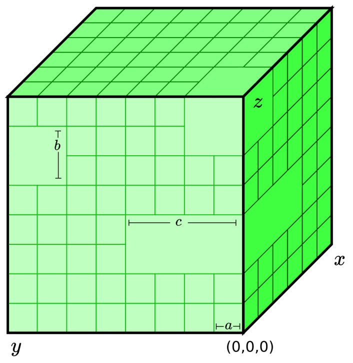


Figure 4.42: Voxelized-segmented environment area.

The final voxel-grid map contains the sparse distance measurements that were captured along the UAV's path with their corresponding UAV's global positions and can be parsed to the multilateration positioning system to estimate the target's global position.

4.4.5 Search-and-Rescue Map and Visualization of Positioned Targets

Since the positioning procedure is completed, the positioned target is stored locally and inserted into the 2D SLAM map. In this way, the first-responders have a belief of the current area condition, the detected targets positions (both in local and global coordinates), and also the current UAV's operating status and pose. To achieve this visualization, we utilize the features of RViZ tool, which are ideal for displaying current ROS topics information since it requires low computational resources. Hence, even with limited network connection, the responders will be able to obtain the current Search-and-Rescue map, as structured by the UAV (if we consider that the UAV is the ROS master), or acquire the new positioning data and update the local map status in the first-responders' ground control station. Consequently, the rescuers can proceed with their plans and decide for their upcoming operation actions, accordingly.

Figures in 4.43 illustrate a positioning scenario, in which the UAV has successfully positioned three targets in the unknown area and tagged them in the created map. It should be pointed out that, the variation of the stored target identifications of the RViZ rescuing map and the depicted on the camera plane, stems from the fact that the latter is used only for the UAV's mapping procedure, as they represent the amount of total positive areas that have been counted until now. Hence, the first-responders will have the final representation of the RViZ rescuing map in order to plan their actions and proceed with their SAR operation.

4.4.6 Occupied Area Coverage and Unexplored Area Investigation

Since the UAV can navigate and localize targets autonomously in the unknown environment, it is essential to conserve a belief of the visited areas and to focus on unexplored map sections when the coverage of the current area is sufficient. For this reason, we enhanced the navigating behavior by being capable to extract and pinpoint such areas from the generated map and thus make the UAV system pivot towards them.

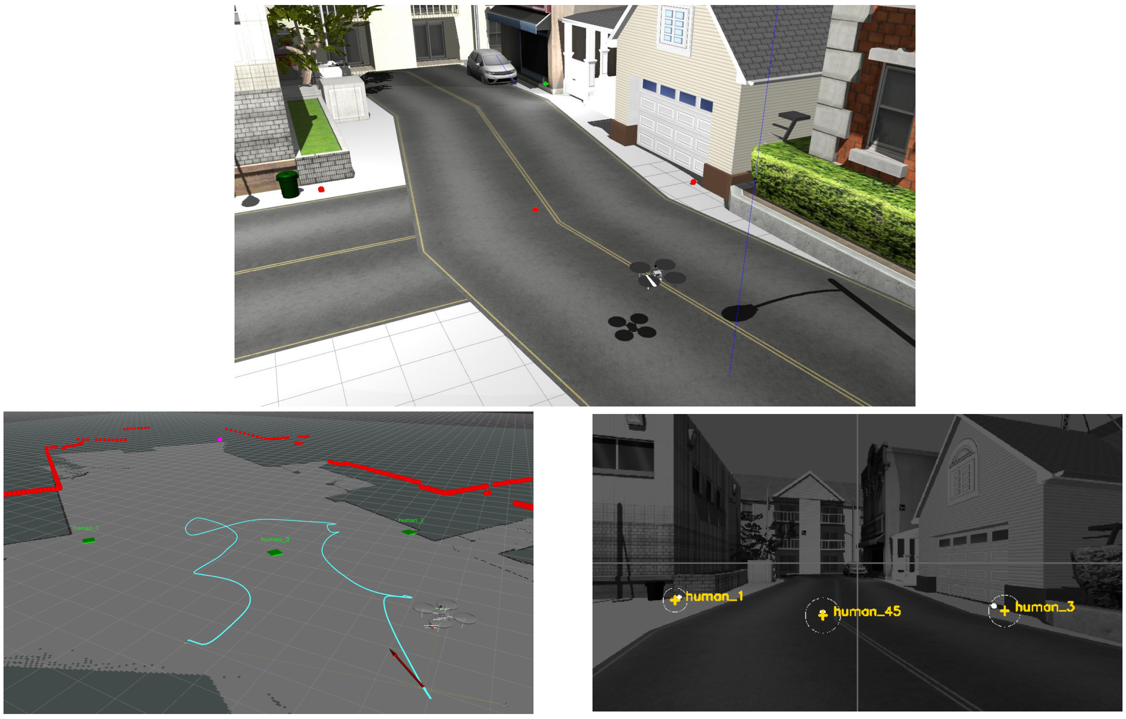


Figure 4.43: Targets localization by the UAV and their illustration in the rescuing map. The green markers depict the approximated positions of mapped targets. As the targets have been positioned, they are viewable within the image frame of the UAV's cameras.

During the mapping procedure, the occupancy grid map is filled with values which indicate cells' occupation status. Moreover, as the UAV is navigating in the area, there are regions in the map that are completely formed and others that contain unexplored grid cells. Also, due to the continuous gimbal scanning movement, a significant environment area is scanned along the way of the UAV system.

Hence, in cases that there are no new targets available and the UAV system assumes that it has adequately examined the existing area, it orientates towards the unoccupied map sections. In particular, during the map generation, there are occupied-free grid cells that neighbors with undiscovered grid cells. These areas indicate map spaces in which the UAV can further explore to fill the unregistered cells. To assume that an area is completely mapped it must form a closed shape form outline, without any discontinuities which may indicate partial area examination. Figures in 4.44 shows the status of the map at the start of the mapping procedure, and the plethora of unexplored regions that need to be furtherly mapped.

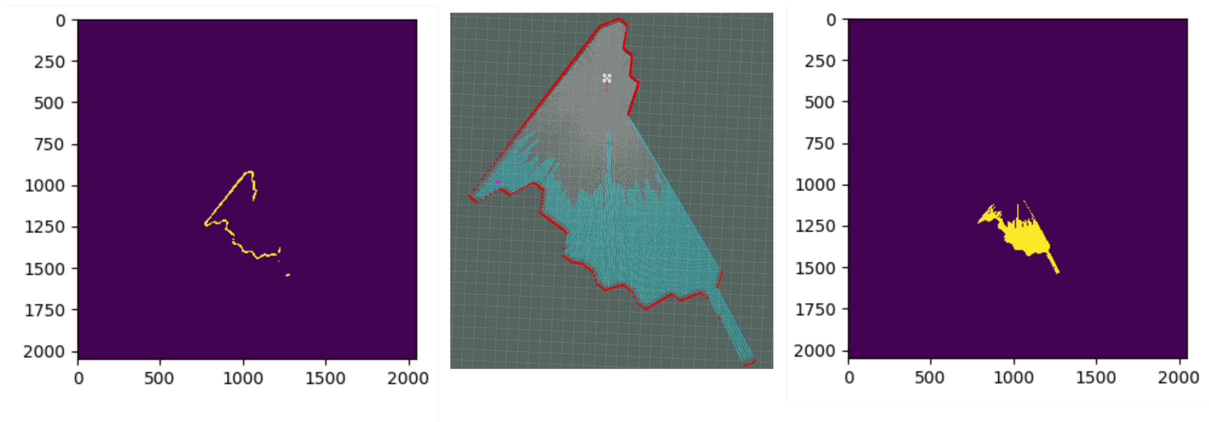


Figure 4.44: Unknown areas extraction. The left image illustrates the current map boundaries (obstacles) and the right image illustrates grid cells that need to be classified. The middle image show the fusion of the current map status along with the detected maps areas that need further exploration.

To detect potential regions for investigation, we initially isolate the occupied shape of the current map and extracted its exterior contour. Thus, we can determine which places are open-unconnected by comparing the contour shape along with the discontinuous map outline. Every region of them is stored separately, and they are spatially registered and sorted depending on their map positions and sizes.

Henceforth, in cases that the UAV can not locate any new target, it orientates towards the prioritized registered unknown area section. This behavior characteristic is sufficient for the UAV to achieve complete search in its existing area along with the autonomous scanning movements of its mounted gimbal system, and thus to decide to move into an unexplored section since the current area is covered. The undiscovered grid cells are published also in a separate *nav_msgs/OccupancyGrid* message, for real-time inspection purposes. Lastly, a brief algorithmic concept of the UAV total behavior, is developed in Algorithm Table 2 and the Figure 4.46 illustrate the package node's inner-communication and message exchange.

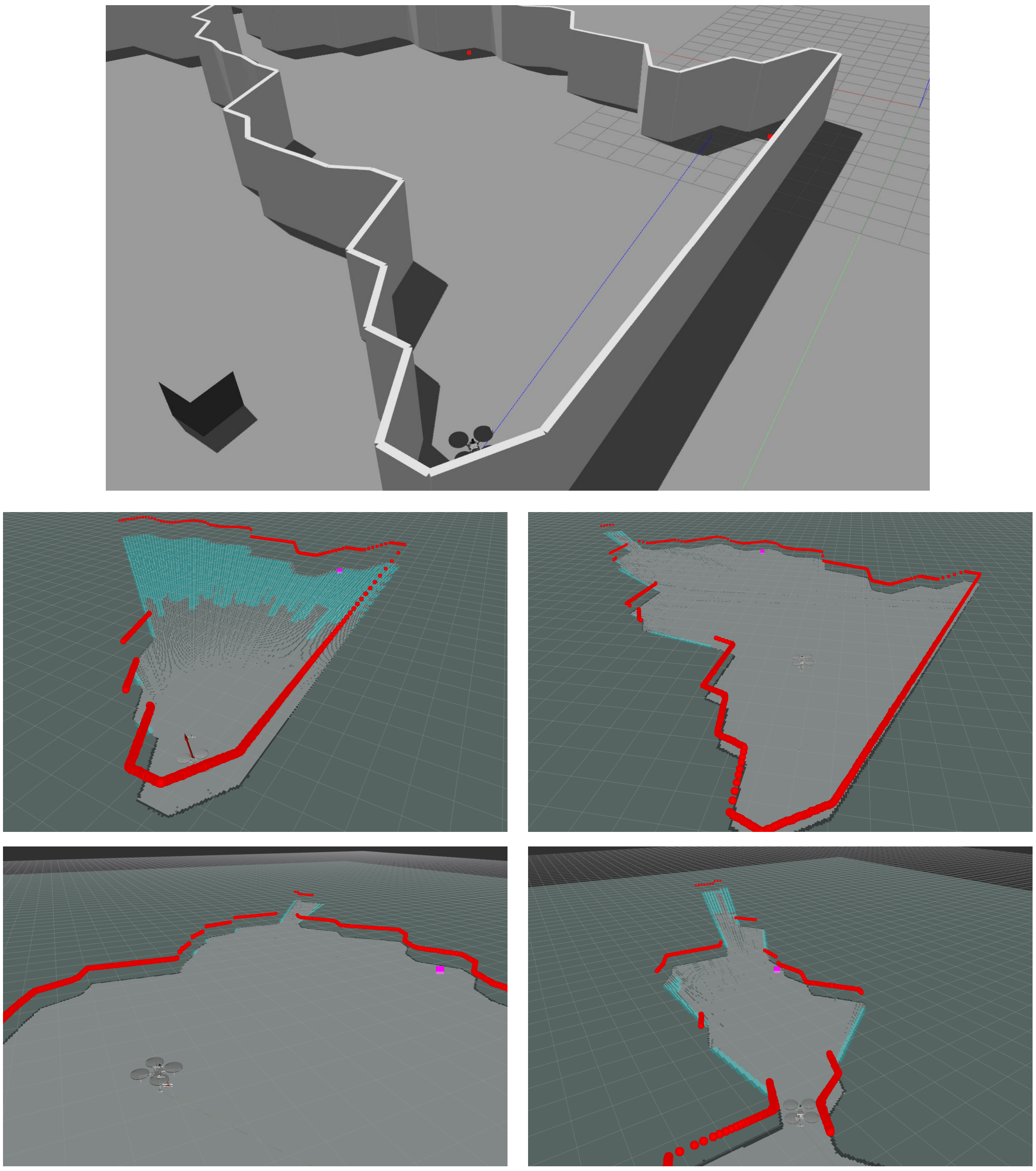


Figure 4.45: Snapshots during the UAV mapping procedure. The blue areas indicate map cells that can be furtherly investigated.

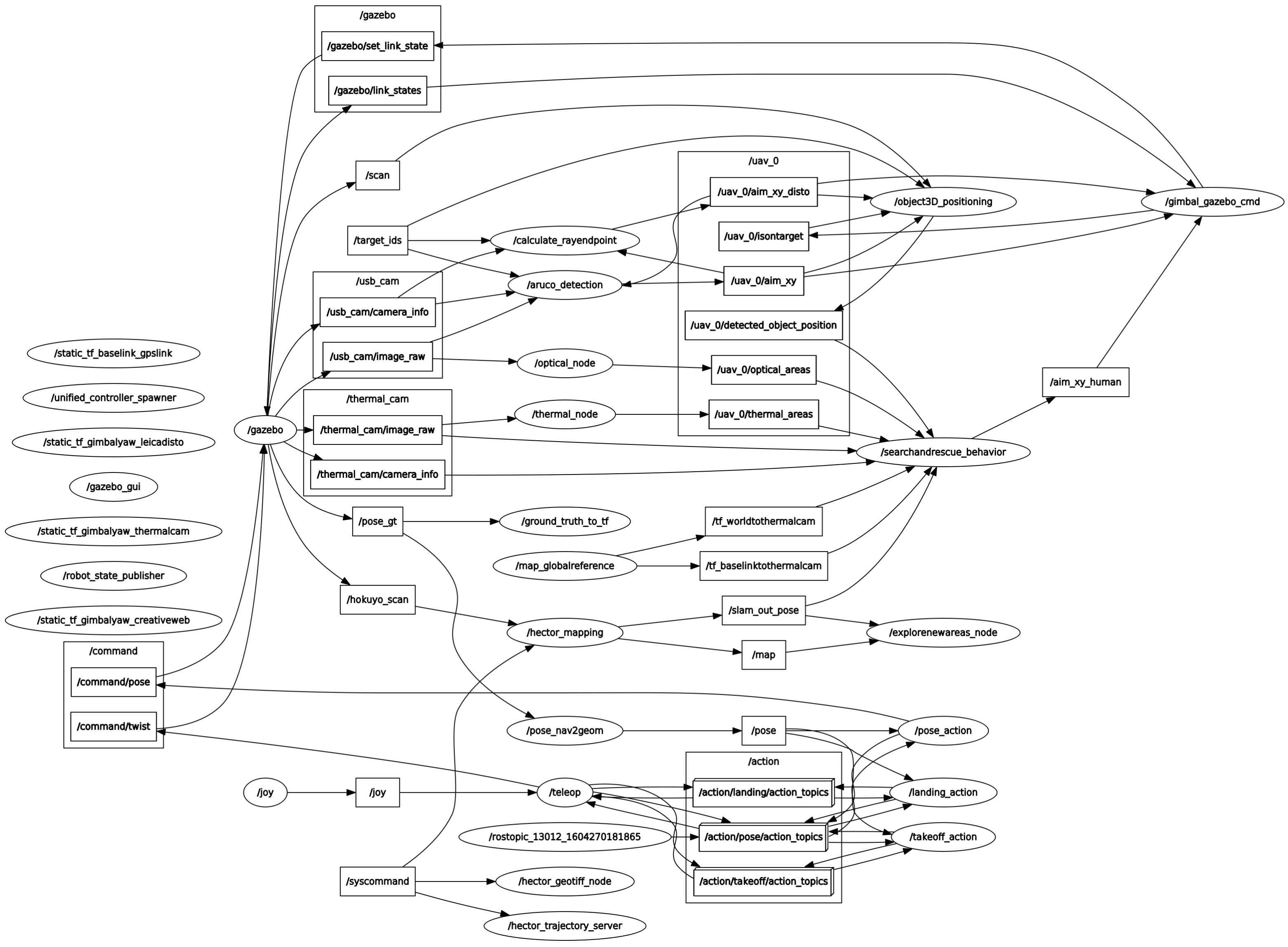


Figure 4.46: The ROS graph of the package. Every ROS node is illustrated with an oval shape and each ROS topic with a rectangle. The arrow lines show the topic publications and subscriptions of each node and the names of the group rectangles represent the topic namespaces.

Algorithm 3: Measurements Criteria Node

```
1  select Sphere A ($C_A, r_A$);
2  select Sphere B ($C_B, r_B$);
3  find Spheres A, B intersection plane, which is perpendicular on line $C_AC_B$ at
     point $M$;
4  if $length(C_AM) \geq r_A$ or $length(C_BM) \geq r_B$ then
5      return ;                              // the spheres are not intersecting.
   /* else, we have an circular intersection plane.                         */
6  define Spheres A, B intersection circular plane as $\mathcal{M}$ ($M, r_M$);
7  find Sphere M, constructed by plane $\mathcal{M}$;
   /* seeking for the third sphere ...                                      */
8  while True do
9      select Sphere C ($C_C, r_C$);
10     if $C_C \in C_AC_B$ line then
11         continue;
       /* we use the sphere M and plane $\mathcal{M}$ to evaluate sphere C ...    */
12     find Spheres M, C intersection plane, which is perpendicular to the $MC_C$ line
         at point $N$;
13     if $length(C_CN) \geq r_C$ or $length(MN) \geq r_M$ then
14         continue;                         // the spheres are not intersecting.
15     if Sphere C does not intersect plane $\mathcal{M}$ then
16         continue;
17     break;                                // third sphere has been founded.
18 define Spheres M, C intersection circular plane as $\mathcal{N}$ ($N, r_N$);
19 find Intersection points of circular $\mathcal{M}$ and $\mathcal{N}$, that are $I$ and $J$;
20 define Middle Plane, which is perpendicular to $\vec{IJ}$, as $\mathcal{O}$;
   /* seeking for the new spheres ...                                       */
21 while True do
       /* as long as we get new measurements ...                            */
22     while True do
23         select New Sphere ($C_S, r_S$);
24         if $C_S \in C_AC_B$ line or $C_S \in \mathcal{O}$ plane then
25             continue;
26         break;                    // the new sphere satisfies the criteria.
```

Chapter 5

Results

In this chapter, we present the results of UAV's learned behaviors. In specific, we show the results of both navigation and human detection training procedures and underline important remarks that have been occurred throughout the tests. Given these results, we evaluate the overall performance of the proposed autonomous approach for reconnaissance in Search-and-Rescue scenarios, by performing experiments in cluttered and custom-generated Gazebo environments.

5.1 Autonomous Navigation and Experiments

5.1.1 Training and Evaluation of the Navigational Behavior

In order to acquire the most appropriate and robust navigating behavior, we trained the aforementioned system under variations of the actor-critic neural networks and training environments. In specific, we test the agent's performance by training on various network sizes, based on different loss functions, and we also use different environment narrowness levels and map update rates during training, to evaluate the agent's generalization behavior.

Actor and Critic Networks' Modifications

At first, we employ different actor and critic networks' formulations, under common Gym environment parameters, to evaluate their learning ability and overall performance. Hence, we initially built a Mean Squared Error model and we performed training pro-

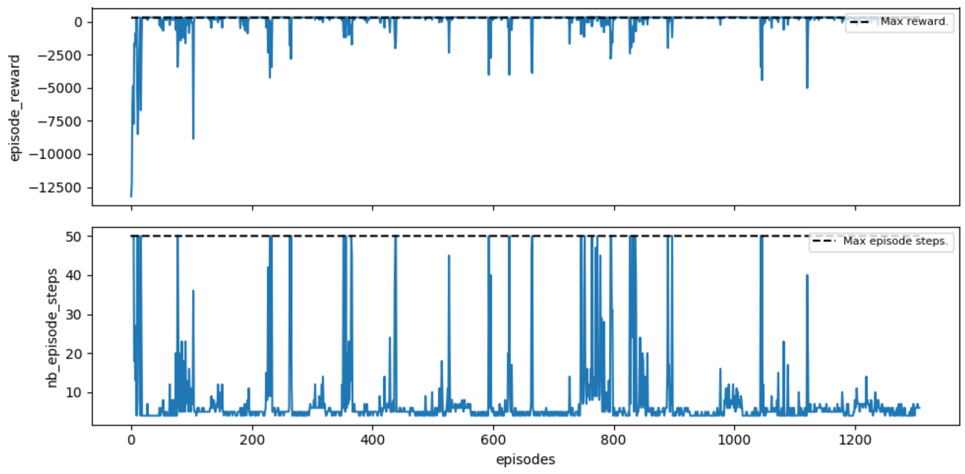


Figure 5.1: Training procedure of the OpenAI UAV agent, with 128-sized dense-layered critic and actor networks, Mean Squared Error loss function, and map update rate of 30.

cedures for 5000 episodes. Figures in 5.1 show the accumulated rewards of the first 1300 episodes of training, simultaneously with the minimization of the recurred steps to successfully reach the final state.

During our experiments, we evaluated the performance of two loss functions, namely the Mean Squared Error and the Mean Absolute Error, as defined in Equations 2.16 and 2.17, respectively. Figure 5.2 show the early episode rewards of both functions as applied on various model sizes, under common testing environment specifications. As it is apparent from the line plots, the two functions behave similarly across the tested models and seem to converge more efficiently on the bigger neural network models.

Hence, a vital feature that affects the agent's learning ability is the size of the model's hidden dense layers. For this matter, we performed training procedures, under common loss function and common environment properties, to evaluate the performance of the difference in size models. As it may be evident from Figure 5.3 and also from Figure 5.2, the increase of the model size implies early convergence and efficiency of the trained agents.

Additionally, Figure 5.4 illustrates the MAE loss value decrease for the implemented networks, which conveys that bigger dense-layered networks tend to accumulate more information and achieve better accuracies as they minimize their loss value.

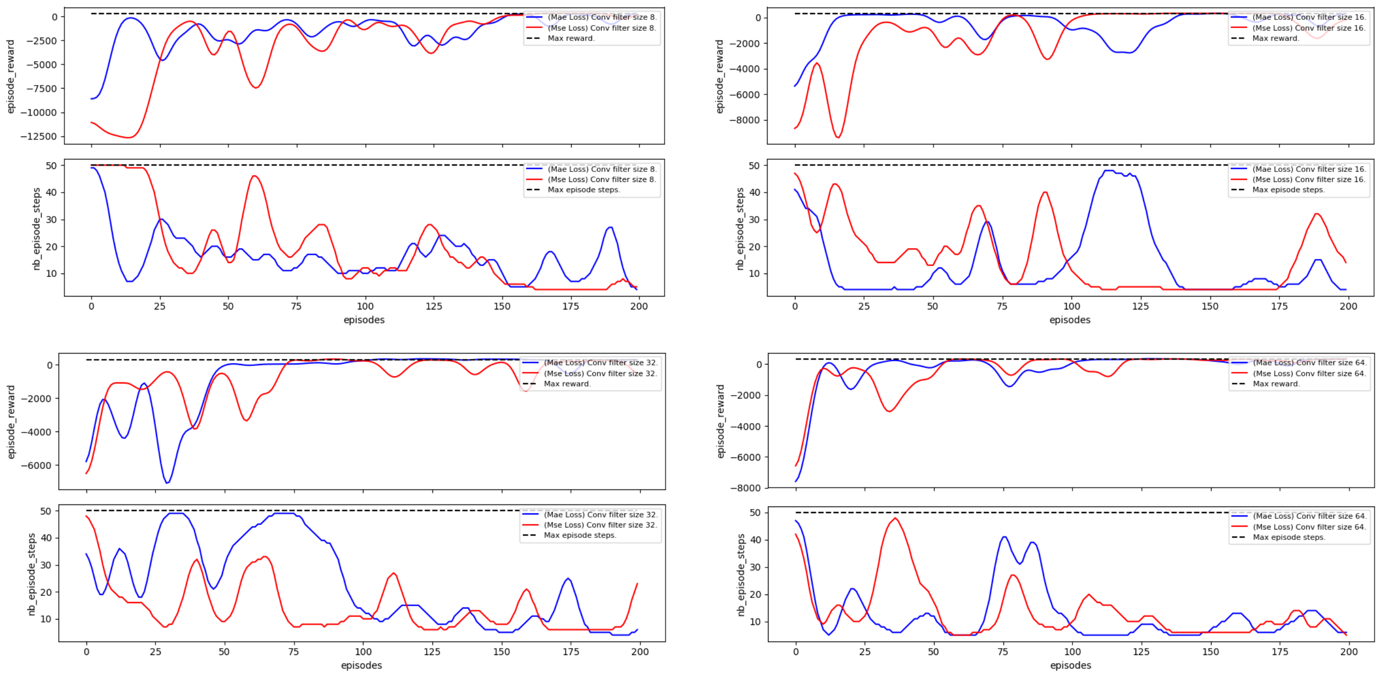


Figure 5.2: Training on Mean Squared Error and Mean Absolute Error loss function models, for 8, 16, 32, and 64 dense layer dimensions.

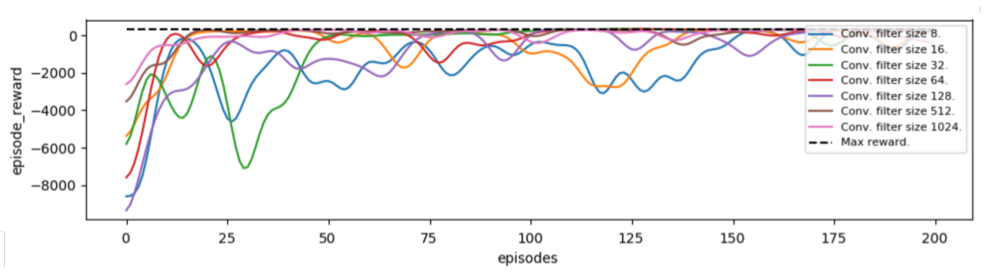


Figure 5.3: The episode rewards for the different in size dense layers of the actor and critic neural networks.

Training Environment Modifications

Meanwhile, as we have mentioned in Section 4.2.2, the autogenerated Gym worlds can be formed ideally based on specified spatial and map update parameters. In specific, every autogenerated world is created in respect of the predefined minimum and maximum narrowness level values, the length value, and also the map switch update rate. Henceforth, in this part, we keep the neural networks' structure stable, as we utilize 128

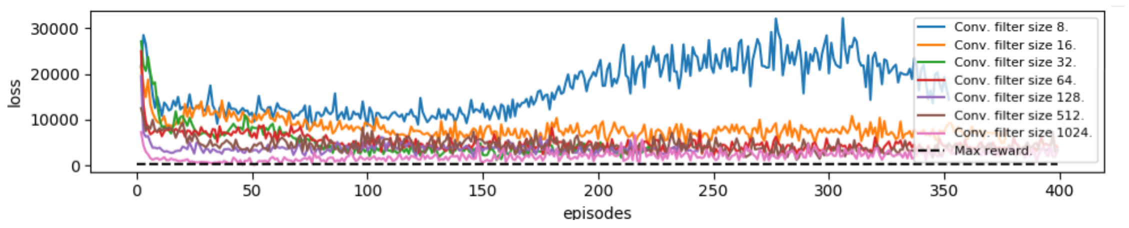


Figure 5.4: The MAE loss value for the different in size dense layers of the actor and critic neural networks.

dense-layered networks with MAE loss function, and we evaluate the UAV autonomous navigation performance on different map widths and training map switching rates.

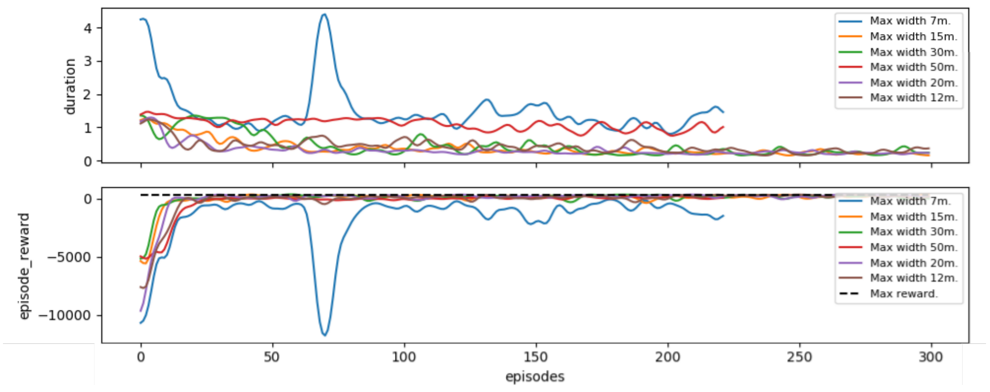


Figure 5.5: Agent training under different map maximum widths.

First, as we assume that we have a stable in length environment world (20 meters), we train our UAV agent on a range of maximum width limitations. In this way, we will evaluate the UAV's learning experience on diverse world's spatial formations, which may restrict its navigational abilities.

In particular, Figures 5.5 illustrate the results for various widths by applying map switches every 30 completed episodes. The line plots portray that environments that are either too narrow or too wide, complicate the agent's navigating behavior, whilst worlds with moderate maximum widths present models that have learned to maneuver successfully in the unknown area. In specific, as it is depicted by the first line plot of Figure 5.5, UAV agents that have been trained on environments with a max width range of 12-30 meters, spend less time to reach the desired goal which is at the end of the map length. Hence, this occurrence indicates that the UAV agent needs neither narrow nor wide maps to gain experience on because these maps can be difficult or too ambiguous to obtain an appropriate navigation technique in a certain number of episodes.

Furthermore, it is essential to evaluate the affection of the interchanging rate of the environment world, used for the learning procedure. For example, if we assume that we change an environment every 10 episode completions, the agent will see in 50 different environments in 5000 training episodes. Thus, it was quite important to check if this feature of out created OpenAI Gym, enhances the agent's behavior on safely navigating in new and unforeseen environments.

For this reason, we utilized the 128-sized dense-layered actor and critic networks, with the MAE loss function, to train for 5000 episodes, on different environment switching rates. Since we acquire the weights of these agents, we loaded and tested them on 100 new maps, to check their performance and gained rewards. From our experiments, we see that in small map interchanging rates (less than 1:10) the agent performs poorly in the unforeseen maps, as it has relatively low and negative scores. On the other side, in cases with higher map-generating rates (greater than 1:200), the simulated UAV scores are also low and cannot in some cases reach the final goal. Nonetheless, at moderate rates (1:30 to 1:80), the UAV behavior shows notably better results, as it acquires greater accumulated rewards and reaches the goal in most of the experiments. This occurrence indicates that we can actively affect the agent's learning procedure by adjusting the refreshness-rate of the training environment, which can either imply a more monotonous and poor navigational behavior in new environment instances or to more generalized and "mature" behavior to face the various and newly occurred environment conditions.

Overall, we concluded that by using a moderate map-update ratio and maps that are neither too wide nor narrow, results to the enhancement of the agent's overall navigating performance and environment adaptability. Also, it is noteworthy that the training procedure of one episode in our OpenAI Gym environment takes approximately 0.027 seconds. The vast majority of the aforementioned models converged and showed desired results roughly before the progression of the second minute and their total training time of 5000 episodes lasted approximately 2.25 minutes. Each time there was a training procedure, the resulted weights were store locally and named accordingly to indicate information about the used networks' sizes, environment parameters, and timestamps to distinguish them from other instances. Hence, this fact enables fast training and evaluation procedures, and thus, facilitates the acquirement of the most appropriate and desired agent's behavior.

5.1.2 Application of the Trained Models in the Gazebo and Results

As the model has been trained in the OpenAI Gym environment, the succeeding step is to transfer and apply the learned model into the Gazebo environment on to the simulated Matrice 100 model. For this reason, a middle-node is developed to load the pre-trained weights of the OpenAI Gym behavior and to communicate with the Gazebo simulator, in order to send pose commands according to the captured Hokuyo LiDAR ranging data. In this way, the trained models can be fully utilized and evaluated in the Gazebo world, in various scenarios without the need of using the Gazebo simulator for training procedures.

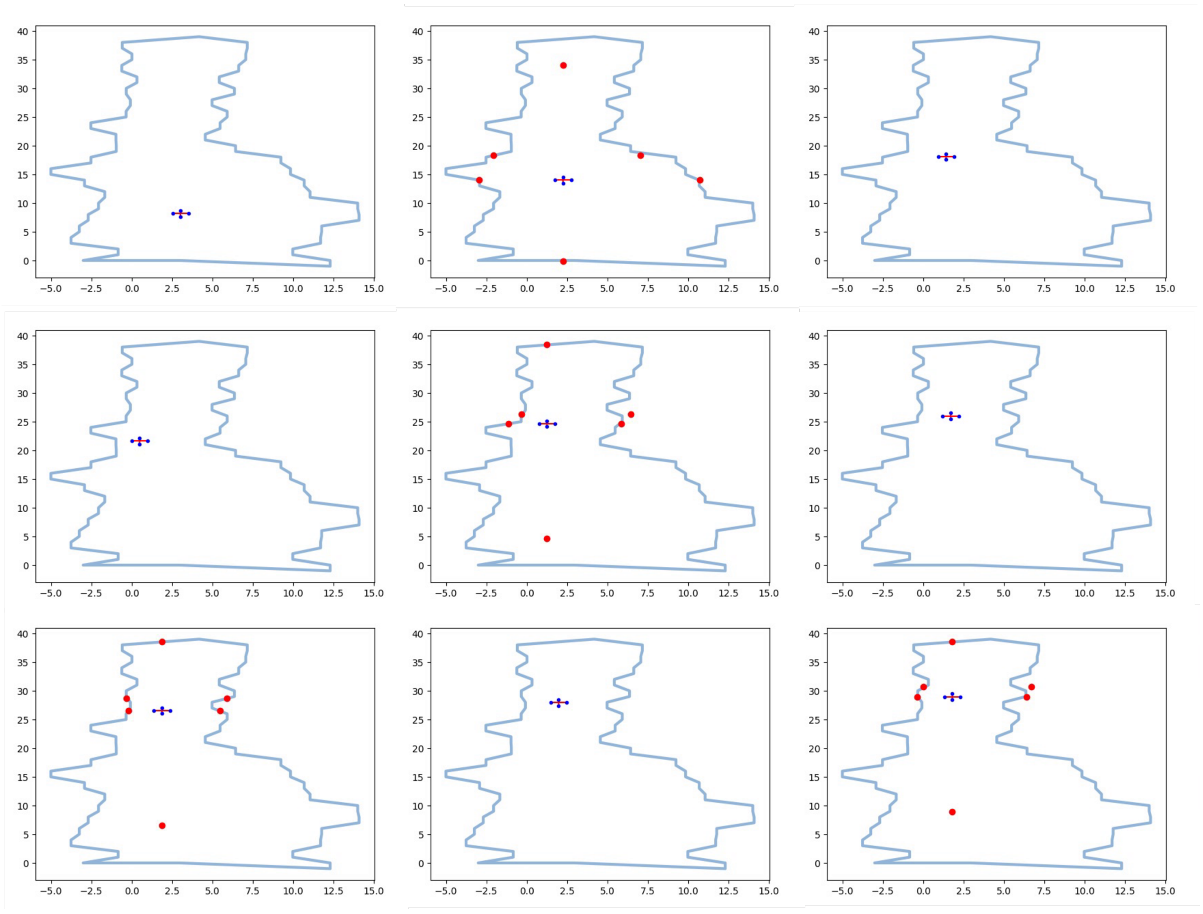


Figure 5.6: Moments of an OpenAI Gym experiment UAV movement in a randomly generated Gazebo world.

Initially, this middle node subscribes to the UAV's 2D LiDAR topic, to capture real-

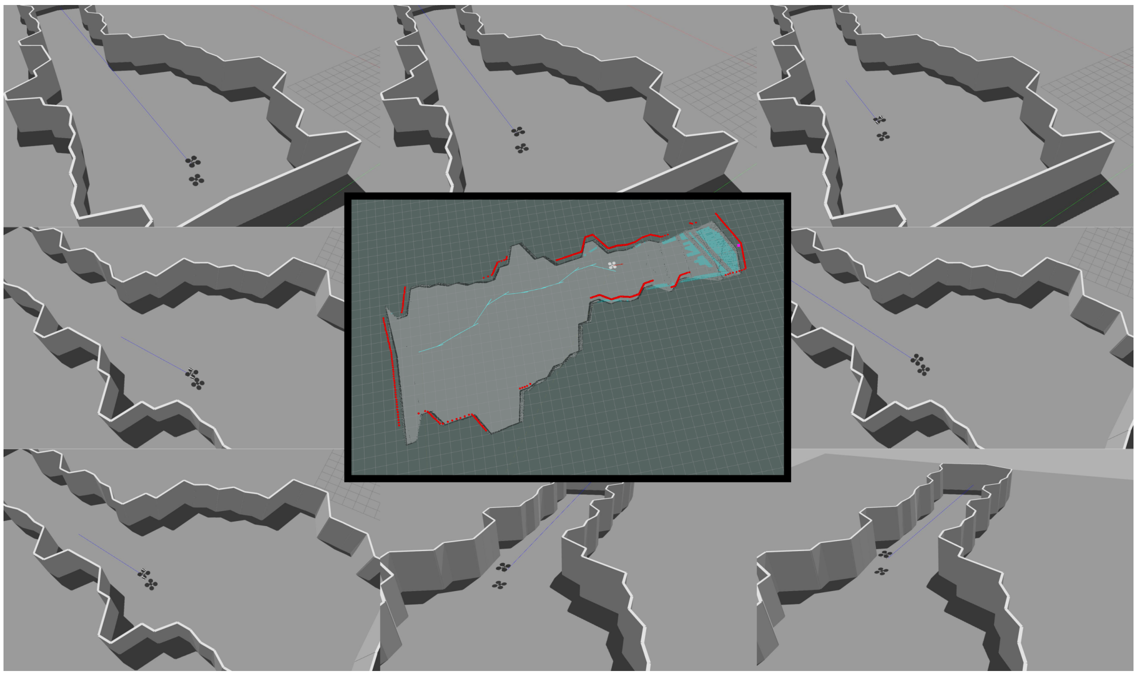


Figure 5.7: Moments of autonomous UAV movement in a randomly generated Gazebo world. The center image illustrates the current map and UAV position belief, as appeared in RvIZ tool. The aqua line depicts the UAV trajectory within the map.

time ranging data from the onboard LiDAR. Given that, the data are filtered to obtain the distance measurements in $\theta = [0°, \pm 45°, \pm 90°, 180°]$ directions, similarly to the OpenAI Gym simulator. Due to the Hokuyo LiDAR model ranging resolution, we average 5 distance values for each of the aforementioned scanning directions. Thenceforth, these measurements are parsed in the pre-trained model as observations, and thus, new movement commands are acquired for the UAV system to follow. Figures in 5.6 show moments during the autonomous navigation of the UAV in a random polygon map in the OpenAI simulator. Figures in 5.7 illustrate the exact same map construction and autonomous navigation of the M100 system in the Gazebo world. As it is clear, the UAV behaves similarly in the Gazebo environment, as it utilizes successfully the obtained knowledge from prior training procedures that occurred in the OpenAI simulator environment.

Overall, as mentioned in Section 4.2.3, the UAV has learned to translate in x, y axis directions. As soon as a target is observed in an extreme position relative to the UAV body, only orientation movements are applied until the target position becomes centered with the UAV heading orientation. As the target is centered in respect of the

UAV heading orientation, the UAV accepts pose commands captured from the used pre-trained model, and thus moves towards the target. Also, during the UAV path, random orientation movements are performed (since there is not any target observed) and also orientates towards unknown areas if it assumes that the current area has been sufficiently covered. It is worth noting that, as soon as a target is detected and the UAV is moving towards it, movement reduction is applied in order to be more precise during the target positioning procedure.

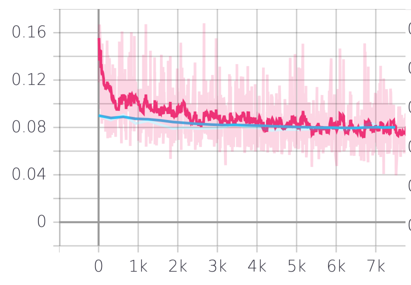
5.2 Autonomous Human Detection and Experiments

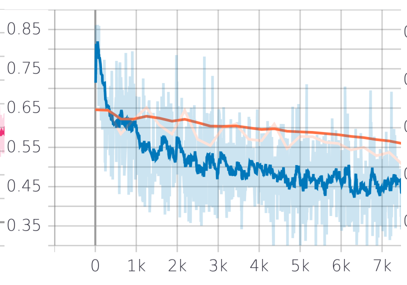
Initially, since our future goal is to develop a real UAV system to operate in emergency situations, the developed autonomous detector was initially trained and tested under real imagery data. Therefore, as we conclude with the appropriate network architecture and characteristics, the system was trained from scratch on a custom dataset that was created within the Gazebo software and tested in various simulation environments.

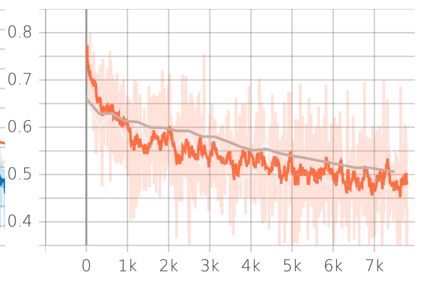
5.2.1 Training on COCO Dataset and Detection Performance

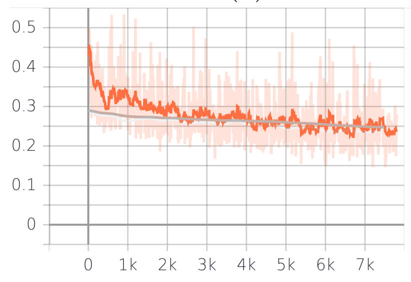
To evaluate the neural network performance in human detection, we selected 5000 images from the COCO training dataset of 2017 to use for training and 50 additional images for testing purposes. In both training and testing datasets, we have included images that either contain or no depictions of people in their frames, to further check the detector performance in various situations. Also, since the COCO dataset contains images with different dimensions, we rescaled their sizes to match the resolution of the UAV's optical camera, and also downscaled them in respect to 4:3 ratio, namely the 256×256 resolution, to minimize detection time and make the neural network demands appropriate for onboard and online execution.

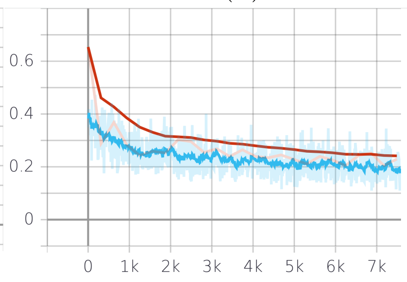
As we have presented notable metrics and loss functions which are made ideally for semantic segmentation scenarios, we had to evaluate their performance in detection accuracy under a common imagery dataset. For this reason, we initialized the neural network model and performed the same training procedure, by altering the used loss function at each time. During the training, we keep performance information for each implemented metric, and also we test custom loss functions in order to furtherly check the system's behavior. In particular, we constructed and trained the detector system on

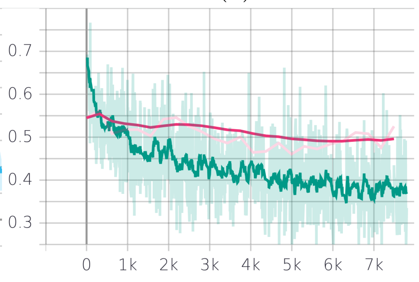

(a)

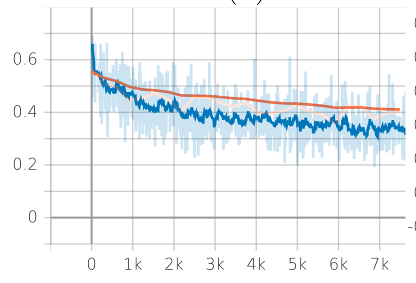

(b)

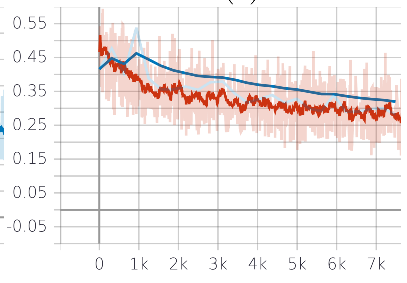

(c)

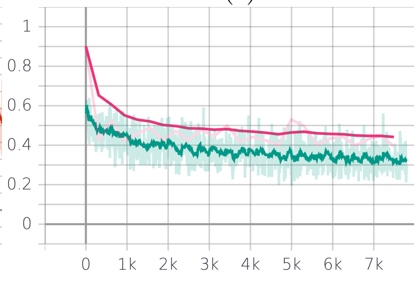

(d)

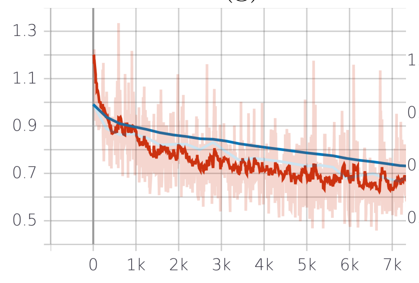

(e)

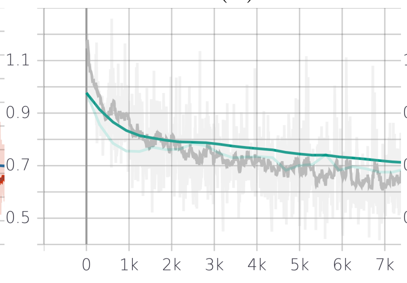

(f)


(g)


(h)


(i)


(j)


(k)

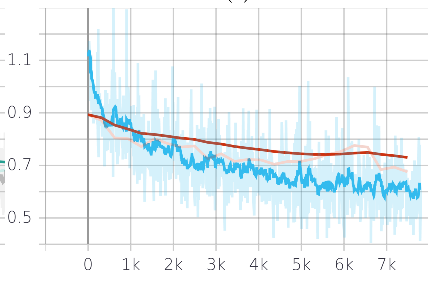
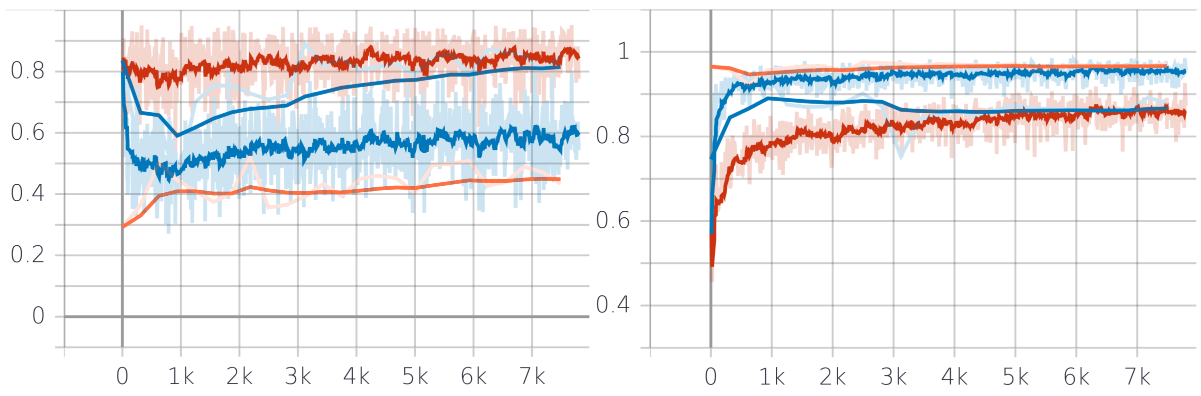

(l)

Figure 5.8: Loss function values during training and testing procedures for COCO dataset (25 epochs). In each figure, the fluctuating lines depict the loss values during the training process and the smooth lines during the testing. Figures (a) : MSE, (b) : MSE + DL, (c) : IoU, (d) : BCE , (e) : $F_\beta(\beta = 2)$, (f) : $F_\beta(\beta = 0.5)$, (g) Tversky($\alpha = 0.7$, $\beta = 0.3$), (h) : Tversky($\alpha = 0.3$, $\beta = 0.7$), (i) : DL, (j) : BCE + DL, (k) : BCE + Tversky($\alpha = 0.7$, $\beta = 0.3$) and (l) : BCE + Tversky($\alpha = 0.3$, $\beta = 0.7$).

the following loss functions,

- Mean Squared Error.
- Binary Cross Entropy.
- Dice Loss.
- Intersection over Union.
- Tversky Loss ($\alpha = 0.3$, $\beta = 0.7$).
- Tversky Loss ($\alpha = 0.7$, $\beta = 0.3$).
- F_β Score ($\beta = 2$).
- F_β Score ($\beta = 0.5$).
- Mean Squared Error + Dice Loss.
- Binary Cross Entropy + Tversky Loss ($\alpha = 0.3$, $\beta = 0.7$).
- Binary Cross Entropy + Tversky Loss ($\alpha = 0.7$, $\beta = 0.3$).



(a) *Sensitivity* value. (b) *Specificity* value.

Figure 5.9: Both Tversky Loss model performances, during training and testing procedures.

The aforementioned functions will affect differently the network's learning procedure, and thus will result in various model behaviors which will treat FP and FN evenly or learn to penalize one of the two quantities more than the other. Figures in 5.8 illustrate the training and testing curves for each loss function, under the same imagery dataset and network architecture. As it is clear, all loss functions aid the detector's learning ability, since the loss value is decreased over the epoch progression.

Meanwhile, it is quite interesting to evaluate the detector behavior which is trained by using the Tversky Loss function, under different α and β values. Figures in 5.9 depict the difference in $Specificity$ and $Sensitivity$ behavior of the two models, as they have learned to penalize dissimilarly the mispredicted positive and negative results. As it is obvious, the detector which had a higher value of the β parameter presents less FN, and consequently keeps better scores in the $Sensitivity$ metric. Similarly, the detector which utilized the Tversky loss function with greater α, compared with the β parameter, had better scores in the $Specificity$ metric, which is strictly connected with the FP quantity and the amount of correct negative predictions.

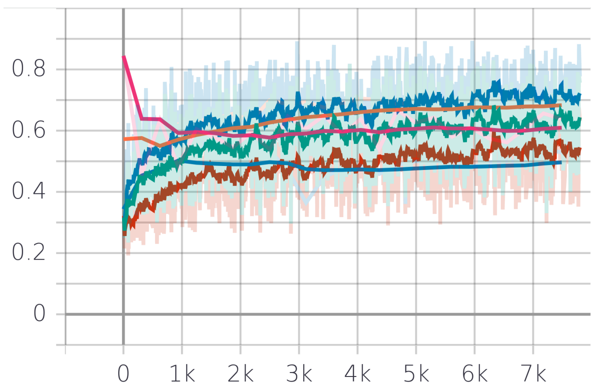
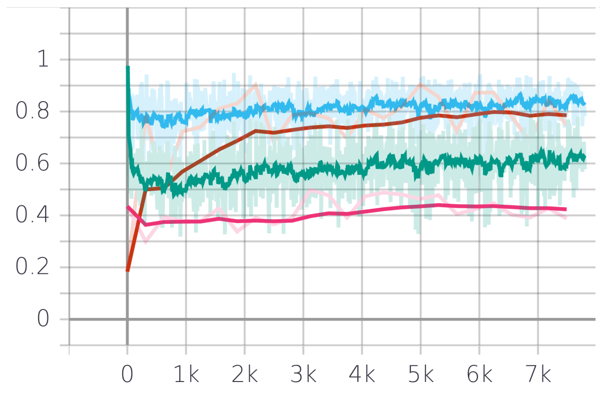


Figure 5.10: Tversky Loss models with the Dice loss trained model (middle line graph).

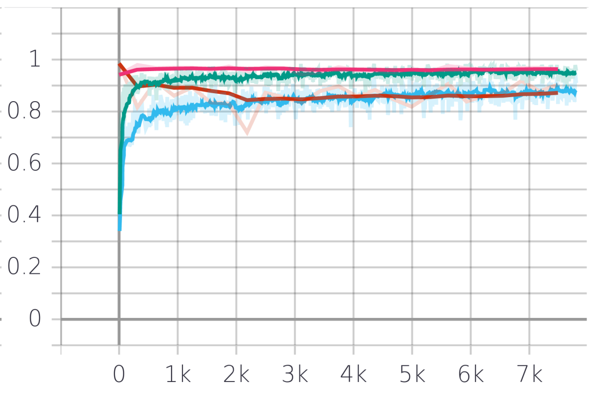
Figure 5.10 depicts the $Precision$ value during training, which also depends on TP and FP values, and pinpoints the Tversky with higher α value predominance. In this line graph, the Dice loss model is also plotted which has a more uniform behavior towards the FN and FP values, as is obvious from its moderate line trend.

Likewise, the behavior of F_β score presents different results depending on the value of β parameter. Specifically, for greater β values the model tends to penalize and mitigate the FN values, in contrast with lower β values. Figures 5.11 show the results of $Sensitivity$ and $Specificity$ parameters during training of the two F_β versions.

It is noteworthy that, F_β score behaves similarly with the Tversky loss, as by adjusting the F_β's parameter, we tune the importance of Sensitivity over Precision. For this reason,

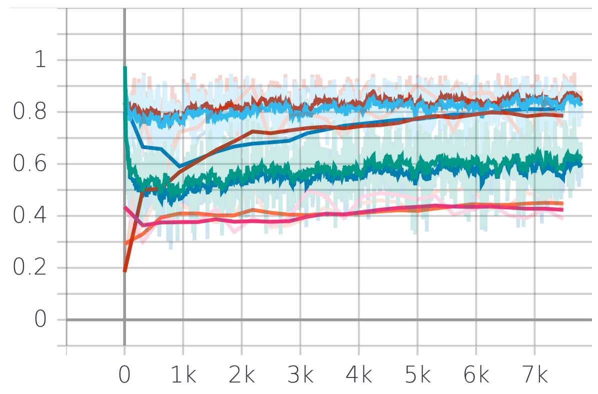


(a) *Sensitivity* value.

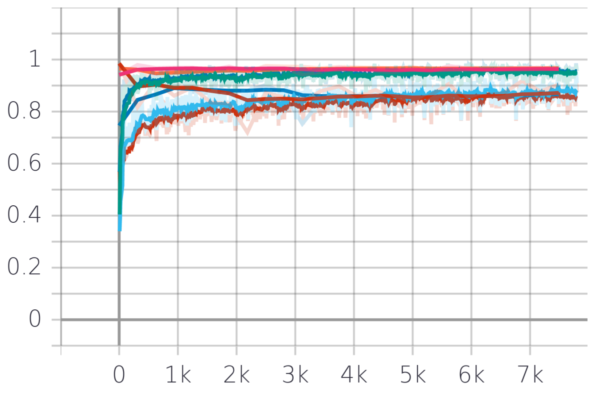


(b) *Specificity* value.

Figure 5.11: F_β loss models performances, during training and testing procedures.



(a) *Sensitivity* value.



(b) *Specificity* value.

Figure 5.12: Tversky and F_β loss models performance similarity.

as it is depicted in Figure 5.12, F_β presents similar results with Tversky-trained models, during testing.

Aptly, the Dice loss and IoU trained models present similar performances, as shown in Figure 5.13, due to their linear dependency.

In the meantime, we implemented custom loss functions as an amalgam of the aforementioned functions, in order to amalgamate their detection characteristics. First, we created a combination of BCE and Dice loss function, to contribute equally to the final calculated loss. The results, as depicted in Figure 5.14, show the creation of a model that preserves characteristics from both the aforementioned metrics since it illustrates an average score during testing, in both *Sensitivity* and *Specificity* criteria. In this way,

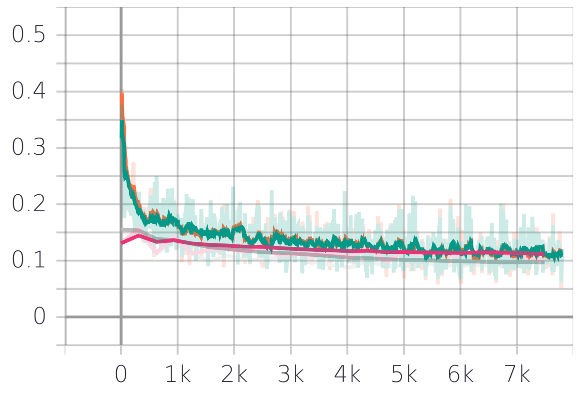
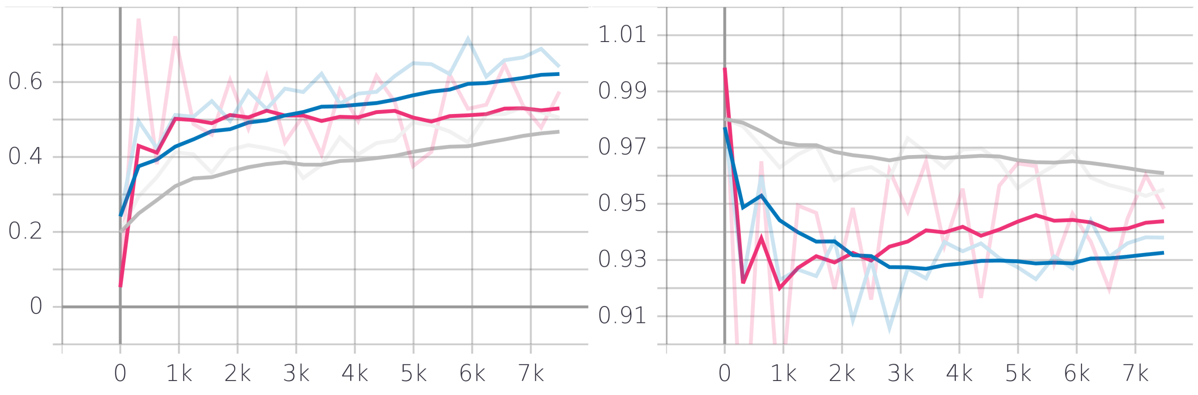


Figure 5.13: MSE loss values, for Dice Loss and IoU trained models.



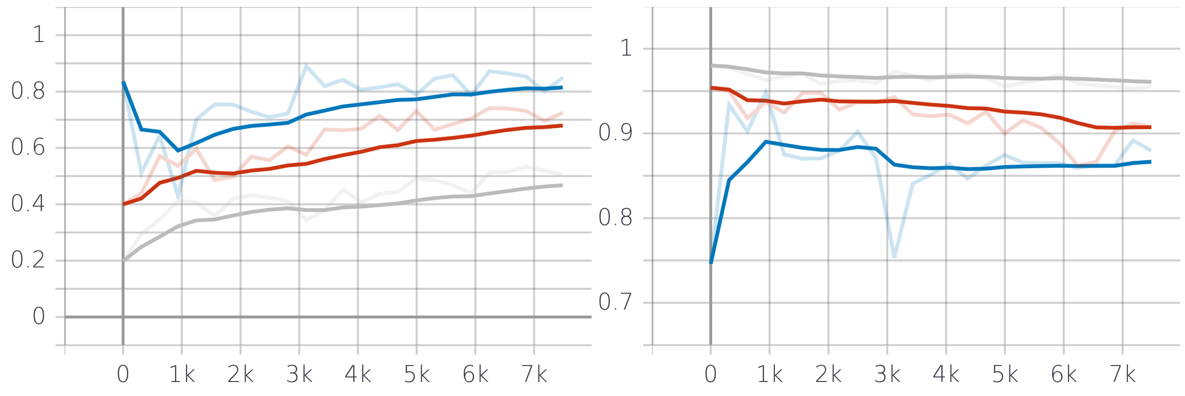
(a) *Sensitivity* value. (b) *Specificity* value.

Figure 5.14: Trained model with a combination of BCE and Dice loss functions. The pink line illustrates the testing scores of the combined model.

the fusion of both loss functions emerges to the improvement of the current model, as the new one has a more intermediate behavior compared with both metrics separately.

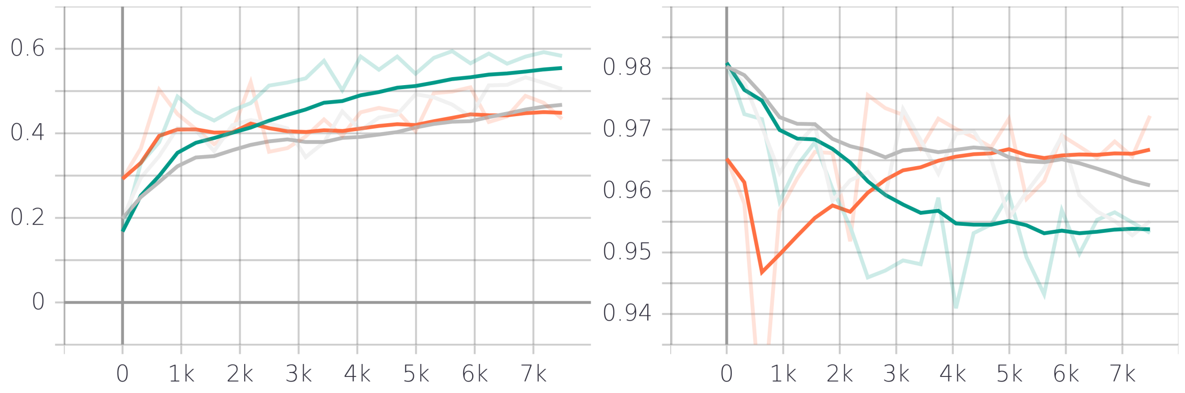
This occurrence is more notable in the second combination, of the BCE and Tversky($\alpha = 0.3$, $\beta = 0.7$) loss. In specific, Figure 5.15 illustrate that the scores of the testing procedure of the fused model, and the moderate behavior in both *Sensitivity* and *Specificity* metrics.

In the third combination of the BCE and Tversky($\alpha = 0.7$, $\beta = 0.3$) loss functions, implies a new model that presents an improved behavior in TP localization. Despite the *Specificity* performance which shows slight behavior differences, Figure 5.16 depicts the model increased *Sensitivity* scores during testing. As the Tversky($\alpha = 0.7$, $\beta = 0.3$) loss



(a) *Sensitivity* value. (b) *Specificity* value.

Figure 5.15: Trained model with a combination of BCE and Tversky($\alpha = 0.3$, $\beta = 0.7$) loss functions. The brown line illustrates the testing scores of the fused model.



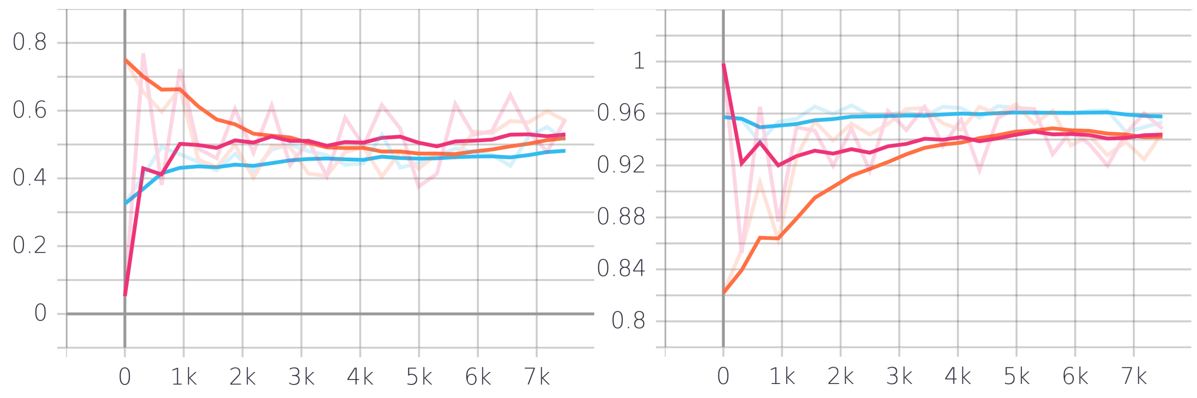
(a) *Sensitivity* value. (b) *Specificity* value.

Figure 5.16: Trained model with a combination of BCE and Tversky($\alpha = 0.7$, $\beta = 0.3$) loss functions. The green line illustrates the testing scores of the combined model.

function deals with the reduction of FP, the combination with the BCE loss function implies further enhancement of its detection behavior.

Last, the combination of the MSE and the Dice loss function presents minor improvements, as the model inherits characteristics from the latter loss function. Figure 5.17 illustrates the model's *Sensitivity* and *Specificity* on test data, during training.

Overall, Table 5.1 presents the scores of *Sensitivity*, *Specificity*, and *Accuracy* for each trained model, evaluated on the testing dataset. As it is clear, the detectors are improved along with the epochs progression as they indicate increasing scores. In terms



(a) *Sensitivity* value. (b) *Specificity* value.

Figure 5.17: Trained model with a combination of MSE and Dice loss functions. The orange line illustrates the testing scores of the combined model.

of positives, the majority of the detectors seem to perform better during training progression, and specifically, those that focus on TP detection and FP minimization (as the Tversky with $\beta = 0.7$ and Fβ with $\beta = 2$) have noteworthy performance. Also, every trained system can acquire good scores on negatives recognition ($Specificity$) early in the training process, since most images contain small proportions of positive areas compared with the negatives. For this reason, many detectors tend to mark the biggest amount of the images as negatives, and thus they yield notably good results in the $Specificity$ metric which may be misinformative about their overall performance.

In particular, the $Accuracy$ score is formed by taking both positive and negatives prediction into consideration, as is defined in 2.14. However, this metric may misinterpret the detector's performance, as it based on both $Sensitivity$ and $Specificity$ quantities. For example, in a case of a detector's assumption that the total image is negative, the $Specificity$ score may be significantly high because there may numerous negative areas in the viewing scene, in contrast with the $Sensitivity$ value which will be crucially small. The $Accuracy$ parameter in this situation will be high (probably bigger than 0.85), which will create the false notion that the detector is accurate. Henceforth, this is the primary reason that we always evaluate each $Sensitivity$ and $Specificity$ parameters in our models, to monitor simultaneously and independently their performances in TP and TN detection.

Overall, according to the metric results presented in Table 5.1, we would use the detector's version which had better scores in $Sensitivity$, since the vast majority of them

Loss Function\Epoch #	**5**	**15**	**25**	**5**	**15**	**25**	**5**	**15**	**25**
MSE	0.43	0.46	0.47	0.95	0.96	0.96	0.88	0.89	0.89
BCE	0.34	0.38	0.43	0.97	0.97	0.96	0.88	0.89	0.89
DL	0.48	0.51	0.52	0.93	0.94	0.94	0.87	0.88	0.89
IoU	0.61	0.62	0.67	0.91	0.93	0.93	0.87	0.88	0.90
Tversky($\alpha = 0.3, \beta = 0.7$)	0.65	0.73	0.78	0.88	0.86	0.87	0.85	0.85	0.85
Tversky($\alpha = 0.7, \beta = 0.3$)	0.40	0.41	0.43	0.96	0.96	0.96	0.88	0.89	0.89
Fβ($\beta = 2$)	0.63	0.72	0.76	0.88	0.86	0.87	0.85	0.84	0.85
Fβ($\beta = 0.5$)	0.39	0.41	0.42	0.96	0.96	0.96	0.88	0.89	0.89
MSE + DL	0.59	0.51	0.51	0.89	0.93	0.94	0.85	0.87	0.88
BCE+Tversky($\alpha = 0.3, \beta = 0.7$)	0.51	0.57	0.63	0.94	0.96	0.92	0.88	0.88	0.88
BCE+Tversky($\alpha = 0.7, \beta = 0.3$)	0.38	0.47	0.51	0.97	0.93	0.96	0.89	0.89	0.90
BCE + DL	0.87	0.52	0.58	0.94	0.93	0.93	0.87	0.88	0.88

Table 5.1: *Sensitivity*, *Specificity* and *Accuracy* scores for certain number of epochs.

have similar scores in negatives detection. Figures 5.18, 5.19 and 5.20 illustrate human detection tests based on a COCO image of 2014 dataset, across all trained detectors. As it is evident, the detectors recognize sufficiently the humans in the viewing scene, even being in groups or alone. According to the results, it is apparent that some detectors include more areas as positives in their final prediction, while others are more cautious about their selection. Since we use a thermal camera companied with our optical camera, in our situation we prefer positive-confident detectors (like Tversky(β=0.7), $F_\beta(\beta = 2)$ and Jaccard) as the given prediction will be postprocessed with the thermo-positive areas. However, in case that the casualties have no high-temperature, we focus on optical detector's beliefs that are strong and remain through exploration and thus command the UAV to approximate the target to furtherly examine and localize it. Figures in 5.21 show human recognition results made on random everyday images, and showcase the *Jaccard* and Tversky(β=0.7) detectors' application and characteristics.

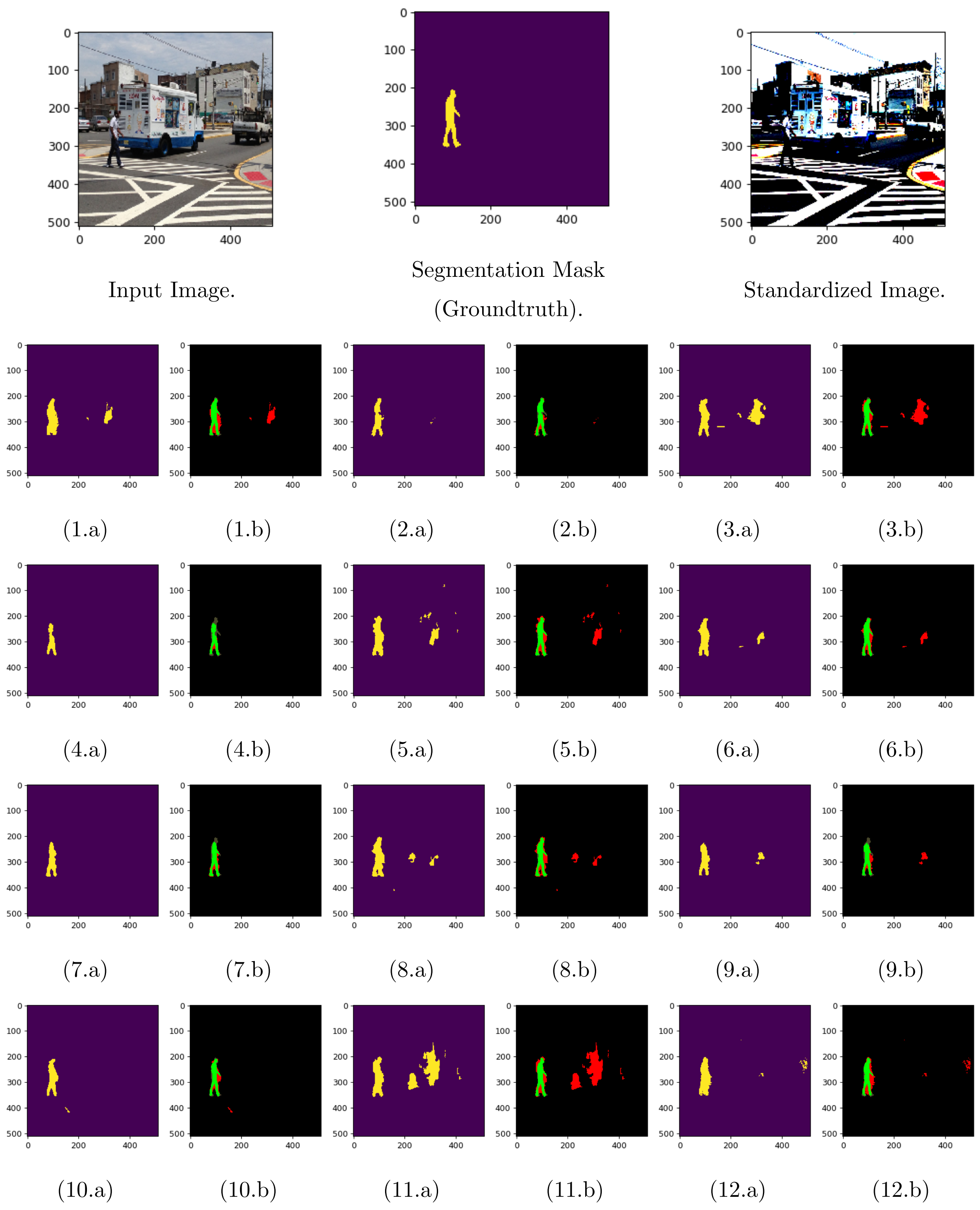


Figure 5.18: Single person detection. The predicted areas (a) and the comparison with the segmentation (groundtruth) mask (b). The comparison images (b) depict with green color the TP, with black color the TN, with red color the FN and with gray color the FP. Figures (1) : BCE , (2) : DL, (3) : IoU , (4) : BCE + Tversky($\alpha = 0.7$, $\beta = 0.3$), (5) : BCE + Tversky($\alpha = 0.3$, $\beta = 0.7$), (6) : BCE + DL, (7) : $F_\beta(\beta = 0.5)$, (8) : $F_\beta(\beta = 2)$, (9) : MSE + DL, (10) Tversky($\alpha = 0.7$, $\beta = 0.3$), (11) : Tversky($\alpha = 0.3$, $\beta = 0.7$) and (12) : MSE.

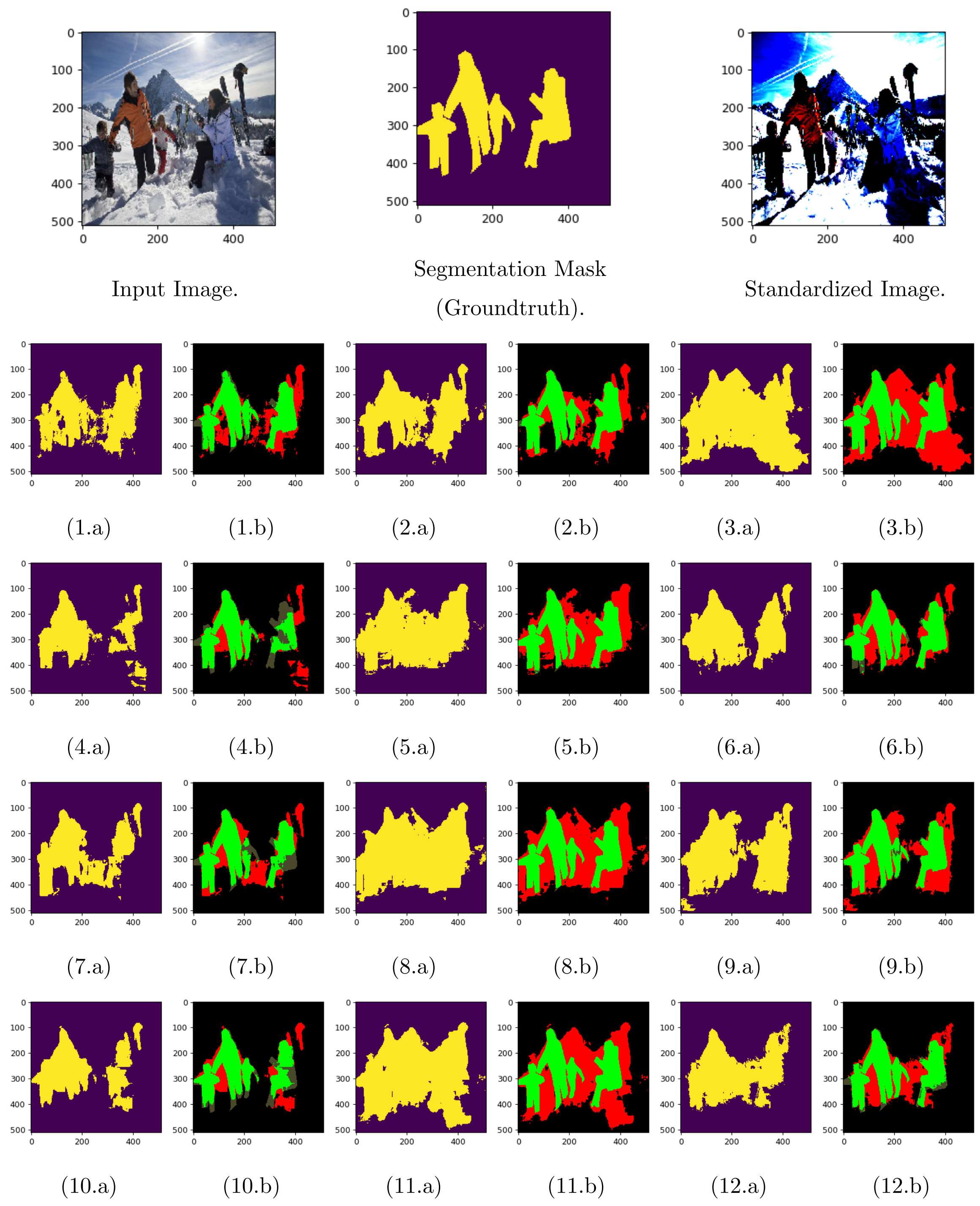


Figure 5.19: Human detection in group of people. The predicted areas (a) and the comparison with the segmentation (groundtruth) mask (b). The comparison images (b) depict with green color the TP, with black color the TN, with red color the FN and with gray color the FP. Figures (1) : BCE , (2) : DL, (3) : IoU , (4) : BCE + Tversky($\alpha = 0.7$, $\beta = 0.3$), (5) : BCE + Tversky($\alpha = 0.3$, $\beta = 0.7$), (6) : BCE + DL, (7) : $F_\beta(\beta = 0.5)$, (8) : $F_\beta(\beta = 2)$, (9) : MSE + DL, (10) Tversky($\alpha = 0.7$, $\beta = 0.3$), (11) : Tversky($\alpha = 0.3$, $\beta = 0.7$) and (12) : MSE.

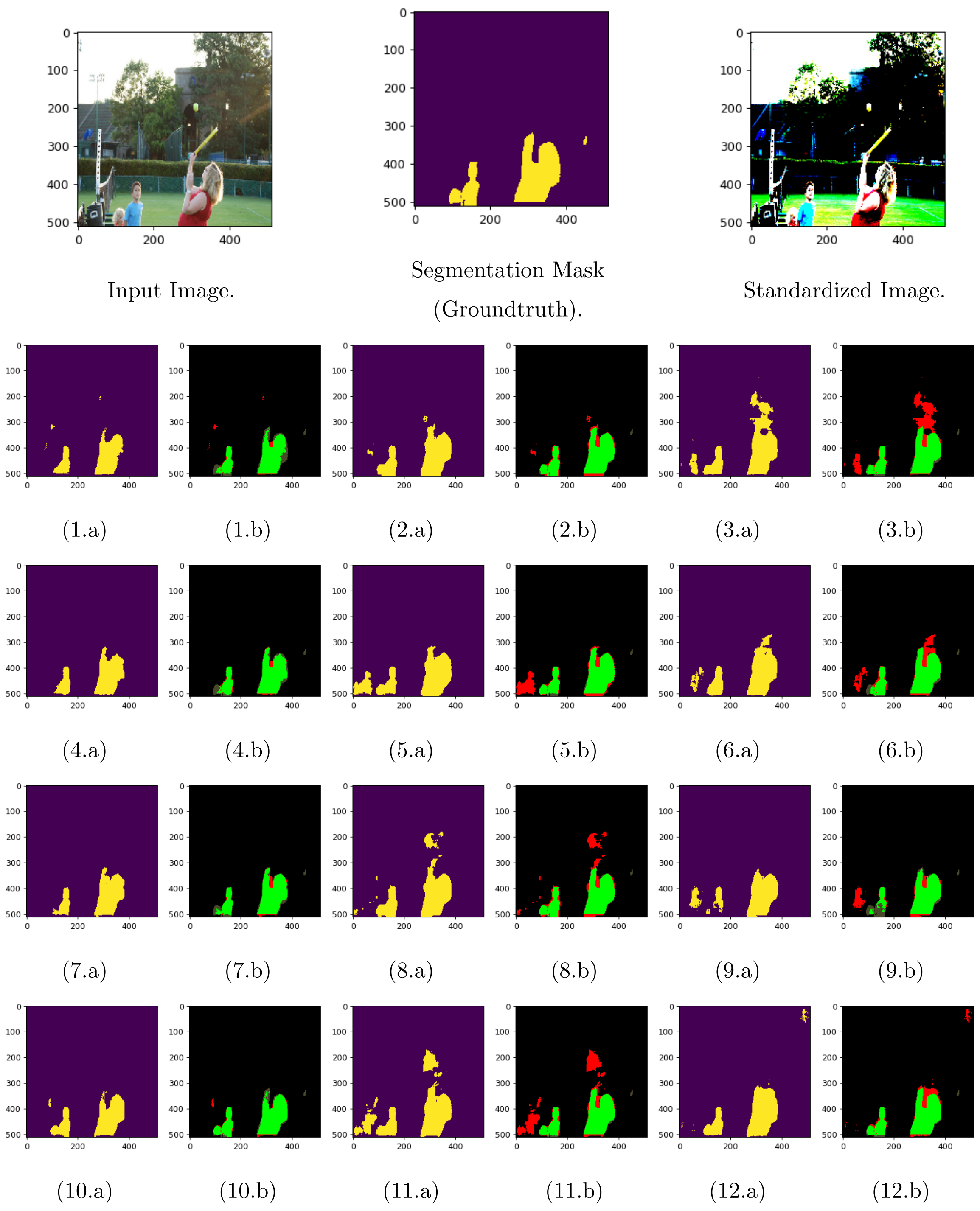


Figure 5.20: Human detection in a tennis scene. The predicted areas (a) and the comparison with the segmentation (groundtruth) mask (b). The comparison images (b) depict with green color the TP, with black color the TN, with red color the FN and with gray color the FP. Figures (1) : BCE , (2) : DL, (3) : IoU , (4) : BCE + Tversky($\alpha = 0.7$, $\beta = 0.3$), (5) : BCE + Tversky($\alpha = 0.3$, $\beta = 0.7$), (6) : BCE + DL, (7) : $F_\beta(\beta = 0.5)$, (8) : $F_\beta(\beta = 2)$, (9) : MSE + DL, (10) Tversky($\alpha = 0.7$, $\beta = 0.3$), (11) : Tversky($\alpha = 0.3$, $\beta = 0.7$) and (12) : MSE.

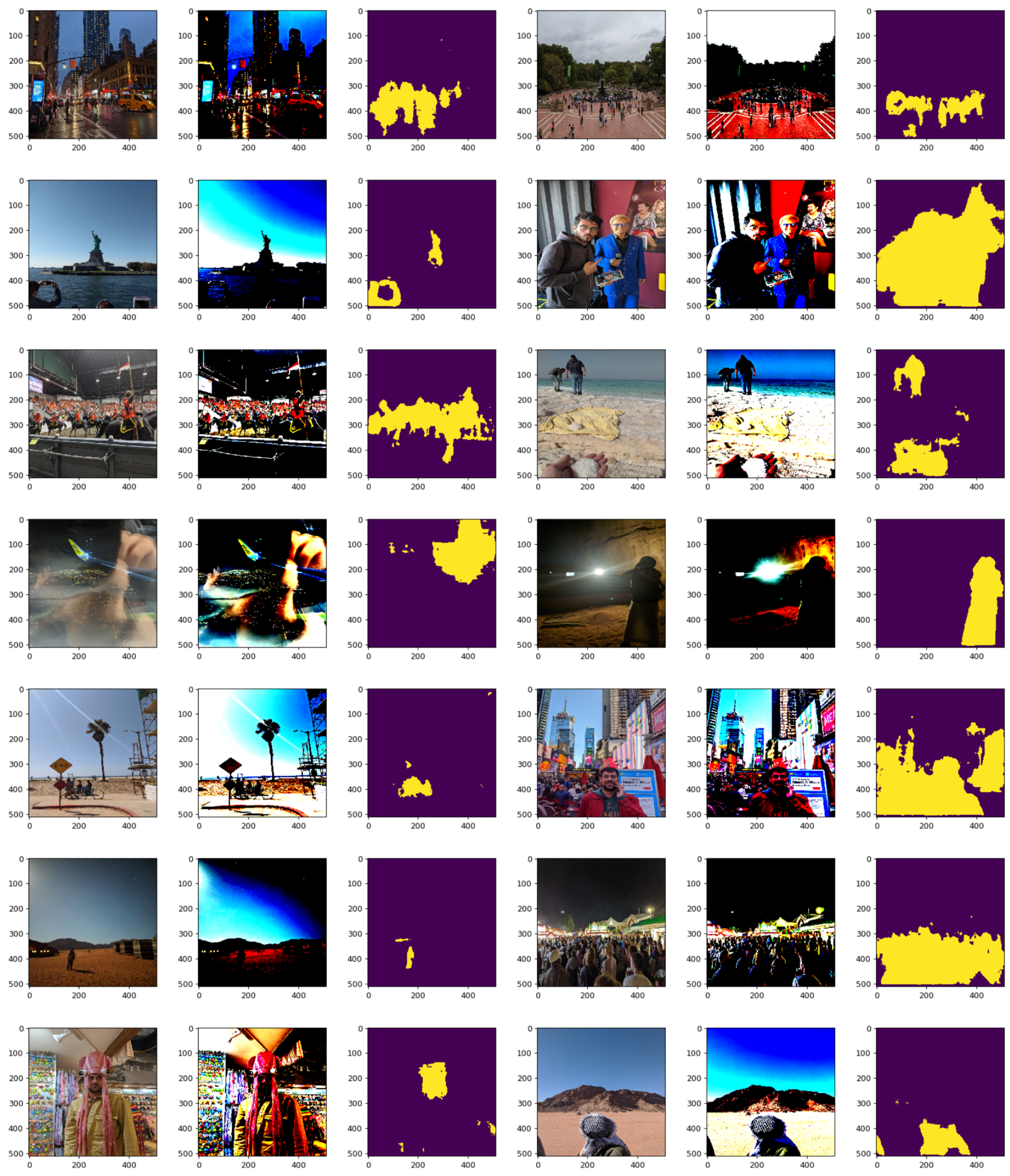


Figure 5.21: Human detection experiments under various scene contexts and conditions.

Loss Function\Epoch #	**5**	**12**	**18**
MSE	0	0	0
BCE	0.32	0.59	0.58
DL	0.60	0.75	0.78
IoU	0.31	0.75	0.70
Tversky($\alpha = 0.3, \beta = 0.7$)	0.46	0.77	0.83
Tversky($\alpha = 0.7, \beta = 0.3$)	0.42	0.58	0.63
Fβ($\beta = 2$)	0.63	0.87	0.85
Fβ($\beta = 0.5$)	0.57	0.71	0.69
MSE + DL	0.75	0.78	0.80
BCE+Tversky($\alpha = 0.3, \beta = 0.7$)	0.70	0.81	0.83
BCE+Tversky($\alpha = 0.7, \beta = 0.3$)	0.51	0.73	0.84
BCE + DL	0.51	0.72	0.72

Table 5.2: *Sensitivity* scores for certain number of epochs.

5.2.2 Training on Gazebo Dataset and Results

In the same way with the COCO dataset, we train our model based on the generated Gazebo dataset, to be able to locate virtual people within the simulator and proceed with their rescue. In specific, we created a person dataset, as described in Section 4.3.1.5, and performed a similar training approach with the COCO dataset, to conclude with the most appropriate and accurate model for our scenario. A vital factor in this part is that this detector has to perform in real-time, as it will run onboard on the simulated UAV, to be able to correlate its results with the rest of captured data (thermography positives, mapping procedure, etc.) and decide about the subsequent actions.

In particular, we generated a 5300 imagery dataset by flying the UAV in various Gazebo worlds and excluded 500 images for the testing procedures. Besides, we extracted the overall mean and variance of the current dataset and stored it for the training and the real-time detection procedures of the detector. Similarly, as with the COCO detector, we trained over all the created loss functions and stored their training and testing performances to evaluate their behavior efficiency.

Figures in 5.22 illustrate the loss function outcomes during training and testing procedures. First, we observe that despite the different formats of the Gazebo dataset as it contains virtual depictions of humans, the loss functions affect the model's behavior, analogously with the COCO training. In specific, the Tversky-based models focus separately on the details of FP and FN according to their parameter setup, and the MSE, BCE, and Dice loss present more moderate results.

As mentioned before, the behavior of the detectors is similar in the Gazebo generated dataset, even though the vast majority of the images are taken from farther distances and depict only one human in their frame. Nonetheless, few detectors seem to be affected by the form of this dataset. In specific, with the current dataset, the MSE detector indicated zero values in the $Sensitivity$ and one in $Specificity$, while the MSE loss was decreasing, due to the limited number of positives in the training images. However, most trained models performed sufficiently as they seize the available information of the dataset, to perform human detection within the Gazebo environment. Figures in 5.23 and the Table 5.2 illustrate predictions and the metric scores for the different trained models, as they were evaluated on the testing data.

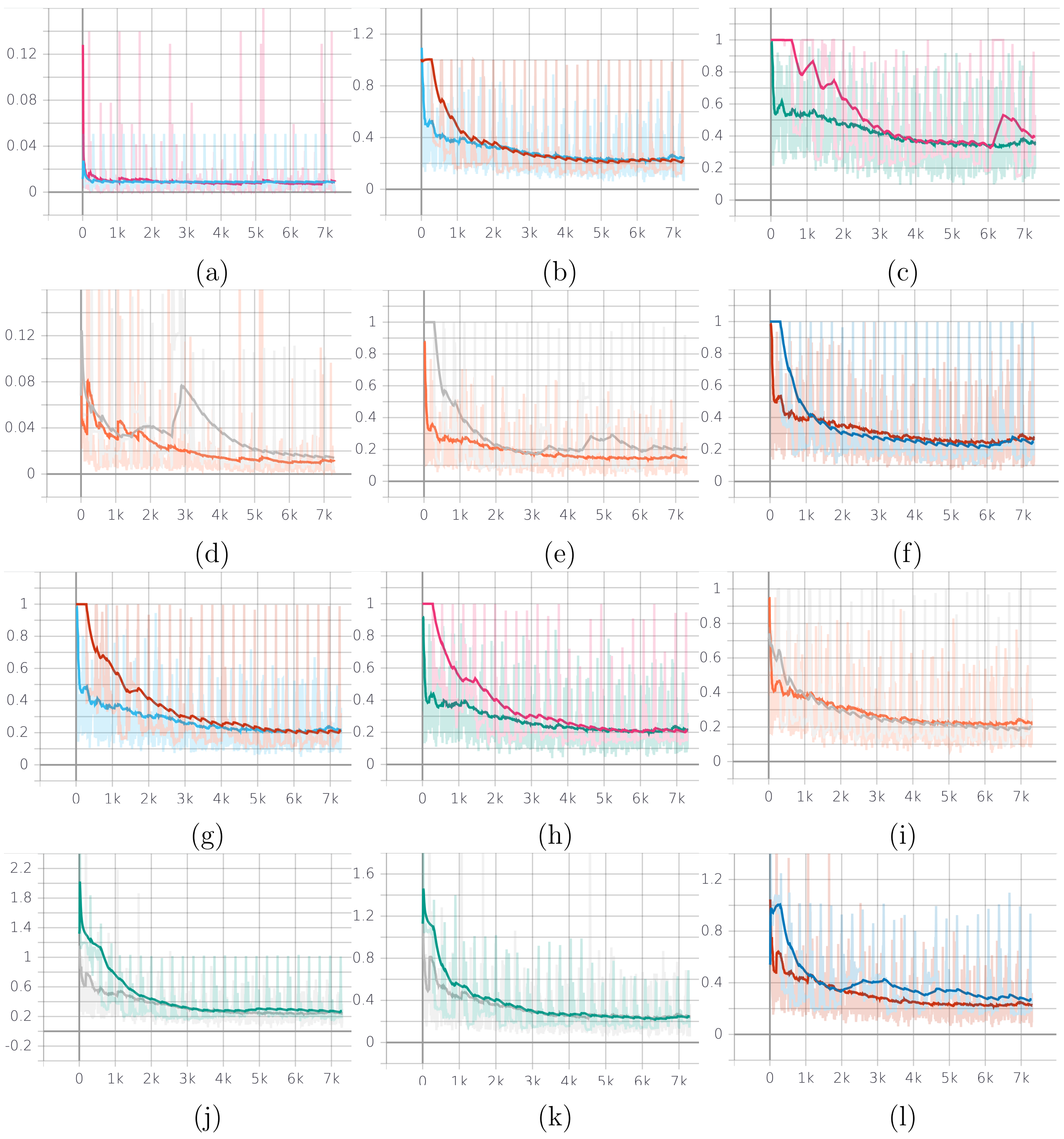


Figure 5.22: Loss function values during training and testing procedures for Gazebo Dataset (25 epochs). In each figure, the fluctuating lines depict the loss values during the training process and the smooth lines during the testing. Figures (a) : MSE, (b) : MSE + DL, (c) : IoU, (d) : BCE , (e) : $F_\beta(\beta = 2)$, (f) : $F_\beta(\beta = 0.5)$, (g) Tversky($\alpha = 0.7$, $\beta = 0.3$), (h) : Tversky($\alpha = 0.3$, $\beta = 0.7$), (i) : DL, (j) : BCE + DL, (k) : BCE + Tversky($\alpha = 0.7$, $\beta = 0.3$) and (l) : BCE + Tversky($\alpha = 0.3$, $\beta = 0.7$).

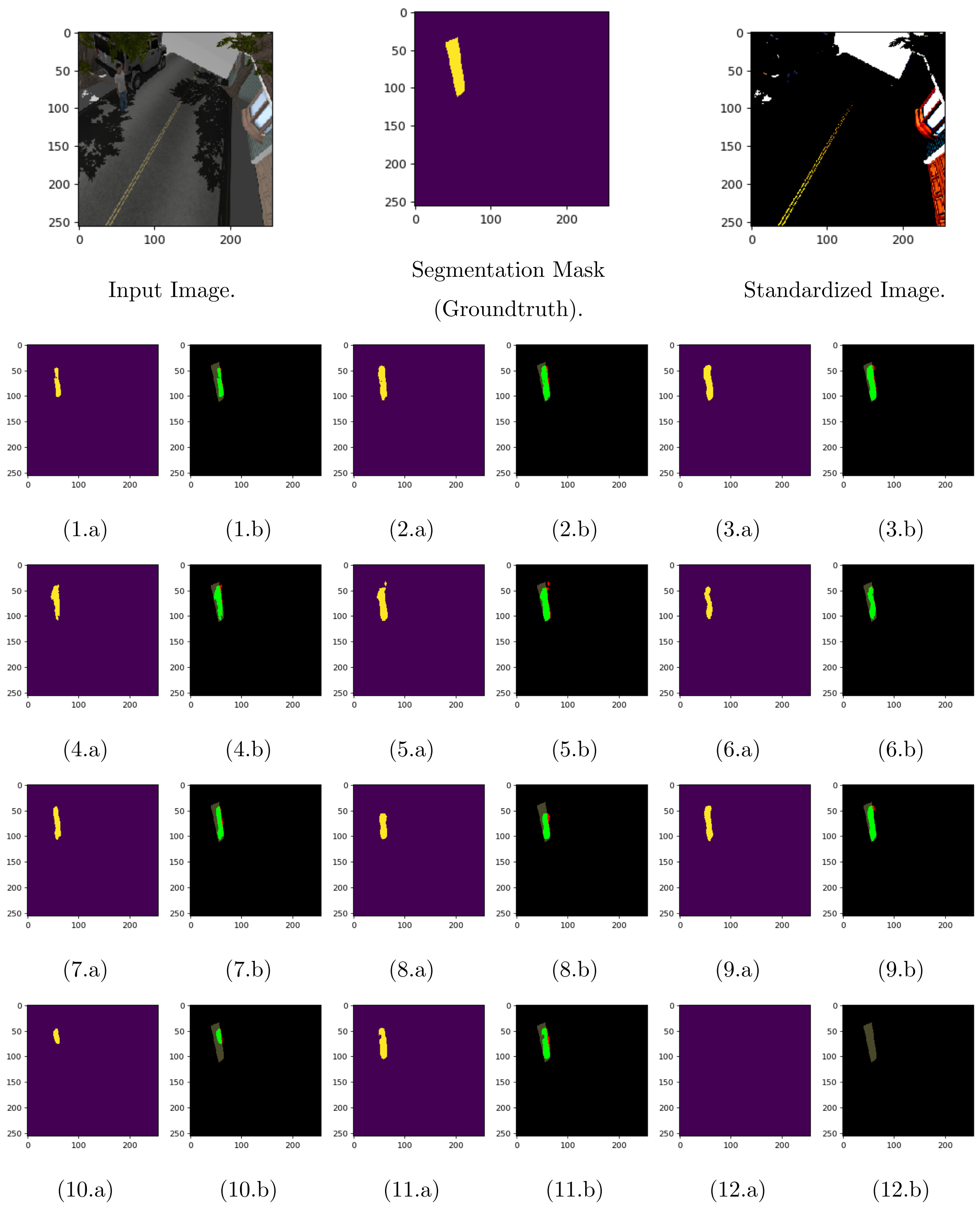


Figure 5.23: Single person detection in the Gazebo world. The predicted areas (a) and the comparison with the segmentation (groundtruth) mask (b). The comparison images (b) depict with green color the TP, with black color the TN, with red color the FN and with gray color the FP. Figures (1) : BCE , (2) : DL, (3) : IoU , (4) : BCE + Tversky($\alpha = 0.7$, $\beta = 0.3$), (5) : BCE + Tversky($\alpha = 0.3$, $\beta = 0.7$), (6) : BCE + DL, (7) : $F_\beta(\beta = 0.5)$, (8) : $F_\beta(\beta = 2)$, (9) : MSE + DL, (10) Tversky($\alpha = 0.7$, $\beta = 0.3$), (11) : Tversky($\alpha = 0.3$, $\beta = 0.7$) and (12) : MSE.

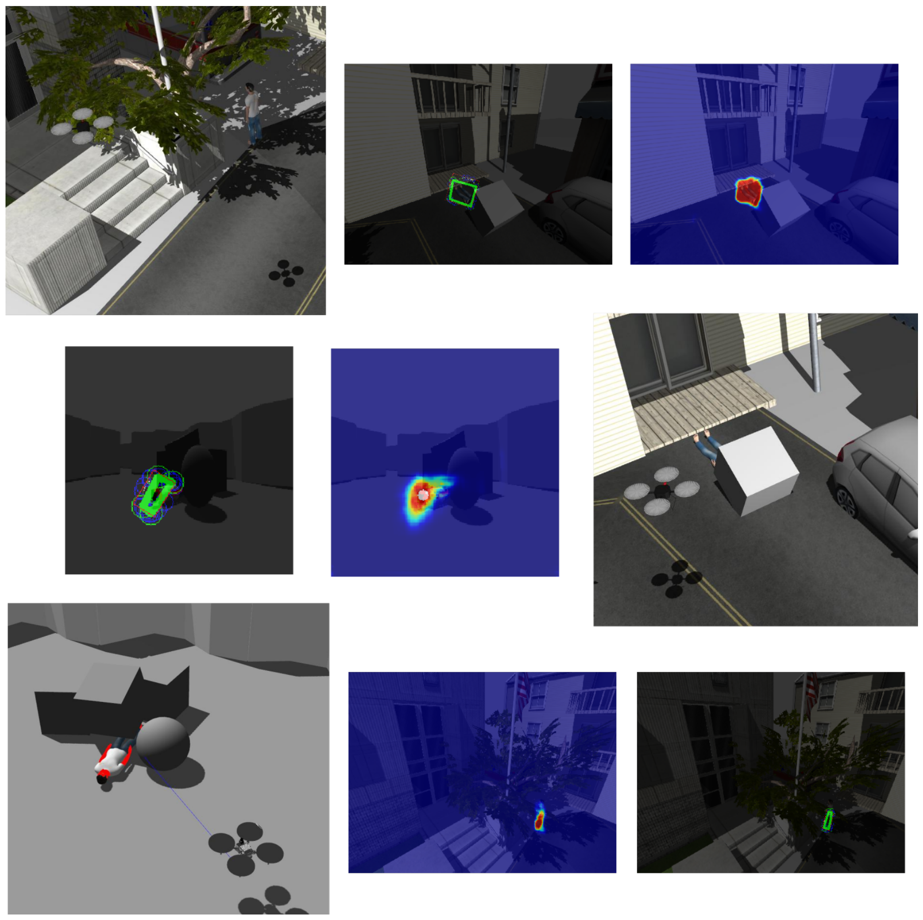


Figure 5.24: Human detection based on the simulated UAV's optical data.

As soon as the training and evaluation procedures have been completed, we selected the Tversky($\alpha = 0.3$, $\beta = 0.7$) Gazebo trained model and we built a CvBridge node to process online the optical feed of the simulated camera with this detector. Due to the downscaling of the captured image, namely from $720p$ to 256×256 resolution, the onboard detector achieves approximately 22 processed frames per second, tested on a $1080GTX\ Ti$. Although, since nowadays UAVs do not carry such powerful embedded GPUs, the detector can be adjusted and minimized in size to achieve similar processing timings. Every prediction made from the optical detector is timestamped accordingly to the robot's clock, and thus it can be correlated with the corresponding thermography image which is captured by the thermal camera. Thence, as we applied the trained detector on the UAV, we performed numerous tests in various Gazebo environments to evaluate its behavior. Figures in 5.24 depict the onboard detectors' ability to locate casualties within the simulator world.

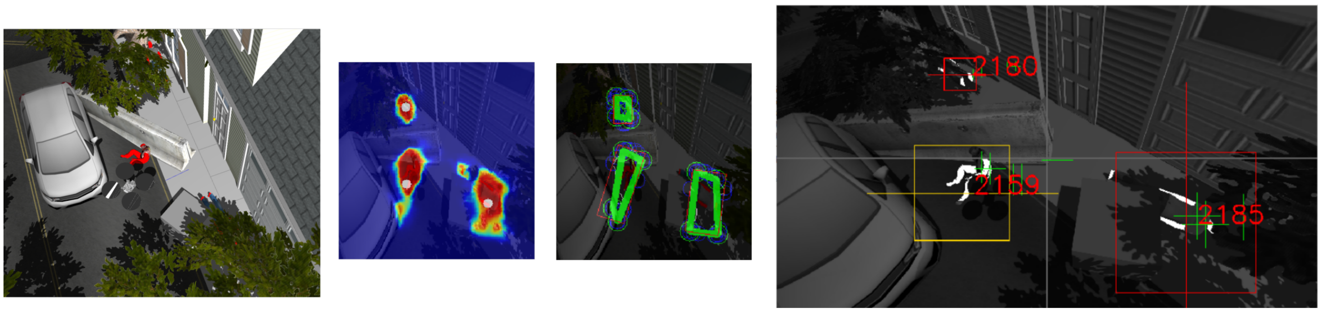


Figure 5.25: Multi-target detection and prioritization, in obstructed area.

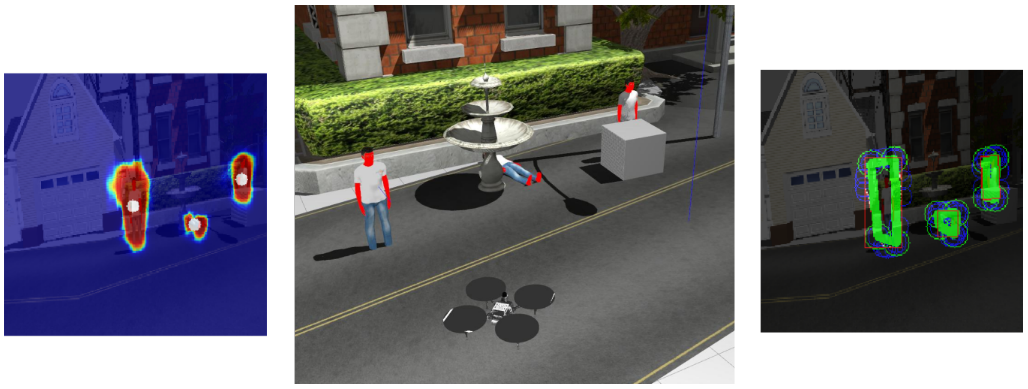


Figure 5.26: Multi-target detection and prioritization, in open area.

A noteworthy fact is that the detector gained the ability to recognize multiple humans within a single frame, although it has been trained on frames that depict a single person. Figures 5.25 and 5.26 illustrate a situation in which the UAV has detected three individuals, in a created occurrence. The information acquired by the optical detector is described in respect of the custom *PositiveAreas* message structure, which contains the capture moment timestamp, the areas' moment coordinates, and the occupation area polygons on the thermal image plane.

5.3 Navigation and Reconnaissance Experiments in Unknown Gazebo Worlds

First, as we have evaluated and concluded with our trained agent's behavior, we proceed with experiments in the Gazebo world. As mentioned in Section 5.1.2, each time a training procedure is completed, the actor and critic neural networks weights are stored locally, to be available for deployment on the custom simulated UAV of the Gazebo. Henceforth, we generated Gazebo worlds from auto-created OpenAI Gym environments, under various specifications and additions, and we proceed with the agent's behavior evaluation. It is noteworthy that in these tests we did not include any thermal-positive or human-related objects, to strictly focus on the UAV's navigation and exploration behavior.

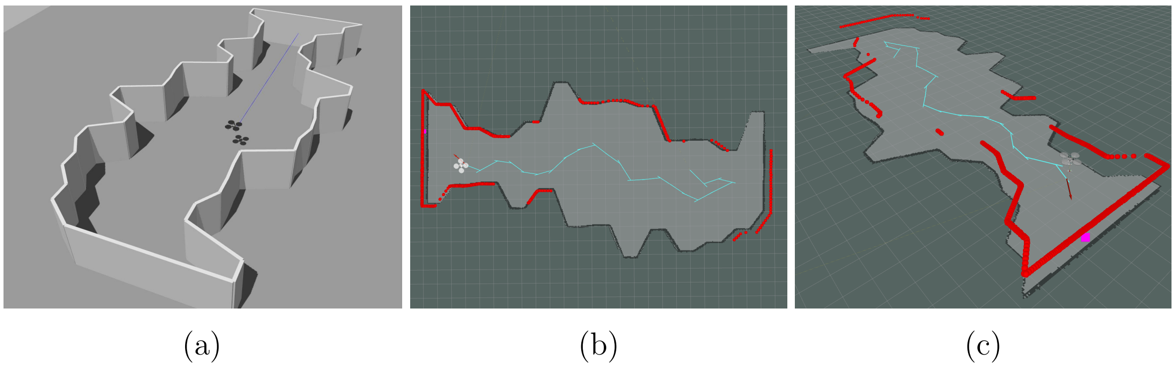

(a) (b) (c)

Figure 5.27: Evaluating the UAV's navigation performance in a 25-meter-long area.

As Figures 5.27 depict, we generated a 25-meter in length world and we initialized the UAV pose to look towards and be close to a map's wall. As the UAV system initiated the exploration and the SLAM approach, it got soon to a dead-end of the area, as it moved forward. For this reason, the agent decided to perform a 180° rotation, to pivot towards the unknown areas that were located on the other side of the map. As the rotation has been completed, the UAV completed a movement sequence until it reached the other side of the environment. Figure 5.27-(b) illustrates the panorama view of the calculated trajectory, which shows the exact UAV movement's decision chain. In this way, the UAV preserved an approaching middle-line trajectory to keep a safe distance from the nearby walls. During this time, the UAV mapped the unknown area and completed the search, when it reached the other side of the map and covered all unexplored areas.

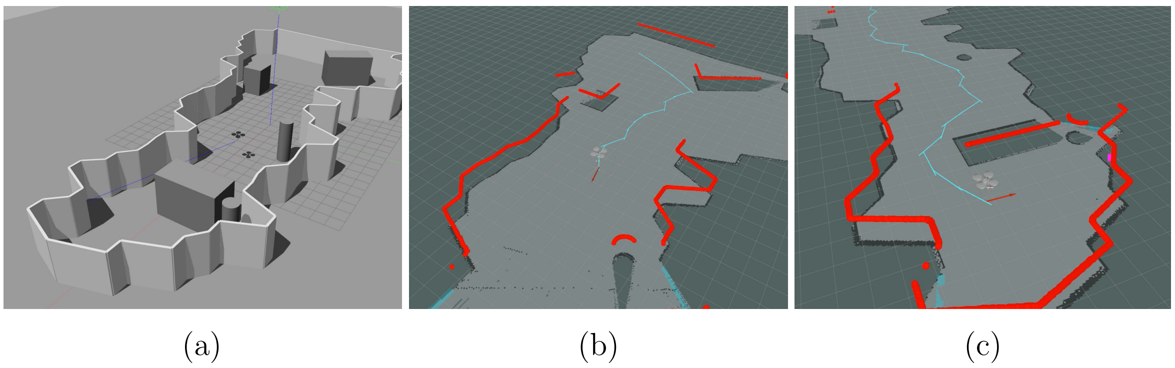


Figure 5.28: Evaluating the UAV's navigation performance in a 32-meter-long area with included obstacles.

In addition, Figure 5.28 shows an experiment made on a 32-meter-long environment, which had various obstacles placed in its area. The UAV started from a certain position and by utilizing the autonomous exploration node it achieved to navigate successfully and safely towards the end of the environment. In particular, at a specific point that is shown in Figure 5.28-(c), the UAV avoided a box object that was on its way to explore the back part of the map. The agent pivoted ideally towards the obstacle-free area and it managed to pass through the open space by maneuvering through the passage. Thenceforth, the UAV investigated the hidden and unexplored area and concluded with the complete mapping of the rescuing field.

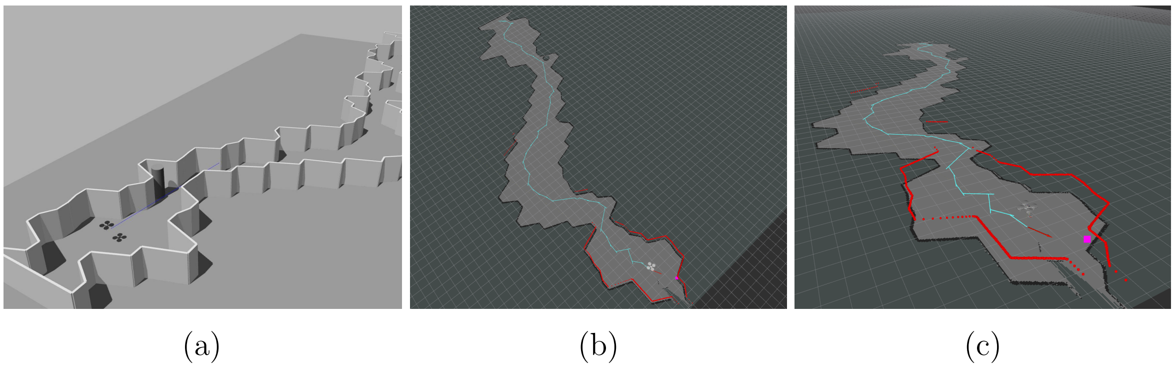


Figure 5.29: Evaluating the UAV's navigation performance in a 52-meter-long area, with the minimum narrowness value of 2 meters, and maximum wideness value of 10 meters.

To further evaluate the performance of the autonomous exploring UAV, we created a 52-meter-long environment with narrower spots and a limited width level. As depicted in Figures 5.29, despite the UAV's limited perception ability (6 distance measurements for each step), it managed to fulfill its primary objective to perform the mapping of the unknown area by achieving a safe moving operation throughout the area exploration.

Search-and-Rescue Experiment in a Cluttered Gazebo World

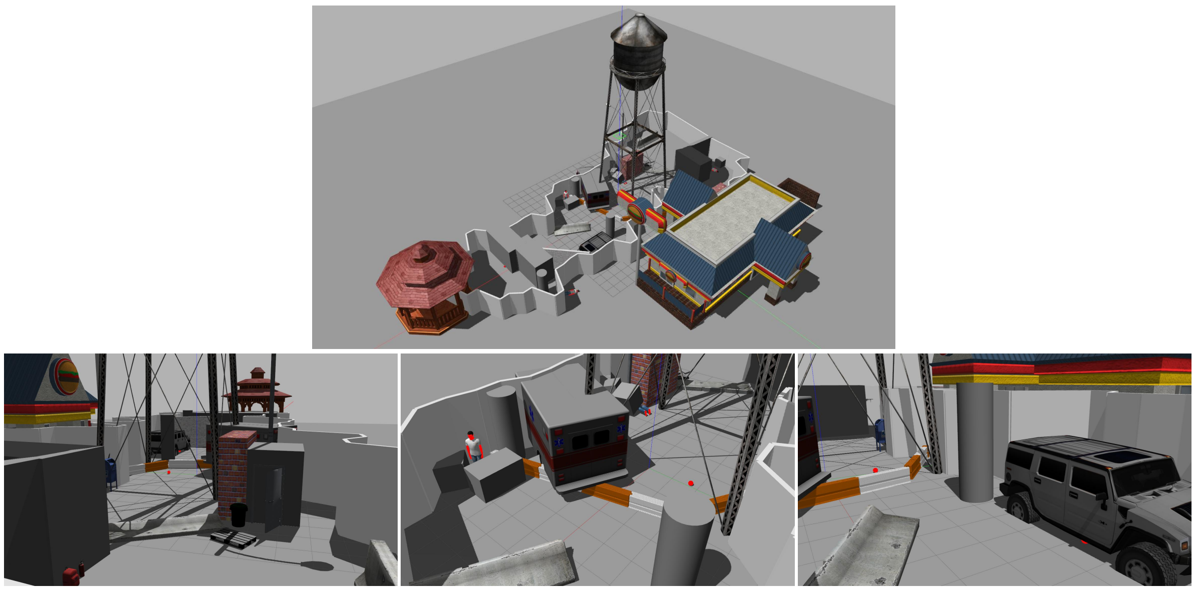


Figure 5.30: A Gazebo environment based on a random maze object enriched with various objects, and reconnaissance targets.

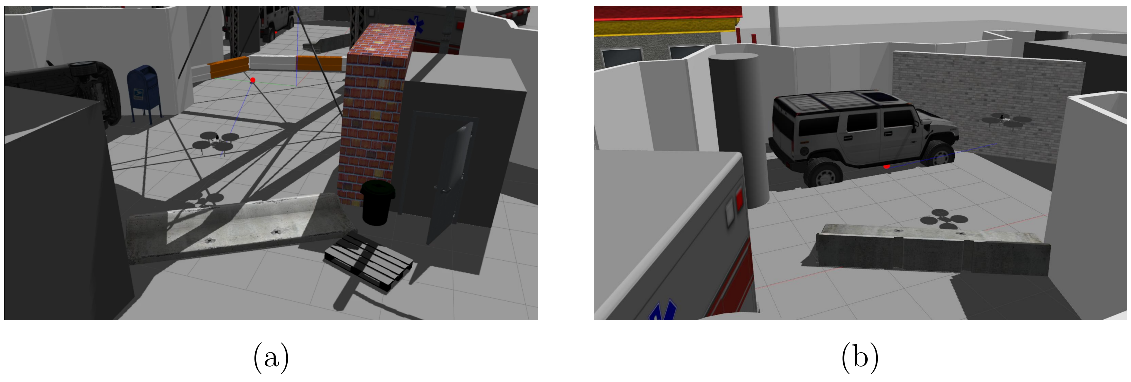


Figure 5.31: The detection of the first two targets.

In this part, we test the performance of our autonomous Search-and-Rescue UAV in a custom cluttered Gazebo environment. The basis of this environment is initiated by the generated OpenAI Gym environment, which is used for example Figure 5.28, and various objects have been added in its area along with the reconnaissance targets. The placement of the object obstacles and the walls have been made ideally to create diversity in the map's accessible areas, in order to check the UAV's navigation ability. Also, as seen in Figures 5.30 we placed both human and thermal-positive objects in different positions and covered them with other objects to make the detection procedure more complex. In addition, colored environment objects have been utilized to evaluate the color camera's detector robustness.

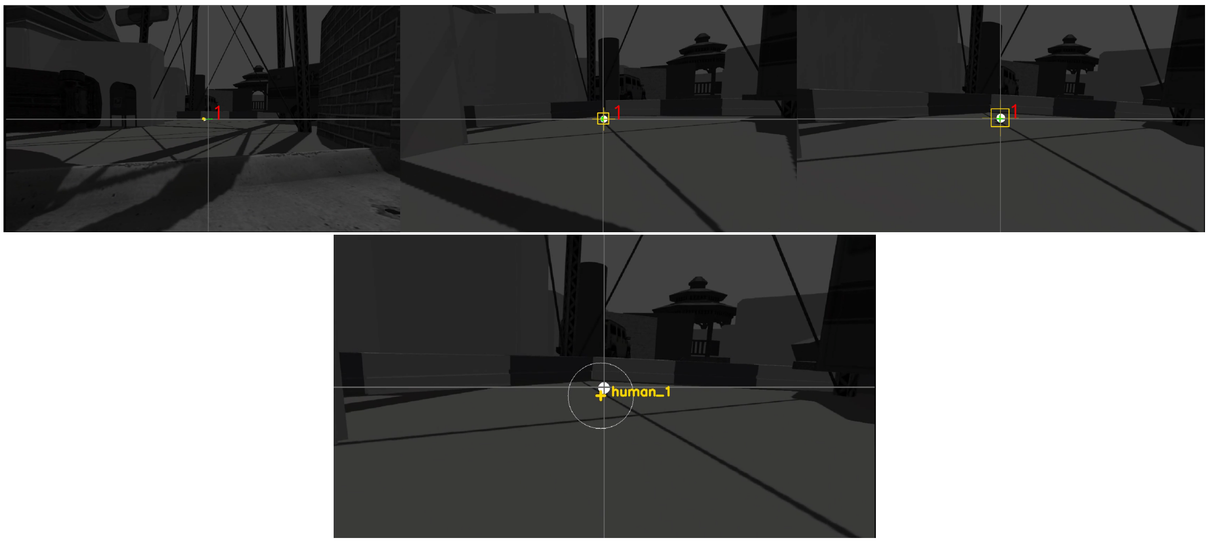


Figure 5.32: Camera captures during the positioning of the first viewed target.

First, as we enable the simulated UAV, it starts directly with the autonomous mapping and target detection procedure. From the early moments of the exploration of the unknown area, the UAV located the first thermal-positive object and enabled the engaging behavior of the gimbal unit to perform its localization. Figure 5.31-(a) depicts this particular moment, as appeared in the Gazebo. As the UAV has engaged in the positive target, it approaches it by performing movements calculated by the autonomous navigation behavior. During this approach, the UAV keeps its gimbal aiming locked on the object's body, as appeared in Figure 5.32, and performs distance measurements. By correlating the captured ranges along with its global positions, and by performing the multilateration approach with the online spatial criteria, it concludes with the object's global position. During this time, the UAV performs simultaneous localization and mapping procedures, and thus updates its pose relative to the continuously updating map of the area.

As the target's position has been approximated, which is depicted in the last snapshot of Figure 5.32, the UAV stores the object location and pins it on the rescuing map. The Figures 5.34-(a,b,c) illustrate the steps of the UAV system until it concludes with the first target's global position.

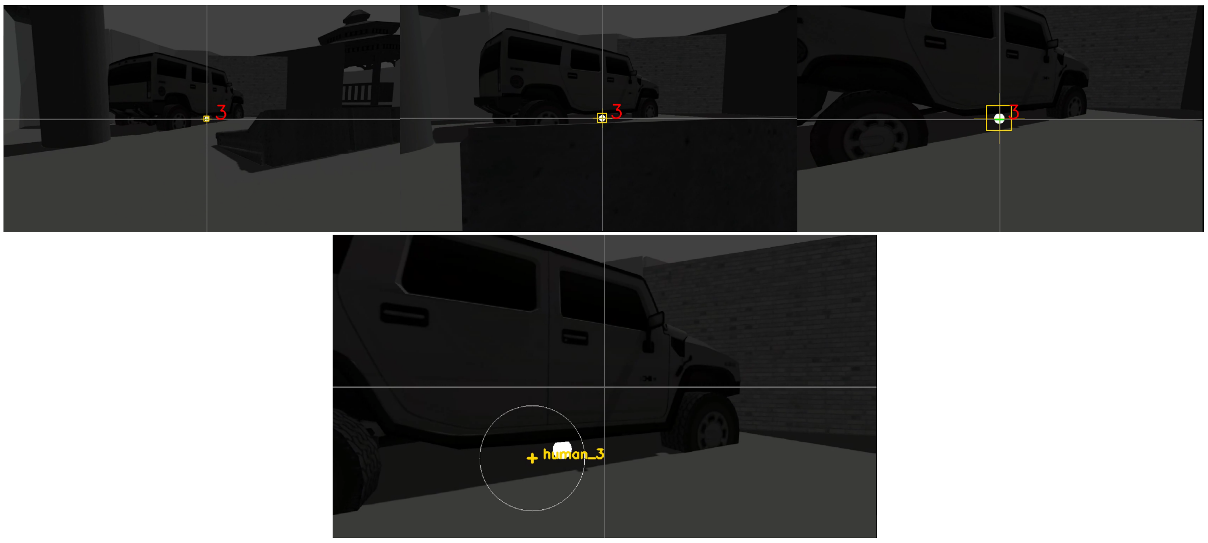


Figure 5.33: Camera captures during the positioning of the second viewed target.

Since the first target is acquired, the UAV ignores it from future detections (last Figure of 5.32) and proceeds with the coverage of the unexplored areas. The second detected target is placed under a car object within the generated environment. Figure 5.31-(b) shows a moment during the UAV's engagement on this target. The UAV performs similar approaching movements while it is engaged on the target, to extract its relative position and add it to the rescuing map. Figures 5.34-(d,e,f) depict the executed trajectory of the UAV until the extraction of the target position, which is evident in its detection frame in last Figure of 5.33. Figure 5.34-(g) shows the rescuing area map illustration after the first two targets' positioning procedures.

Furthermore, as the UAV reached in the end of the map, it pivoted towards its starting point, to face towards the unmapped areas. By knowing the previously positioned targets, the UAV excludes them from the engaging behavior, and proceeds with the area exploration. Similarly, the UAV navigates safely through the environment, while it searches for possible targets.

As the UAV utilizes the color and thermal camera's input, simultaneously, to perform

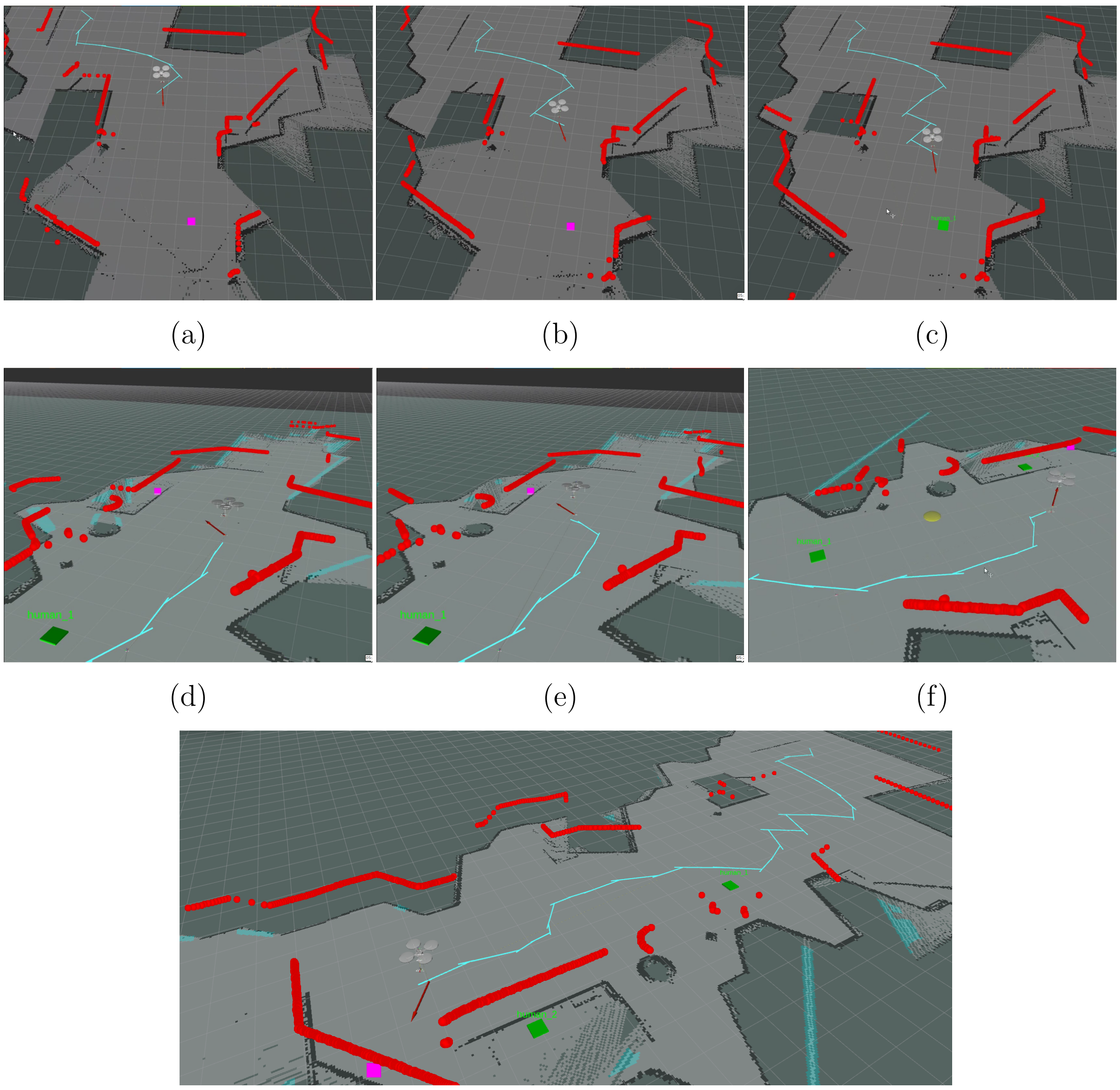


Figure 5.34: Rescuing map depiction in RViZ, during the exploration and positioning procedures of the first two targets.

human detection, it locates the next target at its left side, as appeared in Figure 5.35-(a). Even though, this object is hidden behind Gazebo objects, the UAV manages to locate and georeference it succesfully. Figures 5.36-(a,b,c) show the UAV's followed steps, until it acquired the target's global position.

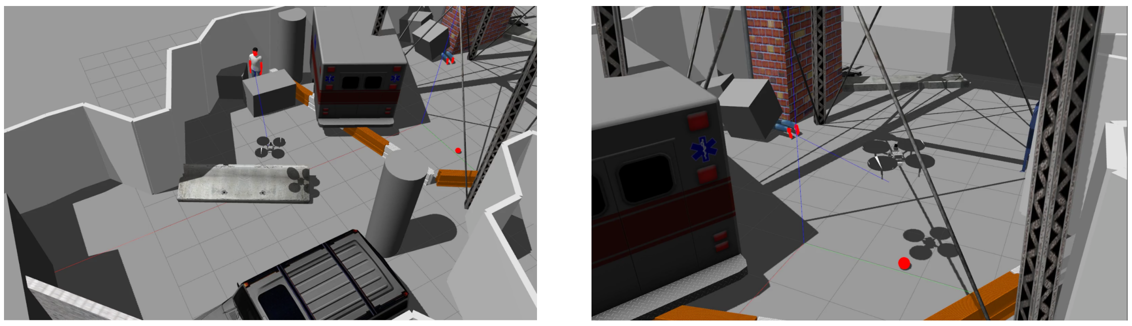

(a) (b)

Figure 5.35: The detection of the third and fourth target.

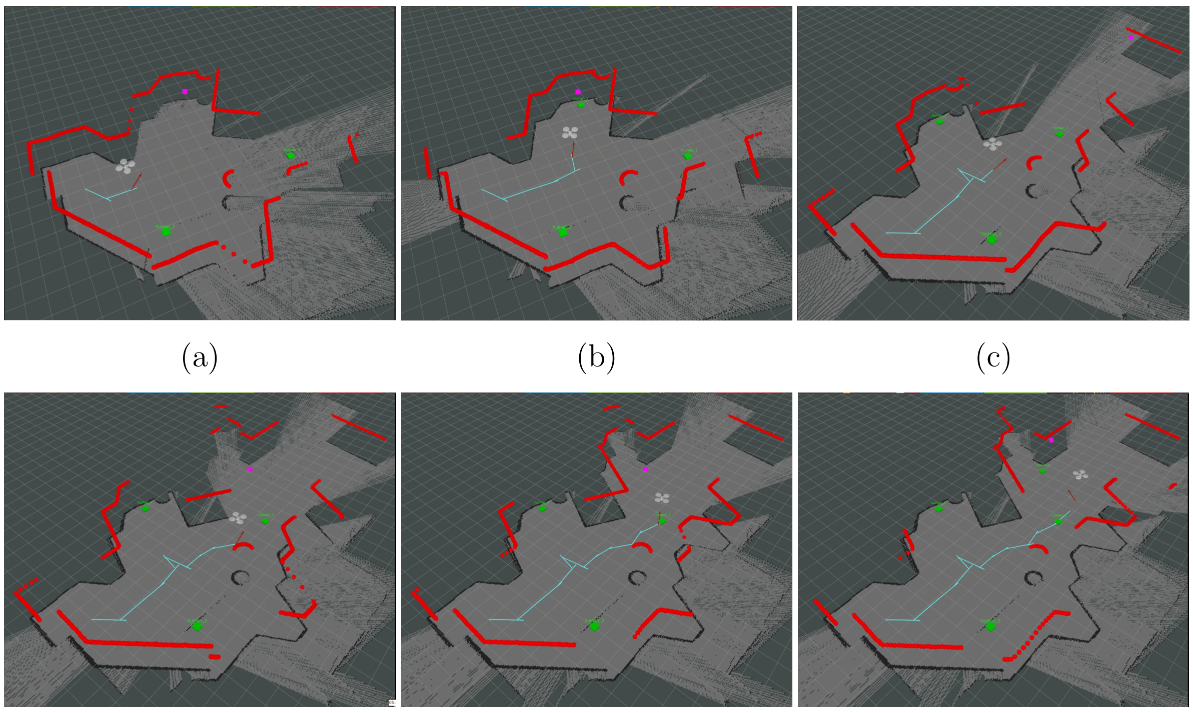

(a) (b) (c)
(d) (e) (f)

Figure 5.36: The mapping procedure of the third and fourth targets.

Also, Figures in 5.37 indicate the human detection results during the movement of the UAV, and show the final approximated and projected targets' three-dimensional positions. As the third target has been positioned, the UAV moves forward, as appeared in Figure 5.36-(d,e,f). During its upcoming steps, it is noteworthy to underline the successful manuevring movements of the UAV, in order to pass through these narrow and obstructed points of the map safely. Eventually, the UAV recognized the fourth target, which can be depicted in Figure 5.35-(b). Although this target was also covered with Gazebo objects, the UAV managed to locate it and obtain its global position. Figure 5.36-(f) show the final approximated position of this target, within the map and global coordinate system.

With the completion of the localization procedure of the fourth human, the UAV rotated towards the unexplored regions of the map that were placed at a corner of the environment, which has been uncovered from the start of the reconnaissance approach. In this part of the map, we have included a human object to be found by the UAV rescuer. Hence, as the UAV translated efficiently and safely around the existing wall obstacle, it located the human's head, as illustrated in Figure 5.38-(a). Thus, the UAV moved towards the target while it performed distance measurements, which were parsed by the multilateration algorithm. Figure 5.39 show the moments during the target detection and show the ability of the UAV to locate the human object, even in low-lighting conditions, and the Figure 5.38-(b) shows corresponding rescuing map status after the positioning of the fifth target.

Overall, the autonomous UAV system performs adequately in most tested Gazebo scenarios. The UAV achieves to navigate safely in the unknown area and performs target search and detection procedures effectively while performing the target positioning approach. Nonetheless, in more obstructed and obstacle-rich environments, the autonomous detection and navigation behaviors are complicated, which fact indicates that they need further training, with more than 5000 images and more testing environment specifications, to be more robust and adaptive in such situations. A brief video from these experiments can be found in the corresponding ROS package repository[1].

[1]This video can be found at my corresponding ROS package repository, at `https://github.com/jimcha21`. I am truly thankful to Manos Stefanakis for editing this video for me, for my thesis presentation.

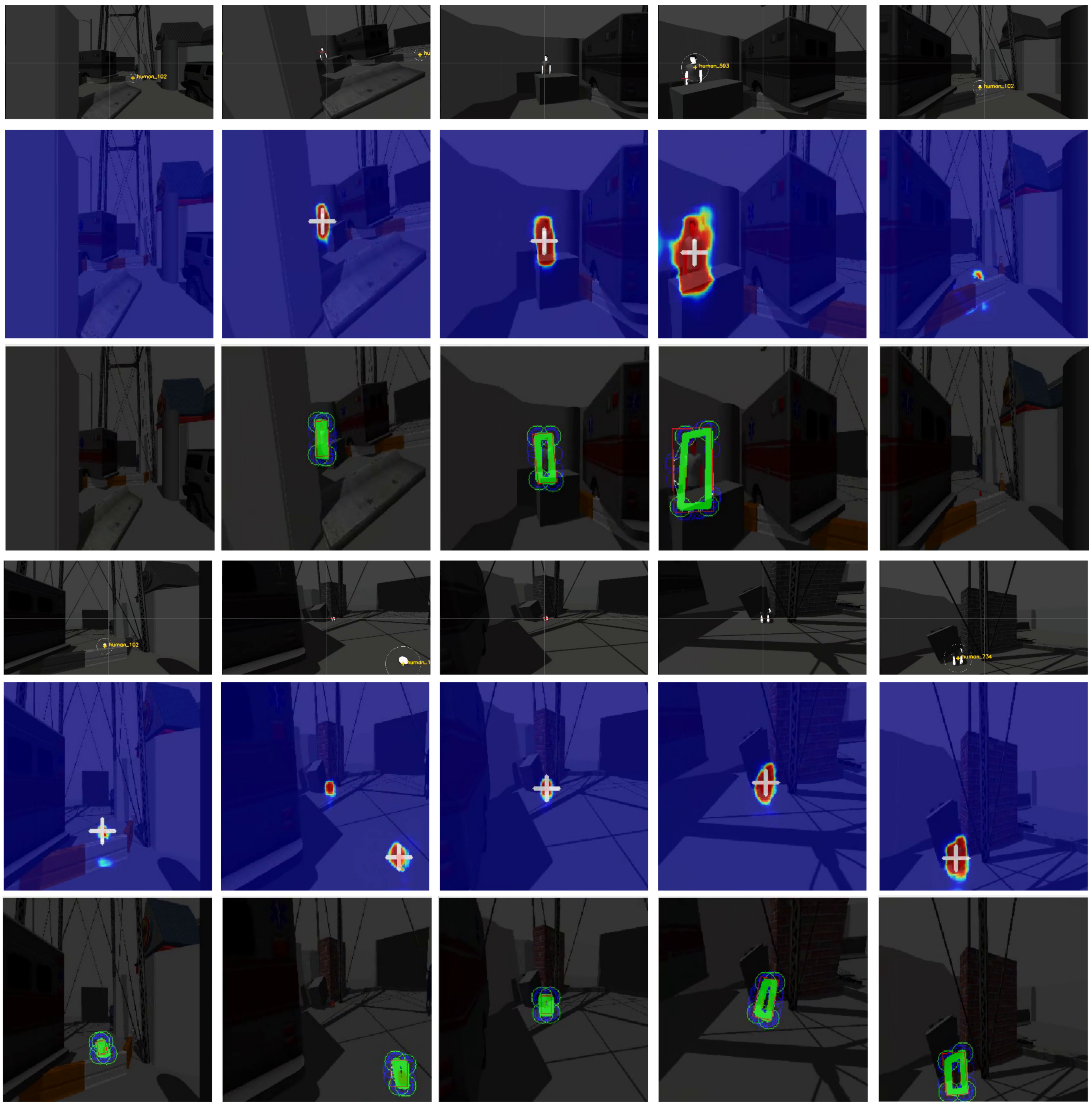


Figure 5.37: Human detection moments during the UAV's autonomous operation. The detection frames, along with predicted human regions and their final segmented forms, indicate the abitily of the UAV to locate humans within the Gazebo environment.

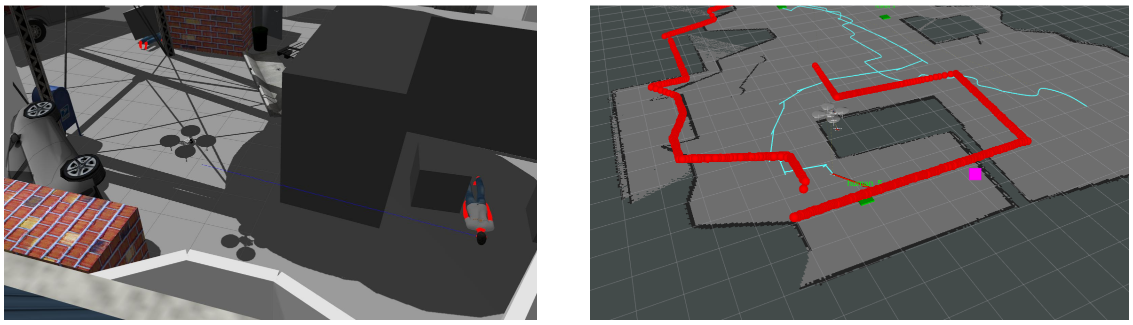


Figure 5.38: The discovery of the fifth human object.

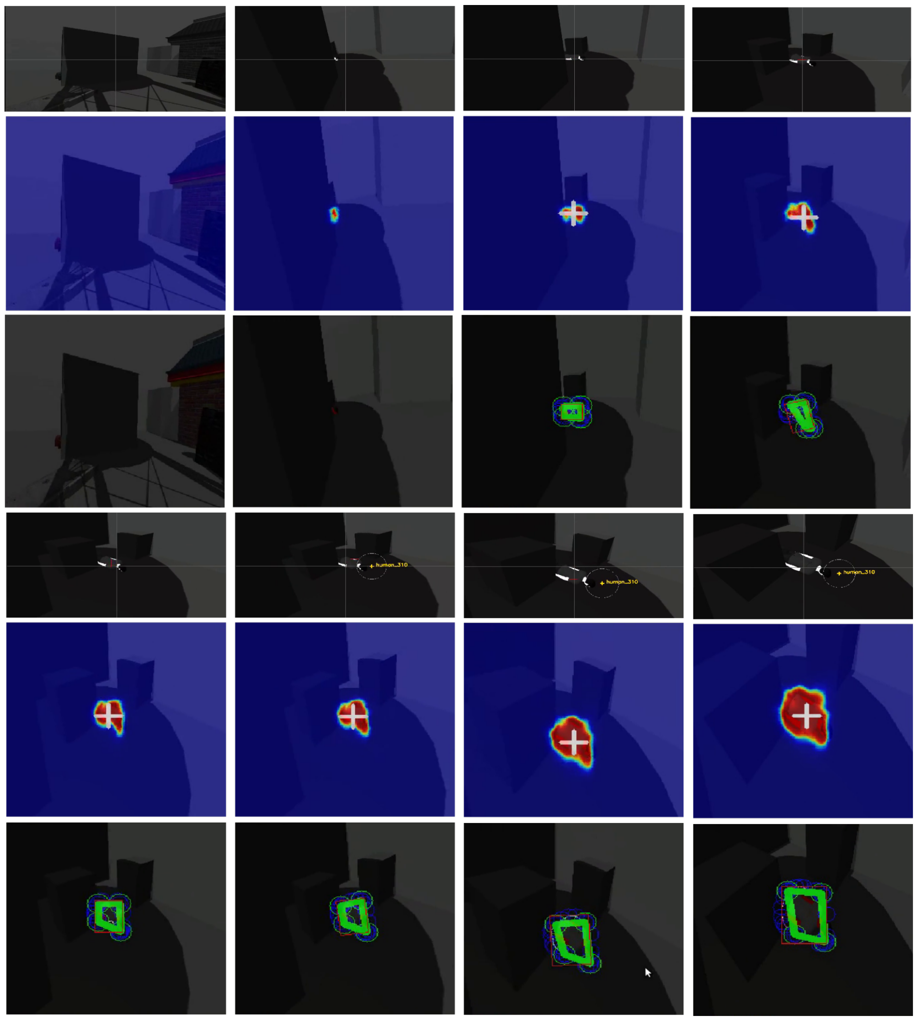


Figure 5.39: Human detection moments during the UAV's autonomous operation and the localization of the fifth target.

Chapter 6

Conclusion

6.1 Conclusion

This thesis presents a complete approach for enhancing the autonomous behavior of a UAV system, that acts in reconnaissance procedures of Search-and-Rescue scenarios. Throughout this study, we showcase applications of Machine Learning algorithms for both robotic perception and control, integrated into a more comprehensive rescuing behavior, to achieve human detection and positioning in unknown environments. In specific, assuming that the UAV has to operate alone in the emergency, we initially propose sensors that are prerequisites for this operation. In order to develop and evaluate our approach, we utilize a robotic simulator and build a simulated UAV with an attached three-dimensional gimbal unit, to perform multiple and various experiments.

As the UAV has to navigate autonomously in the unknown field, we constructed a dedicated OpenAI Gym world and performed a Deep Reinforcement Learning approach to acquire the most suitable agent's behavior. Afterwards, as one of the primary goals is to locate objects of interest in the field, and especially individuals that need help, we present a custom neural network detector to perform pixel-wise human detection. This detector is evaluated in a real-world imagery dataset and therefore it is trained on a custom dataset that has been created from flights within the Gazebo environment. The outcome of this prediction is post-processed and fused with the thermography captures of an onboard thermal camera, to further evaluate the viewed findings.

Thenceforth, by having this information the UAV utilizes its onboard gimbal system to engage on the detected target and simultaneously performs distance measurements. In this way, the UAV georeferences each captured distance with respect to its current global position and approximates the target three-dimensional position by using a multilateration approach. During the UAV's range capturing procedure, we apply real-time spatial criteria to ensure the convergence of the non-linear multilateration approach. As the target position has been acquired, the UAV stores and pinpoints the approximated position on the generated rescuing map, and proceeds with the exploration of the unknown environment.

6.2 Future Work

6.2.1 Evaluation in Real-World Scenarios

As mentioned throughout this thesis, this is a complete ROS package which is built under the standard ROS Enhancement Proposals (REP), and especially the Informational REPs of 144, 103, and 105, which can be installed on any simulated or real UAV system that meets the required technical specifications. For this reason, since this package has been tested and evaluated in a simulation environment, a vital extension could be its testing in real-world scenarios. Specifically, across this thesis we reference situations in which this package could perform successfully in real SAR scenarios, such as the evaluation of the human detection under the COCO dataset, the sensors' noise inclusions in the Gazebo environment, and the package compatibility with ROS-enabled UAVs, which all portend its capability of using it in real-world situations and systems.

6.2.2 Multi-Agent Collaboration

Additionally, a noteworthy expansion of this work could be the inclusion of a multi-agent approach to solve the common SAR scenario. Admittedly, by deploying more than one UAV system in the field, the first-responder teams can minimize the reconnaissance timings drastically, as the rescuing robots can cooperate in the exploration of the unknown area and can find casualties and obtain vital information quicker. Also, a combined localization approach can be implemented in which positioning data of a single target can be shared across the operating UAVs and approximate the victims' coordinates jointly.

For this sight, we purposely developed this software package to run under a predefined ROS namespace to support independent execution in the ROS environment, without the intervention of the execution of its cloned instances from the other UAV systems which will be in the same network.

References

[1] Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software. (2009) 7

[2] Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566). Volume 3. (Sep. 2004) 2149–2154 vol.3 8

[3] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR **abs/1502.03167** (2015) 16, 25

[4] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.A.: Playing atari with deep reinforcement learning. CoRR **abs/1312.5602** (2013) 19

[5] Walker, M.E., Hedayati, H., Szafir, D.: Robot teleoperation with augmented reality virtual surrogates. In: 2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI), IEEE (2019) 202–210 24

[6] Roberts, E.G.: Single-Task and Dual-Task Mental Workload Analysis of Unmanned Ground Vehicle Teleoperation in a Search and Rescue Scenario. PhD thesis, University of South Dakota (2020) 24

[7] Bechar, A., Vigneault, C.: Agricultural robots for field operations. part 2: Operations and systems. Biosystems Engineering **153** (2017) 110 – 128 24

[8] Isop, W.A., Gebhardt, C., Nägeli, T., Fraundorfer, F., Hilliges, O., Schmalstieg, D.: High-level teleoperation system for aerial exploration of indoor environments. Frontiers in Robotics and AI **6** (2019) 24

[9] Hanna, D., Ferworn, A.: A uav-based algorithm to assist ground sar teams in finding lost persons living with dementia. In: 2020 IEEE/ION Position, Location and Navigation Symposium (PLANS). (2020) 27–35 24

[10] Castaño, A.R., Romero, H., Capitán, J., Andrade, J.L., Ollero, A.: Development of a semi-autonomous aerial vehicle for sewerage inspection. In: Iberian Robotics conference, Springer (2019) 75–86 24

[11] Li, S., Feng, C., Niu, Y., Shi, L., Wu, Z., Song, H.: A fire reconnaissance robot based on slam position, thermal imaging technologies, and ar display. Sensors **19**(22) (2019) 5036 24

[12] Franchi, M., Fanelli, F., Bianchi, M., Ridolfi, A., Allotta, B.: Underwater robotics competitions: The european robotics league emergency robots experience with feelhippo auv. Frontiers in Robotics and AI **7** (2020) 3 24

[13] Kawatsuma, S., Fukushima, M., Okada, T.: Emergency response by robots to fukushima-daiichi accident: Summary and lessons learned. Industrial Robot: An International Journal **39** (08 2012) 24

[14] Khasawneh, A., Rogers, H., Bertrand, J., Madathil, K.C., Gramopadhye, A.: Human adaptation to latency in teleoperated multi-robot human-agent search and rescue teams. Automation in Construction **99** (2019) 265 – 277 24

[15] Tan, M., Le, Q.: EfficientNet: Rethinking model scaling for convolutional neural networks. In Chaudhuri, K., Salakhutdinov, R., eds.: Proceedings of the 36th International Conference on Machine Learning. Volume 97 of Proceedings of Machine Learning Research., Long Beach, California, USA, PMLR (09–15 Jun 2019) 6105–6114 25

[16] Kowsari, K., Heidarysafa, M., Brown, D.E., Meimandi, K.J., Barnes, L.E.: RMDL: random multimodel deep learning for classification. CoRR **abs/1805.01890** (2018) 25

[17] Huang, Y., Cheng, Y., Chen, D., Lee, H., Ngiam, J., Le, Q.V., Chen, Z.: Gpipe: Efficient training of giant neural networks using pipeline parallelism. CoRR **abs/1811.06965** (2018) 25

[18] Redmon, J., Divvala, S.K., Girshick, R.B., Farhadi, A.: You only look once: Unified, real-time object detection. CoRR **abs/1506.02640** (2015) 25

[19] Girshick, R.B., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. CoRR **abs/1311.2524** (2013) 25

[20] Girshick, R.B.: Fast R-CNN. CoRR **abs/1504.08083** (2015) 25

[21] Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation (2015) 25, 51

[22] Noh, H., Hong, S., Han, B.: Learning deconvolution network for semantic segmentation. CoRR **abs/1505.04366** (2015) 25

[23] Zhang, Z.: A flexible new technique for camera calibration. IEEE Transactions on Pattern Analysis and Machine Intelligence **22**(11) (Nov 2000) 1330–1334 25

[24] Sun, J., Li, B., Jiang, Y., Wen, C.y.: A camera-based target detection and positioning uav system for search and rescue (sar) purposes. Sensors **16**(11) (2016) 1778 25

[25] Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., von Stryk, O.: Comprehensive simulation of quadrotor uavs using ros and gazebo. In: SIMPAR. (2012) 29

[26] `https://hokuyo-usa.com/products/lidar-obstacle-detection/utm-30lx-ew`. 31

[27] Foote, T.: tf: The transform library. In: 2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA). (April 2013) 1–6 37

[28] Hohenwarter, M.: GeoGebra: Ein Softwaresystem für dynamische Geometrie und Algebra der Ebene. Master's thesis, Paris Lodron University, Salzburg, Austria (February 2002) (In German.). 42

[29] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning (2015) 47

[30] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T., eds.: Computer Vision – ECCV 2014, Cham, Springer International Publishing (2014) 740–755 49

[31] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations. (2015) 51

[32] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR09. (2009) 51

[33] Zeiler, M.D., Krishnan, D., Taylor, G.W., Fergus, R.: Deconvolutional networks. In: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (June 2010) 2528–2535 51

[34] Salehi, S.S.M., Erdogmus, D., Gholipour, A.: Tversky loss function for image segmentation using 3d fully convolutional deep networks (2017) 53

[35] Tversky, A.: Features of similarity. Psychological Review **84**(4) (1977) 327–352 53

[36] Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., Savarese, S.: Generalized intersection over union: A metric and a loss for bounding box regression (2019) 54

[37] Suzuki, S., Abe, K.: Topological structural analysis of digitized binary images by border following. Comput. Vis. Graph. Image Process. **30** (1985) 32–46 59

[38] Sklansky, J.: Finding the convex hull of a simple polygon. Pattern Recogn. Lett. **1**(2) (December 1982) 79–83 60

[39] Suzuki, S., be, K.: Topological structural analysis of digitized binary images by border following. Computer Vision, Graphics, and Image Processing **30**(1) (1985) 32 – 46 60

[40] Garrido-Jurado, S., Munoz-Salinas, R., Madrid-Cuevas, F., Medina-Carnicer, R.: Generation of fiducial marker dictionaries using mixed integer linear programming. Pattern Recognition **51** (10 2015) 62

[41] Garrido-Jurado, S., Munoz-Salinas, R., Madrid-Cuevas, F., Medina-Carnicer, R.: Generation of fiducial marker dictionaries using mixed integer linear programming. Pattern Recognition **51** (10 2015) 62

[42] Kohlbrecher, S., von Stryk, O., Meyer, J., Klingauf, U.: A flexible and scalable slam system with full 3d motion estimation. In: 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics. (Nov 2011) 155–160 64

[43] Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics (Intelligent Robotics and Autonomous Agents). The MIT Press (2005) 65