OpenVINO-based Optimized Object Detection and Semantic Segmentation for Indoor Mobile Robots

Master Thesis Report

Muhammad Affan

Master Thesis
Erasmus Mundus Master in
Marine and Maritime Intelligent Robotics
Universitat Jaume I

October 10, 2023

Supervisor by: Prof. Enric Cervera Mateu, PhD. & Mr. Román Navarro García

Co-funded by the
Erasmus+ Programme
of the European Union

to my teachers, family, and friends

Acknowledgments

I would like to express my deepest gratitude to all those who have contributed to the successful completion of my master's thesis.

First and foremost, I am immensely thankful to my thesis advisor, Prof. Enric Cervera Mateu, Ph.D., for his unwavering support, guidance, and invaluable insights throughout this journey. His recommendation to Robotnik Automation to consider me for this master's thesis opportunity has been crucial in shaping my learning curve.

I am also grateful to the members of my thesis committee, who will potentially review and provide thoughtful feedback and constructive criticism, thereby greatly enriching the quality of this work.

I extend my appreciation to the faculty and staff of Universitat Jaume I for providing a conducive academic environment and access to essential resources. In particular, I would like to mention Josefa Ramos (Técnico Máster MIR Universitat Jaume I) for her assistance and support in administrative matters necessary for undertaking this opportunity.

To my friends and fellow students who provided encouragement and camaraderie during the challenging times, your presence made this endeavor more enjoyable.

I want to acknowledge the unwavering support of my family, especially my parents, Abdul Rab and Rehana, for their unconditional love, encouragement, and belief in my abilities, and my siblings, Muhammad Fozan and Muhammad Azeem, for their encouragement.

Lastly, I am indebted to Robotnik Automation, especially Román Navarro García, for guiding me during the thesis and furnishing resources whenever necessary.

This journey has been a transformative experience, and I am grateful to all who have played a part in it. Your contributions have been instrumental in helping me reach this milestone, and I am truly appreciative.

I also would like to thank Sergio Barrachina Mir and José Vte. Martí Avilés for their inspiring LaTeX template for writing the Master Thesis report, which I have used as a starting point in writing this report.

Abstract

This abstract outlines the activities undertaken during a master's internship at Robotnik Automation. The overall goal was to validate Intel's tools for autonomous mobile robots for industrial inspection and surveillance. To achieve this goal, Intel Edge Insight for Autonomous Mobile Robots (Intel EI for AMR) and, in particular, Intel's Distribution of OpenVINO Toolkit, a robust open-source library for optimizing and deploying deep learning models, were studied, tested, and validated for various perception applications in the context of mobile robots, specifically, object detection and semantic segmentation.

The first objective was to work with the Intel EI framework to assess its feasibility and efficiency in supporting autonomous functionalities for Robotnik's mobile robots. This framework offers various tools and algorithms that aid in real-time decision-making, perception, and control, thereby enabling robots to navigate autonomously in dynamic environments. Ultimately, EI for AMR was deemed unfeasible for use due to its conflicting software ecosystem and dependencies compared to Robotnik's existing software stack. However, one of its frameworks, OpenVINO, was discovered to be a promising candidate for developing and deploying AI applications for mobile robots at Robotnik Automation. Therefore, OpenVINO was used to create object detection AI applications, particularly for docking station, COCO objects, fire, and smoke detection using YOLOv7. The object detection AI has been developed, tested, deployed, and is now commercially used with mobile robots at Robotnik Automation.

Additionally, the study addressed challenges related to point cloud data hindering indoor navigation, particularly during obstacle avoidance. Therefore, point cloud filtering by means of semantic segmentation of RGBD and LIDAR data was further explored, tested, and optimized. The excessive computation and requirement of re-training the model were the limiting factors for point cloud filtering.

Throughout the internship and thesis work, extensive experimentation, testing, and validation were conducted to evaluate the performance and effectiveness of the proposed automation and object detection solutions.

Contents

CHAPTER 1

INTRODUCTION

Contents

This chapter reflects on the motivation and the activities carried out during the master's internship period.

1.1 Background

Before delving into the technical details, it is important to briefly introduce the background, context of the proposed activities, and key terminologies to the reader for better understanding.

1.1.1 Robotnik Automation

This thesis activity was primarily conducted at Robotnik Automation [1], located in Paterna, Valencia, Spain. Robotnik, headquartered in Valencia, is an international mobile robotics company specializing in designing and developing mobile robots, mobile manipulators, and customized robotic solutions for industrial automation and research. Robotnik offers a range of mobile robots, including RB-Robout, RB-Theron, RB-Vogui, RB-1 Base, Summit-XL Steel, Summit-XL, RB-Car, and RB-Watcher, designed for tasks such as logistics, research, surveillance, and inspection. The RB-Watcher robot, the latest addition to Robotnik's offerings, is specifically designed for surveillance and security

(a) RB-Watcher (b) RB-Theron

Figure 1.1: Pictures of Robotnik's platforms utilized during the given master's thesis work

tasks to minimize potential hazards and risks in the deployed environment. Its software stack is adaptable, modular, and integrable with new algorithmic functional capabilities. Consequently, all proposed activities, including software development and validation, are focused specifically on the RB-Watcher robot (see Fig. 1.1a, [2]). To ensure reliability and quality assurance, the software packages are also verified using RB-Theron (see Fig. 1.1b, [3]).

1.1.2 Indoor Autonomous Mobile Robots (AMR)

A self-contained robotic system explicitly designed to navigate and carry out assigned tasks in an indoor space without direct human intervention is known as an indoor Autonomous Mobile Robot (AMR). These robots are equipped with a variety of sensors, algorithms, and control systems that enable them to function autonomously and make decisions based on their perception of the surroundings and predetermined goals. They can perform activities such as path planning, obstacle avoidance, mapping and localization, and interfacing with devices or objects in the indoor environment. The hallmark of these robots is their ability to efficiently carry out repetitive or complex tasks, thereby improving overall productivity and enhancing safety by minimizing human involvement. These robots' applications span warehouses, industries, hospitals, offices, and homes. Examples of indoor autonomous mobile robots include pick-and-place robots in warehouses, cleaning robots, surveillance robots, and more. The indoor AMR application of interest for the proposed activity was security and surveillance.

1.1.3 AMR for Security and Surveillance

These AMRs enhance the security and surveillance capabilities of the facilities in which they are deployed. These robots are equipped with advanced sensors, cameras, and detection capabilities, along with autonomous patrolling and monitoring functionalities to detect and respond to security threats or suspicious activities. The tasks may also require navigating through corridors, rooms, and other indoor spaces, utilizing algorithms for mapping, navigation, and obstacle avoidance. In general, AMRs play a vital role in enhancing security by providing real-time monitoring, triggering alarms, gathering visual data, and even interfacing with centralized security systems. In offices, warehouses, data centers, and critical infrastructure facilities, their autonomous nature reduces the need for constant human presence, minimizing the chances of human error and enabling more efficient and comprehensive surveillance, threat detection, and response.

1.1.4 Intel Edge Insights for AMRs and OpenVINO

Edge Insights for Autonomous Mobile Robots (EI for AMR, [4]) is an end-to-end mobile robot framework developed by Intel. This framework leverages advanced edge computing, data analytics, and machine learning technologies to enhance the capabilities of mobile robotic systems. It facilitates modular software development and deployment on mobile robots with the help of Docker [5] and the open-source ROS2 (Robot Operating System 2, [6]) as shown in Fig. 1.2. Key characteristics of EI for AMR include real-time data processing at the edge, data analytics for businesses to gain better insights, advanced in-built perception algorithms, real-time host-robot health monitoring, fleet management, data security with compliance, and additional customization capabilities.

Intel EI for AMR also leverages OpenVINO (Open Visual Inference and Neural Network Optimization, [7]) for optimized computer vision and deep learning implementations. OpenVINO is an advanced framework developed by Intel that optimizes and accelerates deep learning models for various Intel hardware platforms, including Central Processing Units (CPUs), Graphical Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs), and neural accelerators[1]. Through Intel's OpenVINO library, EI for AMR provides optimized deep learning model deployment on AMRs to process complex data and perform tasks such as semantic segmentation, pose estimation, object detection, tracking, sensor fusion, face recognition, etc.

1.1.5 VDA5050 and Fleet Management

Fleet management refers to controlling, monitoring, and regulating a collection of vehicles, which may include trucks, cars, buses, or Automated Guided Vehicles (AGVs). This process includes activities, such as planning travel routes, monitoring vehicle positions, overseeing fuel usage, and handling maintenance timetables. A novel and promising standard for communication between AGVs and fleet management software is known as

[1] Specialized hardware and processors that are optimized specifically to handle neural network workloads

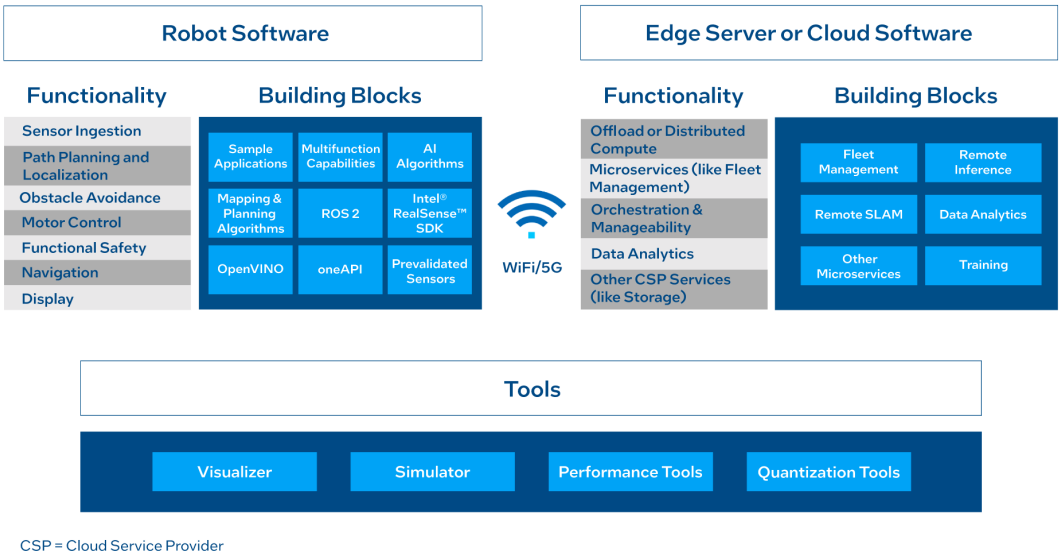


Figure 1.2: EI for AMR Architecture [4]

VDA5050. VDA is an acronym of the Verband der Automobilindustrie (VDA)[2] - the German Association of the Automotive Industry - a pivotal contributor to this technology. The other major contributors include the Mechanical Engineering Industry Association[3], the Institute for Material Flow and Logistics (IFL) at KIT[4], and several AMR industry collaborators. VDA5050 is an open-source communication protocol among different AGV fleet vehicles and central master control [8]. The VDA5050 is becoming increasingly important (especially in Europe) as it encapsulates the complex commissioning and interoperability of various AGVs (even from different manufacturers) and creates a collaborative space (a unified path) for all AGVs to operate. Considering these benefits of the VDA5050, the objective was to evaluate EI for AMR inbuilt fleet management and VDA5050 functionalities, specifically, VDA5050 to ROS2 conversion, the possibility of customized usage, and basic advanced fleet management (robot orchestration).

1.1.6 Automated Docking in the Context of AMRs

Mobile robots with the capability to automatically approach, align, and connect to a designated docking station or charging point are said to be capable of automated docking. There are various levels of autonomy in docking, enabling mobile robots to transfer data, recharge their batteries, switch their batteries, and carry out their assigned tasks without human intervention. Different sets of sensors, algorithms, and control systems are used

[2] www.vda.de
[3] www.vdma.org
[4] www.ifl.kit.edu

to enable the mobile robots to maneuver and align themselves with the docking station precisely. Automated docking ensures operational autonomy and continued operation for an extended period without human assistance in various applications such as, but not limited to, delivery, cleaning, warehousing, and surveillance.

1.1.7 Object Detection

Object detection is a computer vision task involving detecting and localizing objects of interest in an image or video frame. Unlike an image or frame classification task, it not only assigns a label to the object but also identifies the object's class, draws a bounding box around it, and assigns a confidence score for each detection. Object detection finds its application in robotics, autonomous vehicles, medical imaging, and many other domains.

Object detection algorithms have evolved over the past two decades. Initial work that appeared on object detection utilized conventional image processing and machine learning approaches. Notably, the pioneering work of Viola–Jones appeared in 2001 [9]. It combined different image processing concepts, such as Haar-like Features, Integral Images, the AdaBoost Algorithm, and the Cascade Classifier, for face detection problems. Later, HOG Detector [10] and DPM [11] gained a significant reputation.

With the inception of Convolutional Neural Network (CNN) in 2012 [12], deep learning-based object detection models began to emerge in 2014. These models are further categorized into two types: two-stage detection and single-stage detection.

1.1.7.1 Two-Stage Object Detection:

These models use a two-step process for object detection: identifying the potential locations of objects in an image, and then classifying the objects and creating bounding boxes around them. Two-stage methods appeared in the literature before one-stage methods, so two-stage detection methods are discussed first in the following paragraph. Note that the following list is not exhaustive and only includes major milestones of two-stage detectors in the order of their appearance in the academic literature.

Region with CNN features (R-CNN) [13]: It is considered a pioneering work in deep learning-based object detection. It utilizes CNN to generate a smaller set of possible regions where an object of interest could be present. Afterward, it individually extracts features of each specific region and decides whether the region contains an object or not.

Spatial Pyramid Pooling Networks (SPPNet) [14]: The key innovation of SPP-Net is the inclusion of a spatial pyramid pooling strategy to handle input images of varying sizes or dimensions.

Fast R-CNN [15]: It augmented R-CNN models with key innovations like region proposals, single forward pass over the entire image, and region of interest pooling, to name a few. Refer [16] for detailed insights.

Faster R-CNN [16]: It builds on top of Fast R-CNN [15] and introduces the concept of Region Proposal Networks (RPNs) to improve object localization. It generates region proposals and refines them for more accurate detection.

Mask R-CNN [17]: It is an extension of Faster R-CNN. It generates object masks in addition to bounding boxes, boosting accuracy on various object detection benchmarks and robustness to occlusions at the expense of a minor difference in computational speed.

Feature Pyramid Networks (FPN) [18]: It utilizes input images as a pyramid of feature maps, each having a different spatial resolution. This addresses the underlying issue of detecting smaller objects (or objects of different sizes) that faster RCNN was unable to solve.

1.1.7.2 One-Stage Object Detection

Single-stage object detection methods predict the bounding boxes and class labels of objects in an image in a single step. In the following paragraphs, single-stage object detection methods are discussed.

Single Shot Detectors (SSD): It uses a single pass through the neural network to predict object classes and bounding box coordinates. They are lightweight, efficient, and suitable for real-time applications [19].

You Only Look Once (YOLO): It takes a different approach by dividing the image into a grid and predicting object properties directly from the grid cells. It achieves real-time performance and is famous for real-time applications. There have been numerous iterations since the inception of YOLO [20], and the latest in the series is YOLOv8 [21].

1.1.8 Common Object in Context (COCO) Dataset

COCO is one of the most popular, open-source, and large-scale object detection, segmentation, and annotation datasets [22]. It is widely used for training and evaluating object detection and segmentation models with up to 80 object categories and up to 1.5 million object segmentation masks. The dataset is divided into training, validation, and testing subgroups with an approximate size of 118000, 5000, and 21000 images, respectively. Splitting the dataset into train, validation, and test sets is standard practice. The training set is a portion of data used to fit a deep learning model. A validation set is used to evaluate a deep learning model in the development phase frequently. The testing set comprises of entirely unseen data samples for the final evaluation of the model.

1.1.9 Semantic Segmentation

In computer vision, semantic segmentation refers to classifying each pixel in an image into a specific category or class, thus assigning semantic meaning to each pixel. It enables

understanding and distinguishing between different objects, backgrounds, and elements within the image. The goal is to partition an image into meaningful regions corresponding to objects or structures of interest with applications in autonomous driving, medical image diagnosis, and robotics. There are various types of segmentation, such as instance, panoptic [23], binary, multi-class, real-time/video, or panorama segmentation. The focus of this work is real-time multi-class semantic segmentation algorithms.

Starting from 2012, various semantic segmentation models based on CNNs have been introduced in consecutive years up until the present. Discussing all of them is outside the scope of this work; however, significant enhancements in architecture and milestones are summarized below.

1.1.9.1 Early Fully Convolutional Networks (FCN)

FCN was a breakthrough by introducing end-to-end learning for semantic segmentation. It replaced fully connected layers with convolutional layers, enabling pixel-wise predictions. The first time FCN-based object segmentation was presented in [24].

1.1.9.2 Dilated Convolutions

Dilated convolutions expanded the receptive field without increasing the number of parameters, allowing networks like DeepLab [25] to capture context while maintaining efficiency. Specifically, DeepLab introduced Atrous Spatial Pyramid Pooling (ASPP) to capture multi-scale context, and its successors followed the improvement with newer innovations later. FastFCN also gained notable consideration at that time as it improved the speed of DeepLab by incorporating Joint Pyramid Upsampling blocks [26].

The original DeepLab model's creators later integrated the ASPP into DeepLabv2 [27]. Subsequently, in DeepLabv3 [28], they extended this approach by introducing a cascaded deep ASPP module to incorporate multiple contextual features. Following DeepLab and aiming to enlarge the receptive field with multi-scale context incorporation, Pyramid Scene Parsing Network [29] employed a pyramid pooling module to capture global context at different scales.

1.1.9.3 Top-down/Bottom-up approaches

These models have a similar architecture as their predecessors, except the second part of their architectures, are hierarchically opposite of the first half. For example, U-Net [30] architecture combined contracting and expanding paths to capture both local and global contexts, proving effective for medical image segmentation tasks. Similarly, SegNet [31] introduced an encoder-decoder architecture with skip connections, efficiently handling object boundaries and fine details.

1.1.9.4 Vision Transformer-based models

With the inception of the vision transformer [32], many efforts have been carried out to utilize them for various perception applications, including semantic segmentation. Unlike

conventional CNN-based architectures, vision-based transformers apply the transformer architecture, originally designed for natural language processing tasks, to visual data. The ability of vision-based transformer models to capture long-range dependencies in images using a self-attention mechanism makes them superior in performance compared to conventional CNNs. Vision-based transformer models usually utilize a combination of CNNs, encoder-decoder structures, and variations of transformer architectures. Transformers for semantic segmentation tasks offer improved accuracy, better generalization, and increased efficiency at the expense of computational complexity, higher memory requirements, and weaker interpretability. Some notable works on vision transformer-based image segmentation include [33–38]

1.1.9.5 Semantic Segmentation in Real-time Videos

For robotics applications, semantic segmentation goes beyond image segmentation and focuses on complete scene understanding of videos. The primary issue in this form of semantic segmentation is the demanding computational task of expanding the spatial aspect of the video alongside the temporal frame rate. It is illogical to disregard temporal features and focus solely on spatial frame-by-frame characteristics when dealing with video segmentation. Given the interconnected flow between video frames, the temporal context within video semantic segmentation remains crucial, despite the significant computational cost involved.

Researchers have made efforts to address the computational demands of video processing. Strategies such as feature reuse and feature warping [39] have been suggested as solutions. Notably, datasets such as Cityscapes [40] and CamVid [41] serve as extensive resources for frame-by-frame video segmentation [42]. Recent studies have introduced segmentation techniques, including selective re-execution of feature extraction layers [43], optical flow-based feature warping [44], and Long Short-Term Memory (LSTM)-based fixed-budget keyframe selection policies [45]. Despite these advancements, a key challenge is the limited consideration of the temporal context in these approaches. Notably, using video optical flow for temporal information to expedite uncertainty estimation has been proposed as a sensible approach [46]. Prominent Transformer models such as VisTR [47], TeViT [48], and SeqFormer [49] have been applied to video segmentation tasks.

1.2 Work Motivation

Artificial Intelligence (AI) robots equipped with sophisticated algorithms and cutting-edge capabilities have the potential to transform various sectors, bringing versatility while enhancing productivity, reliability, efficiency, and safety. One of the most prominent examples of AI-powered robots is AMRs, transforming industries worldwide.

AI plays a fundamental role in the operation and functionality of AMRs. Through AI algorithms and machine learning techniques, these robots gain the ability to perceive, interpret, and respond to their environment autonomously. Various industries

are benefiting from AI integration and advancing their AMR operational capabilities, including Robotnik Automation. However, a significant limitation, especially for commercial entities, to power AI algorithms in AMRs is the availability and affordability of GPUs. Specifically, Nvidia's GPUs are in high demand and short supply. Many reasons are attributed to this cause, such as the disruption of the global supply chain for semiconductors due to the COVID-19 pandemic, the cryptocurrency mining boom, the trend of AI adoption worldwide, and the increased price of raw materials. To address these challenges, commercial entities are following alternative options suitable to them, such as procuring from different vendors, buying at a higher price, using refurbished GPUs, switching to cloud services, and switching to neural accelerators or CPUs. For mobile robots, utilizing the CPU processors or neural accelerators appears to be the most promising option for AI inference. In this regard, Román Navarro García, Software Engineer Head at Robotnik Automation, suggested exploring the potential of EI for AMRs [4] and OpenVINO [7] for AMRs at Robotnik Automation. OpenVINO's versatility, performance, and ease of optimized usage with Intel's processors was the reason for the consideration, as earlier discussed.

1.3 Objectives

The following activities are outlined to be undertaken during the intended internship duration at Robotnik Automation.

- Work in and validate all the relevant Intel EI for AMR packages on Robotnik's mobile robot platforms
- Develop and Integrate deep learning models for object detection in real-time video streams using Intel OpenVINO
- Assess point cloud filtering strategies to facilitate indoor/outdoor mobile navigation

1.4 Environment and Initial State

No prior work related to AI has ever been attempted in the proposed area at Robotnik Automation. The concerned staff at Robotnik Automation learned about the EI4AMR and OpenVINO through the Intel team, and they were unaware of the associated challenges.

CHAPTER 2

Planning and resources evaluation

Contents

Planning and resource evaluation information is provided in this chapter. Planning and resource evaluation is not as important in industrial settings due to the dynamic nature of the environment. For instance, resources are allocated according to the changing needs of ongoing and future projects. Similarly, plans are dynamically updated as per the customer's changing needs, incoming projects, and resources.

2.1 Planning

In this section, the time planning of activities and sub-activities in the form of a Gantt chart is provided (see Fig. 2.1).

2.2 Resource Evaluation

Robotnik Automation uses an internal Enterprise Resource Planning (ERP) system to manage human and hardware resources. The feasibility of the proposed activities has already been investigated by the Software Department Manager. The proposed activities are software-based, and no direct hardware costs are involved other than utilizing existing mobile robots and vision sensors. Likewise, the human resources costs cannot be quoted due to the industrial nature of the work. Nevertheless, a tentative estimation of work

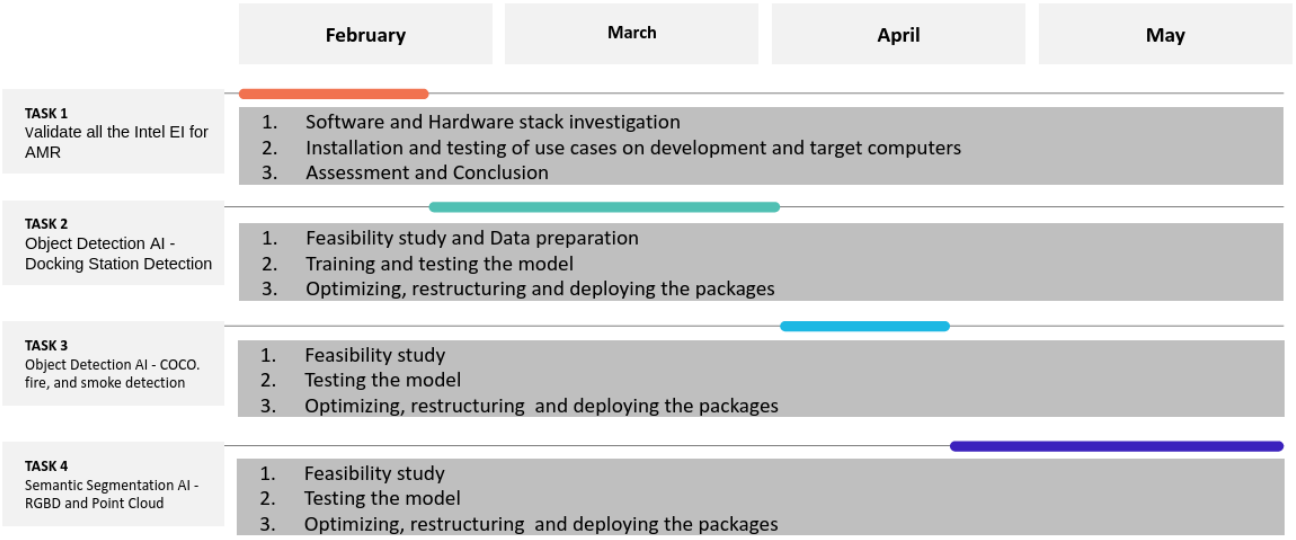


Figure 2.1: Gantt Chart of the activities performed at Robotnik Automation

hours can be inferred from the planning sheet, assuming a 40-hour workweek at Robotnik Automation.

Chapter 3

System Analysis and Design

Contents

This chapter presents the requirements analysis, design, and architecture of the proposed OpenVINO-based activities.

3.1 Requirements Analysis

To carry out proposed project activities at Robotnik Automation, a requirements analysis is performed for each project activity. In this section, the functional and non-functional requirements are listed separately for each project activity.

3.1.1 Functional Requirements

The functional requirements of all the activities are separately identified and outlined in Tables 3.1, 3.2, 3.3, and 3.4.

3.1.2 Non-functional Requirements

The following sections individually discuss the non-functional requirements related to the specific activities, concerning their applicability and reusability at Robotnik Automation.

Input:	Study, Install, and Test Intel's EI for AMR on a Robotic Platform
Output:	Validate the Intel EI for AMR for use at Robotnik Automation
Understand the EI for AMR container-based implementations of different packages. Implement the VDA5050, OpenVINO-AI, and RealSense Framework of EI for AMR on a local robotic platform	

Table 3.1: Functional requirement «Activity1: Validate EI for AMR»

Input:	Real-time video feed
Output:	Real-time bounding boxes around docking station and distance, if docking station is in the frame
Robotnik's Software team wants their robots to autonomously detect and dock with the docking station for charging. The robot captures video in real time and passes it to the AI model, which then performs processing and inference to detect and localize the docking station in the video.	

Table 3.2: Functional requirement «Activity2: Docking Station Detection»

Input:	Real-time video feed
Output:	Real-time bounding boxes around concerned objects, if those objects are in the frame
Robotnik's Software team wants their robots to detect COCO objects, fire, and smoke. The robot captures video in real time and passes it to the AI model, which performs processing and inference to detect and localize objects of interest in the video.	

Table 3.3: Functional requirement «Activity3: Detection of COCO Objects, Fire, and Smoke»

Input:	Dense point cloud or RGBD
Output:	Semantically segmented point cloud
Robotnik's Software team wants to enhance the situation understanding and obstacle avoidance capabilities of their robots. The input to the model is either RGBD frames or dense point clouds from the RGBD camera. The AI model performs the necessary processing and inference.	

Table 3.4: Functional requirement «Activity4: Semantic Segmentation via PointCloud and RGBD»

3.1.2.1 User Manual and GitHub Repositories

All the activities performed and software developed during the project have to be provided with a user manual. The user manual should provide a brief overview of the activity, a step-by-step approach to using or reproducing it on any current or future robotic platform at Robotnik Automation, including software dependencies, and concluding remarks on the project.

3.1.2.2 ROS (Ubuntu) Integrable

Each activity performed or project developed should be integrated with ROS (Ubuntu). To achieve this, additional ROS or software wrappers should be written, wherever necessary, to ensure smooth deployment and usage of the developed packages.

3.1.2.3 Minimum Dependency on ROS (Ubuntu) Version

Robotnik Automation is continuously upgrading its software stack to meet the requirements of its clients and to take advantage of advancements in newer software functionalities. Therefore, at the production scale, it is necessary to minimize software package dependency on the ROS version by disintegrating the actual AI inference code from the ROS core functionality. This can be done using object-oriented programming and Python packaging structure. The package should be tested with different versions of ROS (e.g., ROS Noetic and ROS 2 Foxy) in order to ensure compatibility.

3.1.2.4 Real-time performance

Each AI application developed during the proposed activity should be tested and expected to work on a real-time robotic platform. Therefore, the AI packages must have minimal latency and satisfactory Frames Per Second (FPS).

3.1.2.5 Lighting and illumination conditions

All developed AI applications are expected to have robust performance under various lighting conditions (indoor and outdoor).

3.2 System Design

A brief overview of the system is depicted in Fig. 3.1. More detailed architectural block diagrams are provided in the following sections.

3.3 System Architecture

This section describes the hardware capabilities and additional sensing modalities used for the given project.

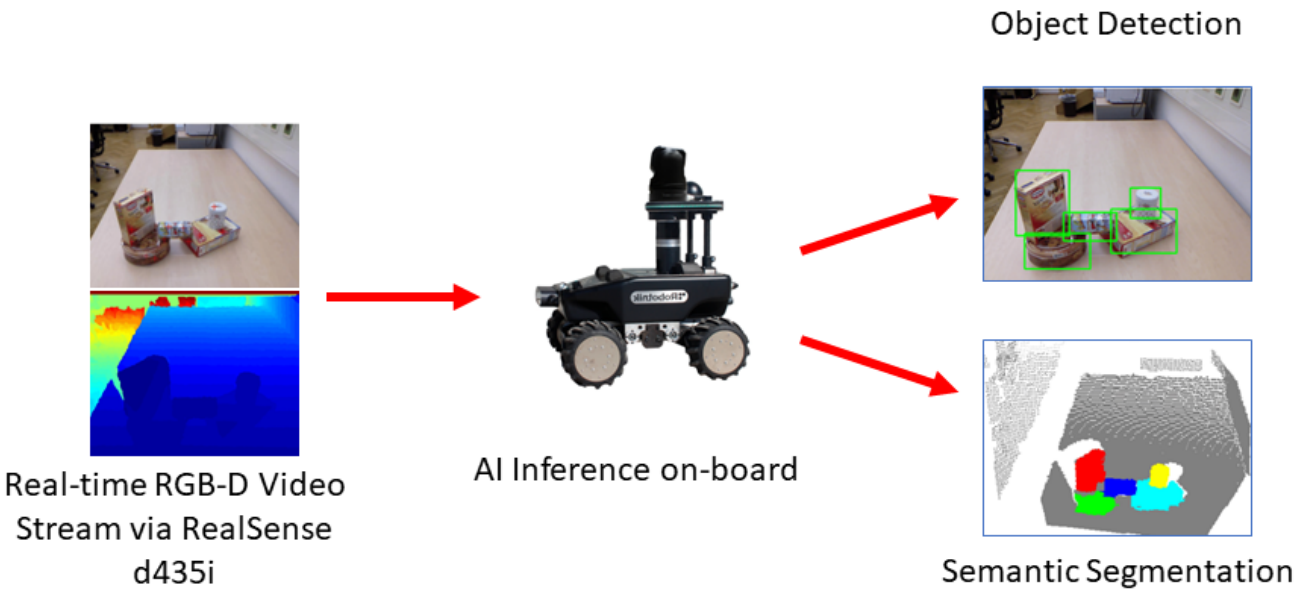


Figure 3.1: A simplified diagram of the system

3.3.1 Hardware Specification

- Intel RealSense D435i [50]
- Dedicated GPU server with NVIDIA GeForce RTX 3090 (used for training object detection models)
- Master Laptop#1 (used for scene segmentation purposes): 12th Gen Intel® Core™ i7-12650H × 16, NVIDIA RTX3060, 16GB DDR4 RAM, 1 TB HDD
- Master Laptop#2 (used for object detection algorithm development, optimization, and testing): 7th Gen Intel® Core™ 7700HQ × 8, NVIDIA GTX1050, 32GB DDR4 RAM, 1 TB HDD
- The target hardware system (RB-Watcher) specifications are available via the datasheet [51]

3.3.2 System-level Software Specification

- Ubuntu: 20.04 LTS (Long Term Support)

- ROS Noetic and ROS 2 Foxy (Some functionalities of code are tested with different versions of ROS as per the software requirement at Robotnik Automation, discussed in Section 3.1.2.3)
- Nvidia Driver Version is 510.47.03 and CUDA Version is 11.6 (Laptop#1)
- Docker Engine (client) with Docker Desktop (Server) version is 23.0.3
- Nvidia Driver Version is 470.199.02, and CUDA version is 11.4 (GPU server)

3.3.3 Activity-oriented Software Specification

For finding the activity-level software dependencies, refer to Appendix A.1.

CHAPTER 4

Work Development and Results

Contents

The developed work and the preceding results are explained in detail in this chapter, including all possible deviations from the initially intended goals.

4.1 Validate EI for AMR

There are three core benefits of the EI for AMR toolkit. First, it allows containerized software development that is deployable and scalable to many hardware systems. Second, it accelerates deployment by minimizing middleware and firmware-level developments. Third, it offers inherent device-to-edge deployment capabilities with the help of Intel's software ecosystem and cloud resources. Lastly, it provides guided tutorials, examples, and online support.

To use EI for AMR (version 2022.3.0), the base software requirements are Ubuntu 20.04 LTS, ROS 2 Foxy (with data distribution service), OpenVINO version 2021.4, Intel oneAPI Base Toolkit version 2022.2, Intel RealSense™ SDK v2.50 or higher, and the supported simulation platform is Gazebo v11.8.1 or higher. Likewise, the EI for AMR development platform's minimum hardware requirements are Intel processors, 8 GB RAM, a 64GB hard drive, and a RealSense camera D435i (if applicable). EI for AMR requires two platforms: a development station (developer's computer) and a deployment station (robot's target computer).

The EI for AMR requires a fresh installation of Ubuntu (with all the necessary GPU and other support packages), which was performed on a separate workstation as a first step to validate this toolkit at Robotnik Automation. Next, EI for AMR is downloaded, extracted, and installed. The EI for AMR toolkit is installed in the form of successive container images, so to validate the installation, any sample application can be independently run via a container.

After a successful installation, the use cases of VDA5050 and OpenVINO-AI on a local robotic platform were tested. The use of VDA5050 with EI for AMR was found to offer no significant benefits compared to Robotnik's existing Open-RMF (Open Robotics Middleware Framework[1]) usage. EI for AMR was also found to have hard dependency conflicts between Robotnik's existing software stack and EI for AMR. The validation of OpenVINO was also performed by assessing OpenVINO capabilities inside EI for AMR containers. Specific tests were conducted for use cases of containerized object detection and semantic segmentation, which were found suitable for usage. However, it was decided to use OpenVINO separately from EI for AMR due to the previously mentioned software dependency conflict. The details of OpenVINO-based packages are discussed in the following sections.

The lessons learned were that OpenVINO can be used with CPU, integrated-GPU[2], and accelerator. It has a model optimizer, and inference can be done locally or remotely.

4.2 Object Detection

In the following sections, work developments related to object detection are presented.

4.2.1 Docking Station Detection

Mobile robots at Robotnik Automation can autonomously dock to the docking station if it is visible to the camera. This process combines conventional image processing with ArUco markers[3], Light Detection and Ranging (LIDAR) data[4], and a set of movement primitives (with odometry) to reach the desired target goal while continuously minimizing the deviation error

As it can be seen in Fig. 4.1, the docking station is equipped with an ArUco marker to guide the robot during docking. The mobile robot performs satisfactorily under normal conditions. However, it fails to dock when the illumination is insufficient to read ArUco markers. To address this issue, two luminescent/reflective surfaces (located on the left and right of the ArUco marker) have been added to the docking station, as depicted in Fig. 4.1. Despite this additional feature, the mobile robot occasionally misses the docking station. Consequently, the software team at Robotnik Automation

[1] A modular open-source software framework that ensures interoperability between multiple robot fleets and physical infrastructure

[2] Integrated graphics support built into the processor

[3] An open-source library for camera pose estimation using squared markers [52]

[4] 3D point cloud captured by laser-based remote sensing technology [53]

Figure 4.1: A picture of the Docking Station used by mobile robots manufactured by Robotnik Automation

aimed to implement a more robust mechanism capable of detecting and measuring the distance from the docking station. Introducing an AI model for docking station detection emerged as an attractive alternative solution. Therefore, a decision was made to develop an optimized AI pipeline for this purpose, leveraging the advantages of OpenVINO. A comprehensive overview of the development-to-deployment phase is presented in Fig. 4.2 and further detailed in the subsequent sections from 4.2.1.1 to 4.2.1.5.

4.2.1.1 Model Selection

The first stage of the development process involves the feasibility study, requirement analysis, and resource consideration. During this stage, various similar examples were explored and benchmarked. The YOLOv7 framework [54, 55] was identified as a suitable candidate for docking station detection due to its reasonable efficiency, versatility, scalability, online support/resources, ease of integration, and optimization.

Specifically, the striking balance of YOLOv7 between accuracy and speed on resource-constrained hardware is of significant interest. Additionally, the possibility of scaling the object detection pipeline to other tasks and OpenVINO support for the YOLO architectures were practical considerations for this adoption. It is noteworthy that at

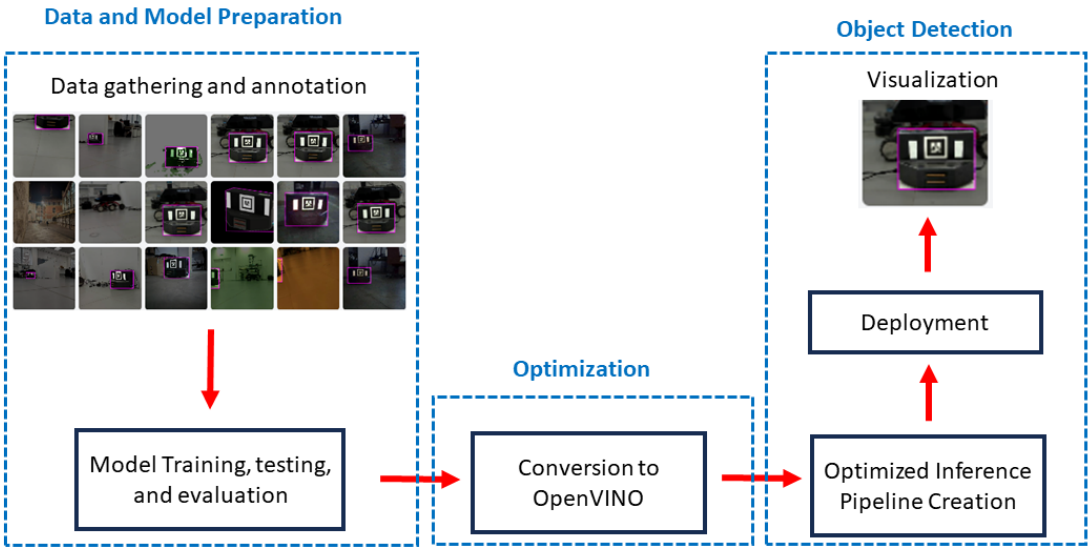


Figure 4.2: Simplified representation of docking station detection pipeline

the initiation of this project in February 2023, YOLOv7 was the most state-of-the-art model in the YOLO series with optimization support.

YOLOv7, short for "You Only Look Once version 7," was the most state-of-the-art object detection model and the latest iteration of the YOLO family of neural architectures at the initiation of this project with several key innovations and improvements over its predecessors. YOLO models are renowned for their ability to perform real-time object detection and classification in images and videos. Usually, the YOLO series of networks comprises three components - Backbone, Neck, and Head. Firstly, the Backbone encompasses a convolutional neural network that generates image features, often referred to as embeddings. These embeddings capture meaningful patterns and characteristics within the input images. On the other hand, the Neck comprises a set of neural network layers meticulously designed to amalgamate and blend the extracted features. This fusion of features is then transmitted to the subsequent stage, setting the stage for accurate predictions. Lastly, the Head is responsible for taking in the feature representations provided by the Neck and utilizing them to generate prediction outputs. YOLOv7 follows the same architecture with three key innovations: Extended-Efficient Layer Aggregation Network (E-ELAN), model scaling, and Bag of Freebies (BoF). First, E-ELAN (an extension of ELAN [56]) is used to improve the learning ability of the network without destroying the original gradient path (deteriorating the original performance), as shown in Fig. 4.3. E-ELAN is a combination of expand, shuffle, and merge operations to increase the cardinality of the computational blocks in the network - a notion of grouping convolution. In a standard convolutional layer, all input channels are connected to all output channels. This can lead to a long gradient path, as the gradient must travel through all of the input channels before it can reach the output channels.

Whereas group convolution divides the input channels into groups, and only allows connections between channels within the same group. This shortens the gradient path, as the gradient only needs to travel through the channels in the same group. E-ELAN additionally incorporates shuffle convolution, which introduces a random reordering of input channels prior to group convolution operations. This way, the gradient path is effectively shortened, requiring gradients to traverse through a randomized sequence of channels. This strategic utilization of group convolution and shuffle convolution within E-ELAN enhances the YOLOv7 model's capacity for effective learning, particularly in deep network architectures. This advancement contributes to YOLOv7's remarkable accuracy across diverse object detection benchmarks. Furthermore, E-ELAN not only enhances learning but also maintains computational efficiency by reducing the network's parameter count. This efficiency, coupled with its improved learning capabilities, positions YOLOv7 as an excellent choice for real-time object detection applications, underscoring its significance in the field of computer vision.

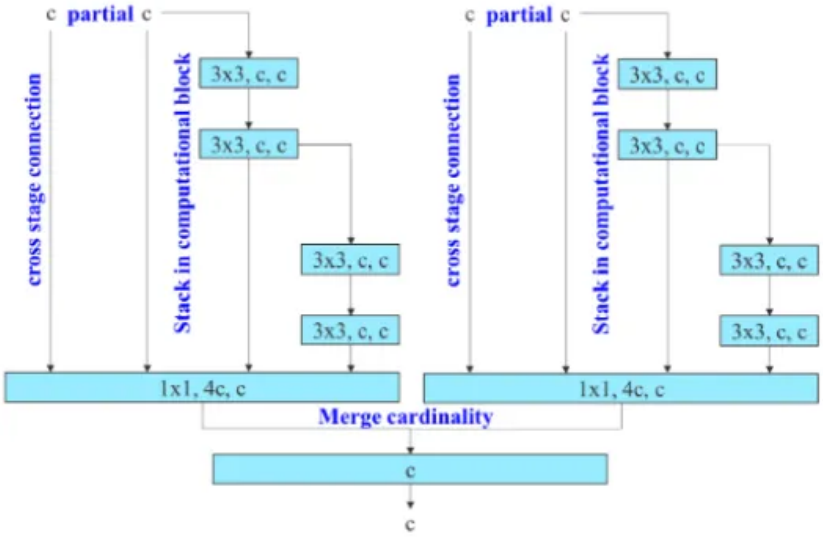


Figure 4.3: E-Elan architecture [54]

Second, model scaling serves as a technique aimed at modifying certain characteristics of a model and generating variants of varying scales, tailored to fulfill specific requirements for inference speeds. In this approach, the dimensions of width and depth are coherently scaled, aligning with concatenation-based models. This optimization technique aims to enhance the performance of YOLOv7 by systematically adjusting both width and depth parameters in a coordinated manner, as shown in Fig. 4.4.

Lastly, YOLOv7 combines BoF to improve the model performance without increasing the training cost. These techniques include Re-parameterization Convolution (RepConv), varying granularity of label assignment, batch normalization, augmentations, implicit knowledge distillation, feature map fusion, and channel pruning that balance accuracy, stability, and computational efficiency. By adopting and adjusting these tech-

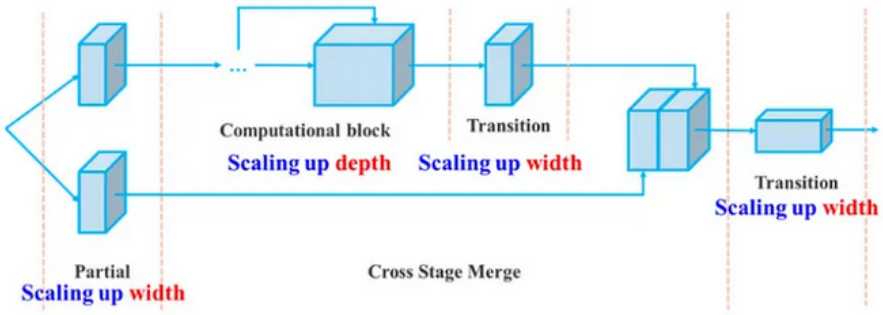


Figure 4.4: The compound scaling used in YOLOv7 [54]

niques, YOLOv7 achieves improved object detection performance while maintaining flexibility to tailor the model to specific use cases and hardware platforms.

4.2.1.2 Data Preparation

After finalizing the model, the second stage involved data gathering and preparation. Since the docking station had a unique shape, there was no existing dataset available for this task. Therefore, data had to be manually collected, labeled, and annotated. The data was collected in the form of pictures of docking stations using an Intel RealSense Depth Camera D435i and a Google Pixel 7 Pro, personal mobile phone, with a 50-megapixel back camera, capturing images at various illumination levels, orientations, and poses. The dataset comprised approximately 1200 images. Special consideration was given to capturing data in very low or bright lighting conditions to ensure the model could work accurately in extreme conditions. The collected data was manually labeled and annotated using Roboflow [57], a data labeling, preprocessing, and training platform. All pictures in the dataset were resized to 640 x 640 pixels. The labeled data was further augmented with geometric transformations (horizontal flip, crop, or translation), adjustments in color (intensity, saturation, hue), and the application of kernels (blur). Specifically, horizontal flipping between -7° and +7°, ±9° horizontal shear, ±9° vertical shear, up to 5% grayscale images, adjustments in hue between -46° and +46°, exposure changes between -25% and +25%, and up to 3% pixel noise were applied. The labeled data was made available on Roboflow Universe [58].

4.2.1.3 Model Training

The third stage is training the model, i.e., YOLOv7. The unavailability of GPU-capable platforms at Robotnik Automation was a significant hurdle for this task. This problem was solved by leveraging the GPU servers of UJI. Instead of training the model from scratch, a transfer learning approach was adopted that utilized the pre-trained YOLOv7 model weights of the COCO dataset. The labeled data of the docking station (created in an earlier stage) were downloaded from Roboflow and placed inside the YOLOv7 data folder. Some necessary changes (e.g., number of classes, hyperparameters, etc.)

were made in the .yaml files in the configuration folder of the original YOLOv7. The term "hyperparameter" here refers to the user-defined numerical values that control and influence the learning process of a machine learning model. The remaining training process was fairly simple with the GPU servers of UJI. Both tiny and full YOLOv7 models were trained to evaluate the accuracy-to-speed trade-off on mobile robots.

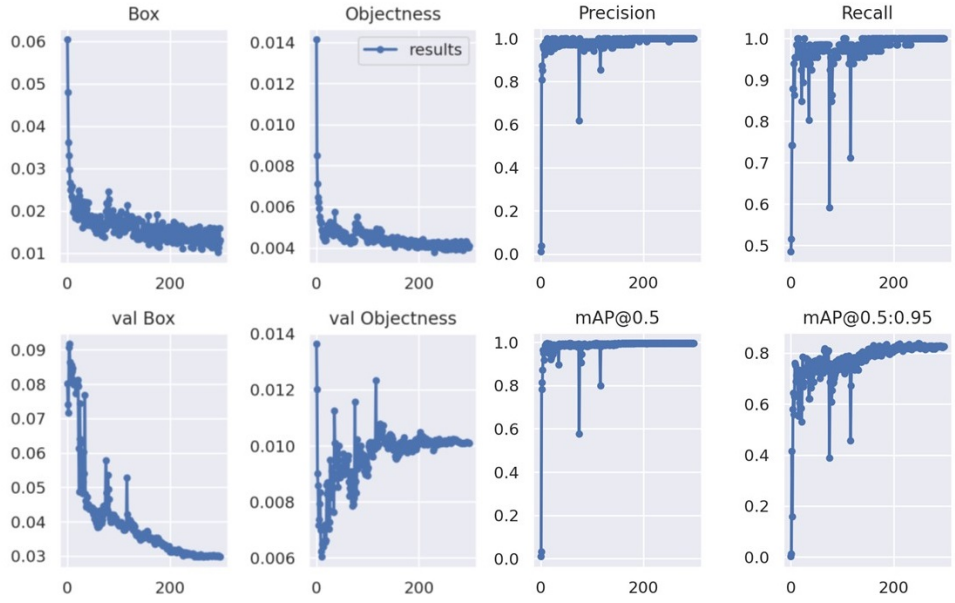
After the completion of training, both models were evaluated before proceeding to the optimization or deployment phase. This involved evaluation and analyses of performance metrics over training and validation datasets to gather insights about whether the model would be able to generalize to future unseen data. Among these metrics, the epoch loss results, confusion matrix, F1 score, precision curve (P Curve), precision-recall curve (P-R Curve), and recall curve (R Curve) play vital roles in assessing a model's performance in different aspects. It is to be noted here that the given dataset was split into three sets - training, validation, and testing - in a proportion of 88%, 8%, and 4%, respectively.

The results discussed in the following paragraphs are generated with the training and validation sets. Specifically, except "Training & Validation Results" all the plots of the remaining metrics are created with the validation set.

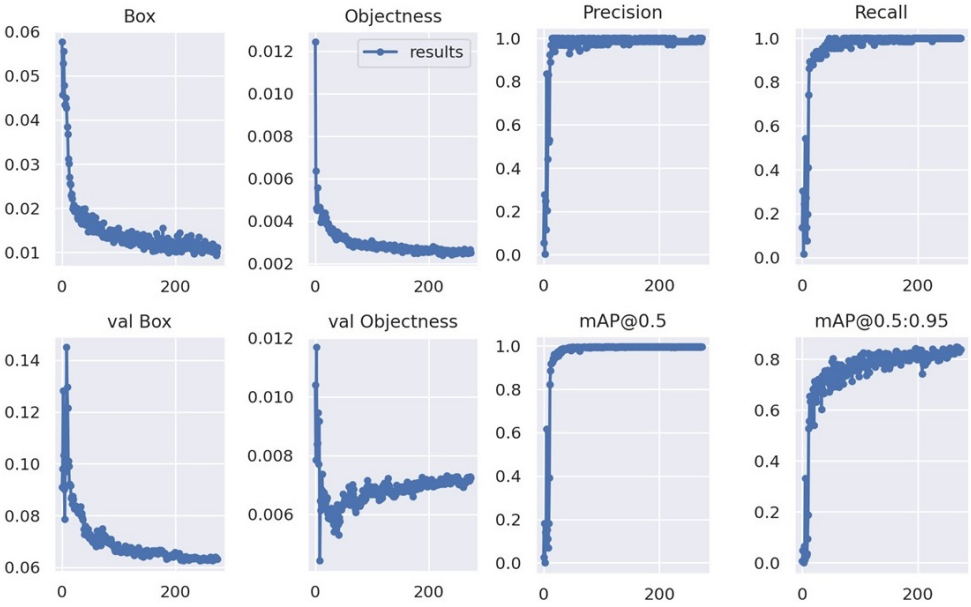
Training & Validation Results: This description explains the plots in Fig. 4.5, which provide useful insights about the training and validation performance of the model. Referring to Fig. 4.5, there are eight different plots against epochs (the number of training/validation passes the algorithm has taken). Each plot is briefly explained below:

1. Box: The Box metric reflects the localization accuracy of the bounding box predicted by the model. The box loss, sometimes called the regression loss, is a loss function used to train the YOLOv7 model to accurately predict the bounding boxes around an object of interest in an image. Mathematically, it is computed by taking a mean of (1- CIoU) between potential and actual bounding boxes, where CIoU is an acronym of Complete Intersection over Union, initially introduced in [59]. CIoU is an augmentation of the Intersection over Union (IoU), one of the most important performance metrics used in object detection and semantic segmentation. IoU encapsulates the difference in an overlap between the predicted and the true/annotated bounding boxes. Mathematically, the overlap is estimated by dividing an area of intersection between the original and predicted bounding boxes to the union of both boxes. A higher IoU (near 1) means a perfect overlap. However, IoU has some limitations, as it only considers the overlap between actual and potential bounding boxes, leading to issues with bounding boxes of different sizes, aspect ratios, and center offsets. CIoU addresses these inherent issues of IoU by incorporating two additional terms to prevent deviation from the true sizes and centers of the true bounding boxes (see [59] for more details). In Fig. 4.5, the decrease in the box (or regression) loss over the training and validation sets shows that the model can correctly identify and localize the object within the scene.

2. Objectness: In a given context, the objectness score implies the likelihood of the proposed region of interest (ROI) containing an object. The proposed ROI is a list



(a) YOLOv7 (Tiny)



(b) YOLOv7

Figure 4.5: Results for the docking detection model

of possible rectangular regions in an image that could be bounding boxes around the object of interest. YOLOv7 assigns an objectness score to every proposed ROI between 0 and 1. For prediction, YOLOv7 discards the ROIs with lower objectness scores, and selects the ROIs with higher objectness scores. YOLOv7 refines the top ROIs to form the final bounding box prediction. In light of the earlier explanation of the objectness score, the objectness loss is used here to train the YOLOv7 model to predict the likelihood of the docking station's existence in the proposed ROI. It is estimated as a binary cross-entropy loss with the sigmoid layer in a single class (*BCEWithLogitsLoss* [60]) between the predicted objectness probabilities and the ground truth objectness score. In Fig. 4.5, the decrease in objectness loss over the training and validation sets shows that the model can correctly distinguish between the docking station and the background.

3. Precision & Recall: To comprehend the concepts of precision and recall, it is essential for a reader to understand the associated terminologies of True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). These terminologies typically reflect the four possible outcomes of a binary classification problem and are briefly explained below.

   a) True positives (TP) means the model correctly predicts an instance belonging to the positive class. In other words, the model accurately identifies instances that are positive.

   b) True Negatives (TN) means the model correctly predicts an instance belonging to the negative class. In other words, the model accurately identifies instances that are negative.

   c) False Positives (FP) are cases where the model incorrectly predicts that an instance belongs to the positive class when the true class is negative, also called Type I error.

   d) False Negatives (FN) are cases where the model incorrectly predicts that an instance belongs to the negative class when the true class is positive. This is also called a Type II error.

   Both precision and recall are estimated with the help of the above four scores. Precision is the fraction of predictions that are correct, i.e., TP/(TP+FP), while recall is the fraction of ground truth objects that are detected, i.e., TP/(TP + FN). Higher values of both precision and recall are preferable. It is worth noting that the above four scores (TP, TN, FP, and FN) serve as the foundation for other evaluation strategies, such as the F1 score and confusion matrix. Returning to the discussion, the precision and recall scores in Fig. 4.5 appear to improve as the epochs pass, signifying the model's improvement.

4. mAP@0.5: In simple words, the mean average precision (mAP) metric indicates how effectively the model can localize and identify objects when allowing for a moderate amount of overlap between predicted and ground truth bounding boxes.

In technical terms, mAP is the average of the Average Precision (AP) scores for individual classes, calculated at an IoU threshold of 0.5. AP is the area under the P-R curve, which is a plot of the precision of a model as a function of its recall (discussed in 4.2.1.3). The IoU threshold defines a minimum cut-off overlap between predicted and true bounding boxes required for a prediction to be labeled as a correct prediction. In other words, a prediction of a docking station with an IoU value below 0.5 would not be considered a correct prediction by the model. In Fig. 4.5, the given metrics reach higher values over the duration of training, indicating the model's effectiveness.

5. mAP@0.5:0.95: This metric shows the mean average precision at an IoU threshold of 0.5 to 0.95. It provides a more comprehensive assessment of the model's performance across different levels of bounding box overlap. Higher values indicate better object detection across a range of IoU thresholds. In Fig. 4.5, the mAP@0.5:0.9 metrics reach higher values over the training duration, indicating the model's effectiveness. It can also be seen that the evolution of YOLOv7 recall, precision, and mAP over training epochs is more stable than YOLOv7(tiny).

Confusion matrix: The confusion matrix provides a holistic overview of a classifier's performance by summarizing the TP, TN, FP, and FN predictions in a tabular form (an NxN square matrix, where N is the number of classes). In a binary classification problem, the top left and bottom right corners of the confusion matrix represent TP and TN blocks, respectively. All the upper diagonal elements are FP cases and all the lower diagonal elements are FN cases, considering the columns represent the actual/true values in a confusion matrix. The performance of the model could be analyzed by looking into the instances when the classes are correctly or incorrectly classified. If normalized for a binary classification problem, an ideal 2x2 confusion matrix would be of the form [1, 0; 0 1]. Assuming that the rows represent predicted classes and the columns represent actual/true classes, the sum of each column would be equal to 1 if the confusion matrix is normalized. All diagonal elements are considered correct classification, while off-diagonal elements (false-negative, false-positive) are considered mislabeled classification.

The above explanation of a confusion matrix for classification problems could be extended to object detection with a minor difference in interpretation. The primary reason for this difference in interpretation is due to the difference in the nature of the object detection task. Firstly, in the object detection task, an additional class of background is appended. Therefore, in the case of the docking station, the number of classes is 2 (Background and Docking Station). Consequently, the confusion matrix would be a 2x2 matrix, as shown in Fig. 4.6. It is important to note here that the background class is not predicted as a background by the model. This is because there are no annotated samples of background in the object detection dataset; instead, every background image in the dataset is marked as "Null". In other words, a background image is considered to have no object of interest. Therefore, the TN portion of the confusion matrix (i.e., the bottom-right box) would ideally be empty, and it does not mean the incorrectness of the model or confusion matrix. Another key consideration here is of two terminologies,

Background FP and Background FN, as appeared in Fig. 4.6. Background FP refers to background objects that do not belong to either of the classes but are detected as one of them, and Background FN refers to Docking Station or Non-Docking Station objects missed by the detector and considered as some other background objects. It is important to note here that the Background FP includes cases even when more than one correct detection is made for the same object. Therefore, the terms "FP" and "FN" are concatenated along with the term "Background" instead of simply writing "Background" to signify that background does not represent a true class.

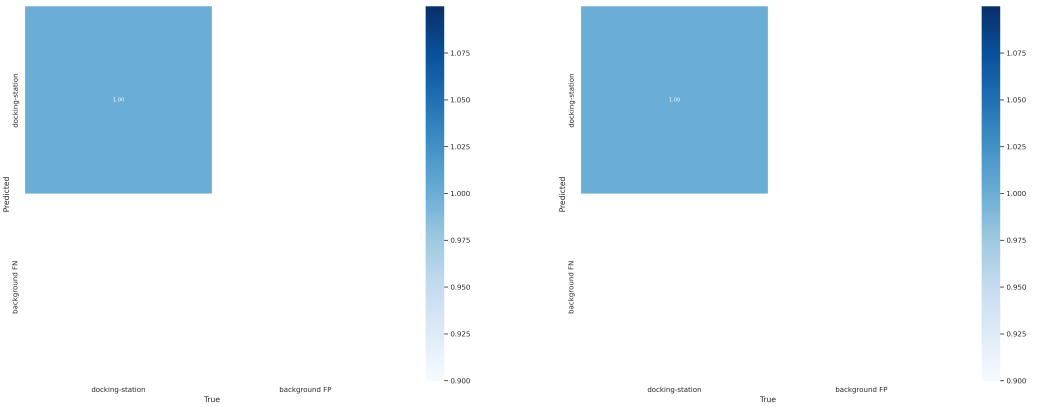
The model should ideally minimize both Background FN and Background FP (off-diagonal elements). As evident from Fig. 4.6, there is no FN or FP detection (no numeric values inside the FN/FP box means zero). Likewise, a numeric value of 1 (maximum, equivalent to 100%) appears inside the TP box. Therefore, it can be concluded that the given model detects objects satisfactorily. It should also be noted here that YOLOv7 (full) and YOLOv7 (Tiny) have similar confusion matrix plots, depicting there is no major difference in their performance for the given task.

It is essential to mention here that the confusion matrix in Fig. 4.6) and all the subsequent figures in the following docking station detection problem are generated with a confidence threshold of 0.5 and IoU threshold of 0.65.The concept of IoU threshold is already explained earlier; however, it is important to introduce the concept of confidence threshold, as it will also be referred to later. The confidence threshold refers to the level of confidence with which the object detection model is allowed to make detections. When an object detection model makes a prediction, it assigns a probability score to all of its predictions, indicating the model's confidence that a specific prediction instance belongs to a specific positive class. A confidence threshold serves as a numerical boundary criterion that plays a pivotal role in the classification of objects. If the predicted probability or score is above the threshold, the instance is classified as positive; otherwise, it is classified as negative. To illustrate, consider a scenario in which object detection with a confidence threshold of 0.5 is required. If the model predicts an object of interest with a 0.4 confidence value, it would not be classified as an object of interest.

P-R curve: A P-R curve for an object detection problem is a plot of precision (y-axis) versus recall (x-axis). It is a useful tool for evaluating the performance of object detection models, as it shows the trade-off between precision and recall.

The ideal P-R curve is a straight line from the top left corner of the graph (0,1) to the top right corner (1,1) and from the top right corner (1,1) to the bottom right corner (1,0). This signifies the maximum area under the P-R curve (AUC-PR), equivalent to 1. The AUC-PR is a metric that is often used to summarize the performance of an object detection model on a given dataset. A higher AUC-PR indicates that the model has a better overall performance, meaning that it is able to detect all of the ground truth objects with good precision. In practice, P-R curves often deviate from ideal behavior. Therefore, it is important to evaluate a model's P-R curve in comparison with an ideal curve to assess its suitability for the given task.

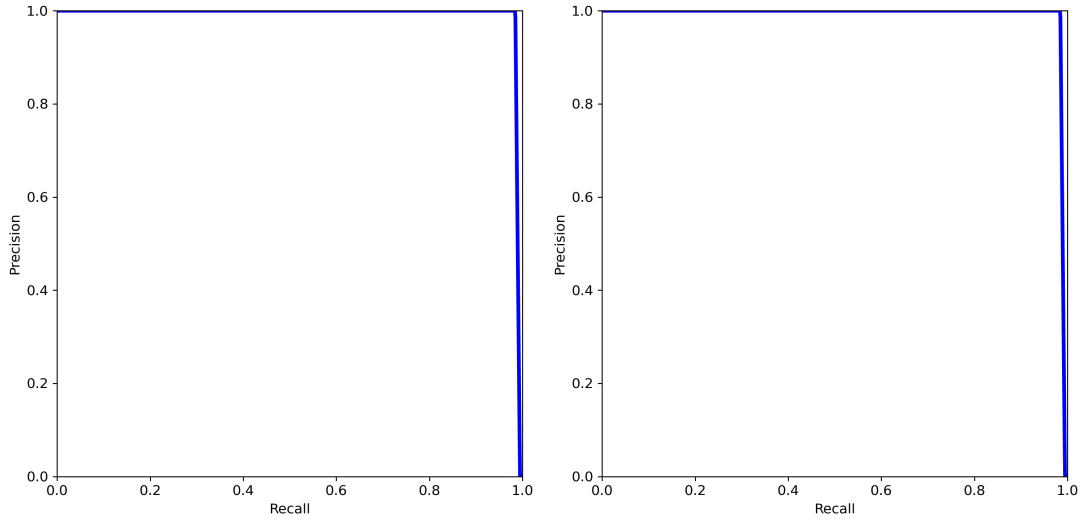
The P-R curve of the docking station detection model is plotted in Fig. 4.7. As



(a) YOLOv7 (Tiny) (b) YOLOv7

Figure 4.6: Confusion Matrix Plot for the docking station detection models

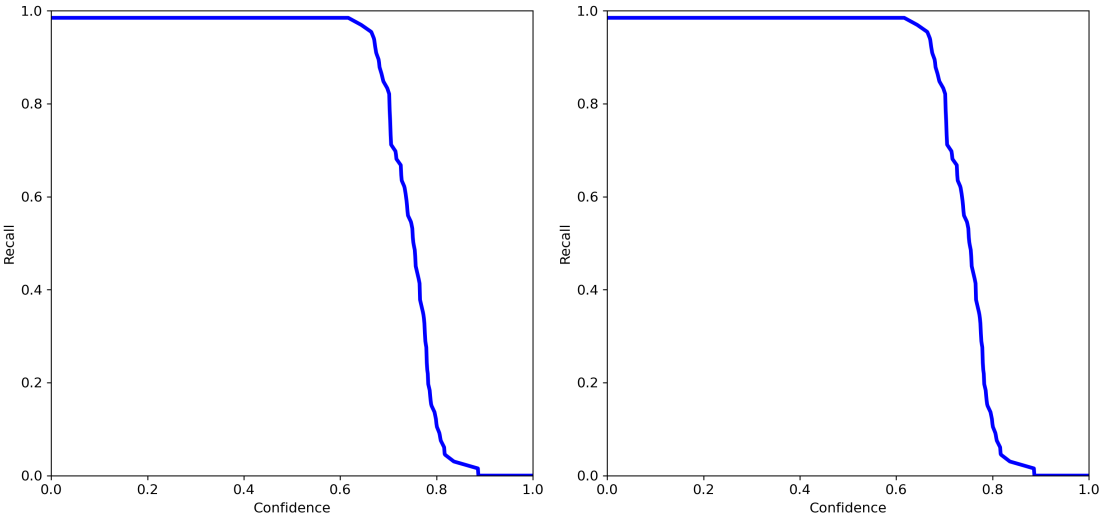
evident from the figure, both models have a near-ideal plot and achieve an approximate AUC-PR of 0.98 at mAP@0.5. Therefore, it can be concluded that these models are accurate and usable.



(a) YOLOv7 (Tiny) (b) YOLOv7

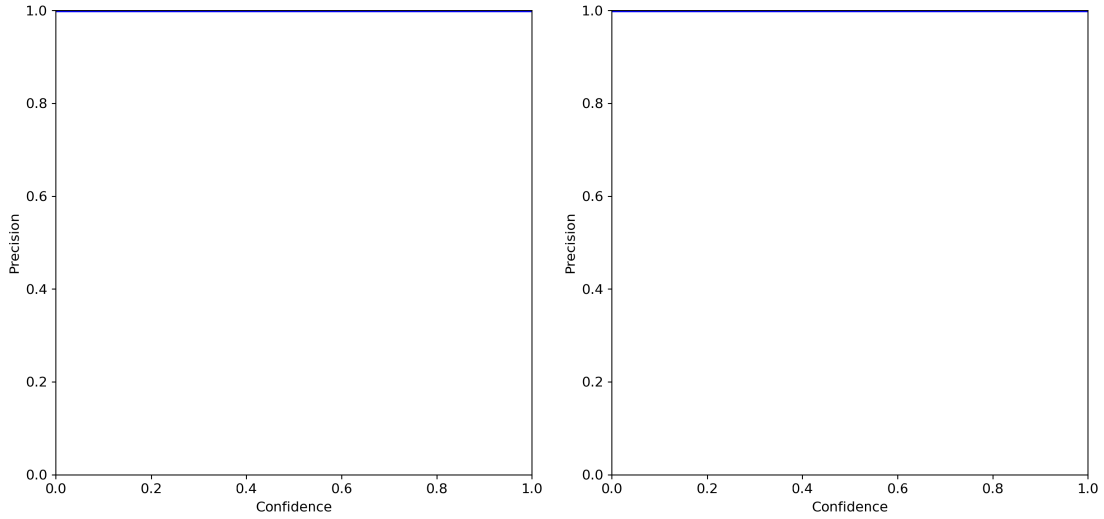
Figure 4.7: P-R Curve Plot for the docking station detection models

F1 Score curve: The F1 score curve is a visual plot of the F1 score (y-axis) against confidence or decision thresholds (x-axis). F1 score is a commonly used metric in object detection tasks to evaluate the model's performance. It is mathematically computed as, $F_1 = \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$. The F1 score provides an estimate of a model's accuracy, taking



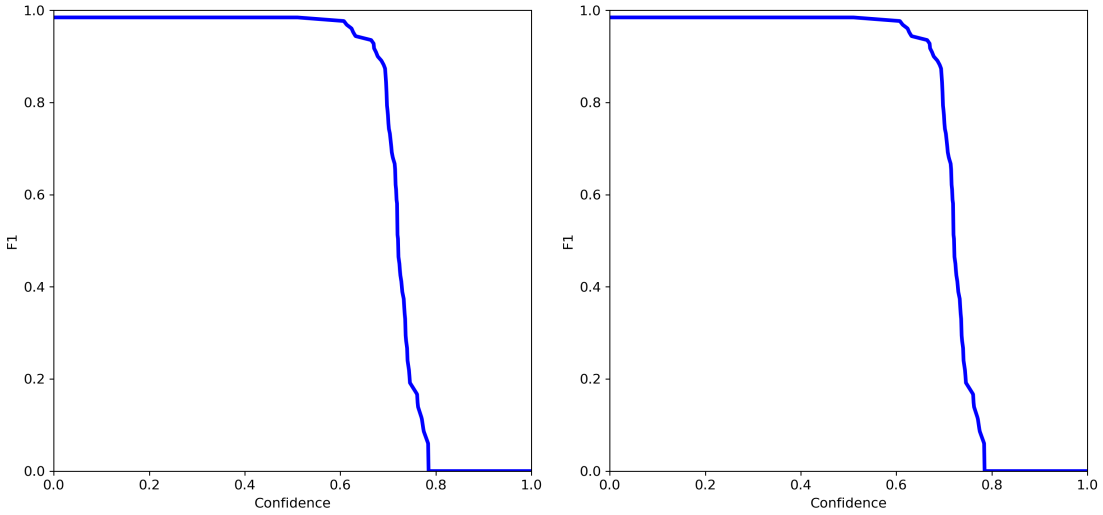
(a) YOLOv7 (Tiny) (b) YOLOv7

Figure 4.8: R Curve Plot for the docking station detection models



(a) YOLOv7 (Tiny) (b) YOLOv7

Figure 4.9: P Curve Plot for the docking station detection models



(a) YOLOv7 (Tiny) (b) YOLOv7

Figure 4.10: F1 Curve Plot for the docking station detection models

both precision and recall into account. Usually, when the model has a higher confidence threshold, precision improves while recall deteriorates. In the case of the docking station dataset, the normal and tiny YOLOv7 models achieve similar maximum recall scores of around 0.98 at zero confidence threshold, as shown in Fig. 4.8. The precision score for both models is close to 1 at all confidence values, as shown in Fig. 4.9.

Now by plotting the F1 score curve, a balance between precision and recall in terms of confidence threshold is investigated, and then an AI model could be operated at that value of confidence. From Fig. 4.10, it can be seen that both models operate well between the confidence threshold range of 0-0.78 at the IoU of 0.65, and the maximum F1 score value is approximately 0.99. In other words, the model can predict accurately between this range of confidence threshold with an approximate 0.99 F1 score. The F1 score at a higher confidence threshold is zero because it is linked to precision and recall values. Since recall shrinks to zero at a higher confidence threshold, the F1 score also reduces to zero. However, analyzing Fig. 4.10, it can be concluded that the confidence threshold of 0.5 is the most suitable.

4.2.1.4 Model Optimization

The fourth stage involves the conversion and optimization of the trained YOLOv7 models from PyTorch to OpenVINO. PyTorch is a free and open-source machine learning library based on the Torch library, widely used for developing deep learning models and their associated applications. Initially developed by Meta AI[5], it is now part of the

[5]https://ai.meta.com/

Linux Foundation[6]. PyTorch offers a flexible and dynamic computational framework, allowing users to conveniently build, train, and optimize neural networks. Its defining feature, the dynamic computation graph, makes it more intuitive and suitable for tasks involving dynamic or variable-sized data compared to static computation graph frameworks like TensorFlow. Considering all the aforementioned technical advantages provided by the PyTorch framework, along with its flexibility, community support, ease of experimentation, and compatibility with hardware acceleration, all recent versions of YOLO (including YOLOv7) were originally developed as PyTorch models. This, in turn, contributes to and advances the state of the art in object detection.

The conversion of the PyTorch model to OpenVINO is a two-step process: the conversion of the PyTorch model to ONNX (Open Neural Network Exchange[7]) and then the conversion of the ONNX-based model to OpenVINO Intermediate Representation (IR). Although OpenVINO can directly read, load, and infer with the ONNX model, it is recommended and better to use IR. All PyTorch models are defined in Python and can be exported to ONNX representation using the built-in *torch.onnx.export()* method. Calling the *torch.onnx.export()* method requires a model instance, input shape, and output model path/name. Running the aforementioned proecdure, internally traces the model execution and then exports the traced model to the requested path (as *.onnx*). The correct and accurate conversion to the ONNX model is essential for the next step, i.e., IR. Special consideration is required for the opset version[8] while converting a PyTorch model to ONNX, as old opset versions may not support the new neural network layers, leading to compatibility issues with ONNX runtime and the model. The opset used during this task is 11. The Python file (code) responsible for converting PyTorch to ONNX is provided in the [54] with the name "export.py".
See the command for converting the PyTorch model (e.g., tiny) to the ONNX without the non-maximum suppression layer [62]:

```
$ python3 export.py  --weights best-tiny.pt  --img 640 --batch 1
```

See the command for converting the PyTorch model (e.g., tiny) to the ONNX with the non-maximum suppression layer:

```
$ python3 export.py --weights .best-tiny.pt --grid --end2end --simplify
--iou-thres 0.65 --conf-thres 0.5 --img-size 640 640
```

The additional arguments are recommended by the original YOLOv7 implementation [54]. Therefore, additional arguments are passed as it later simplifies the inference implementation and end-to-end usage.

After obtaining the ONNX representation, a model optimizer is used. Specifically, OpenVINO's Model Optimizer with the ONNX network path, output layers, input

[6]https://www.linuxfoundation.org/

[7]ONNX is an open-source standard for representing deep learning models. ONNX runtime optimizes and accelerates ONNX-based machine-learning models [61]

[8]Each ONNX model uses a specific set of operators and specifications, called opset version

size, batch, and the other necessary arguments as inputs. YOLOv7 ONNX conversion to IR requires output node specification as well. In order to know the output layers of YOLOv7, Neutron [63] is used for visualization. After identifying the output nodes (here */model/model.77/m.0/Conv*, */model/model.77/m.1/Conv*, and */model/model.77/m.2/Conv* are three output nodes), the model optimizer is called.
See the command for converting the ONNX to the IR without the non-maximum suppression layer:

```
$ mo --input_model best-tiny.onnx --output /model/model.77/m.0/Conv,
/model/model.77/m.1/Conv,/model/model.77/m.2/Conv
```

See the command for converting the ONNX to the IR with the non-maximum suppression layer:

```
$ mo --input_model best-tiny.onnx
```

4.2.1.5 Model Deployment

The fifth and final stage was the integration of the optimized YOLOv7 model obtained in an earlier stage into the existing software stack of robots at Robotnik Automation. For this purpose, a ROS wrapper was written that subscribes to RGB camera nodes (or launches, if non-existent), and gathers real-time images (video feed). It publishes to YOLOv7 nodes (or launches, if non-existent) the images (video feed) with marked bounding boxes, bounding boxes pixel coordinates, and confidence scores. The launch and configuration files are set up accordingly. The ROS wrapper uses standard and private ROS libraries outlined in Appendix A.1. The YOLOv7 inference class is separately importable as per the software standardization guide of Robotnik Automation.

After the whole integration, the RB-Watcher was incorporated with the developed pipeline and tested in real-time. The performance and speed were found to be satisfactory by the software department team. The code repository and demonstration video are in the results section 4.4. Additionally, a support guide (readme file) is created to help colleagues reuse and reproduce the package on any of the existing and future mobile robots at Robotnik Automation and provided along with the code repository.

4.2.2 Detection of Objects from the COCO Dataset

In order to facilitate additional surveillance and inspection functionalities, the pretrained YOLOv7 model with the COCO dataset [22] is utilized, optimized, and integrated. Using the mentioned AI model, a customized ROS package is developed to detect objects of interest within the available classes of the COCO dataset (defined by the user). By default, only human detection capability is enabled; however, based on the user's needs, other classes from the COCO dataset can also be detected. Therefore, the object detection demo from the COCO dataset in section 4.4 only displays a bounding box around humans in the video.

4.2.3 Detection of fire and smoke

A YOLOv7 model is also trained using a custom fire/smoke dataset, available at [64]. The preparation of the fire/smoke dataset is relatively simpler than that of the docking station dataset. This is because images of fire and smoke are not manually captured and annotated; instead, various smaller fire/smoke datasets are combined (or augmented) to create a larger fire/smoke dataset for our use [64] to achieve satisfactory performance. The curated dataset comprises approximately 1800 training images, 200 validation images, and around 70 images for testing. Each image in the dataset has dimensions of 640 x 640. This dataset is also created using Roboflow [57], similar to the docking station dataset. Data augmentation is applied, including geometric horizontal flip, maximum zooming up to 38%, color hue variation between -31° and +31°, kernel blur up to 1.75px, exposure variation between -25% and +25%, and pixel noise up to 5%. The same training procedure is followed as for the docking station detection. The mean Average Precision (mAP), Precision, and Recall of the training process are 84.2%, 88.8%, and 79.0%, respectively. Similar to Section 4.2.1.3, the YOLOv7 (tiny) trained model is evaluated.

Training & Validation Results: Referring to Fig. 4.11, it is interesting to see that the Box and Objectness loss gradually reduce over a larger span of epochs. This indicates that training over a higher number of epochs has helped improve the model. Ultimately, the model converges to lower precision, recall, mAP, and other loss scores at the end of the training epochs, indicating the model's suitability for use.

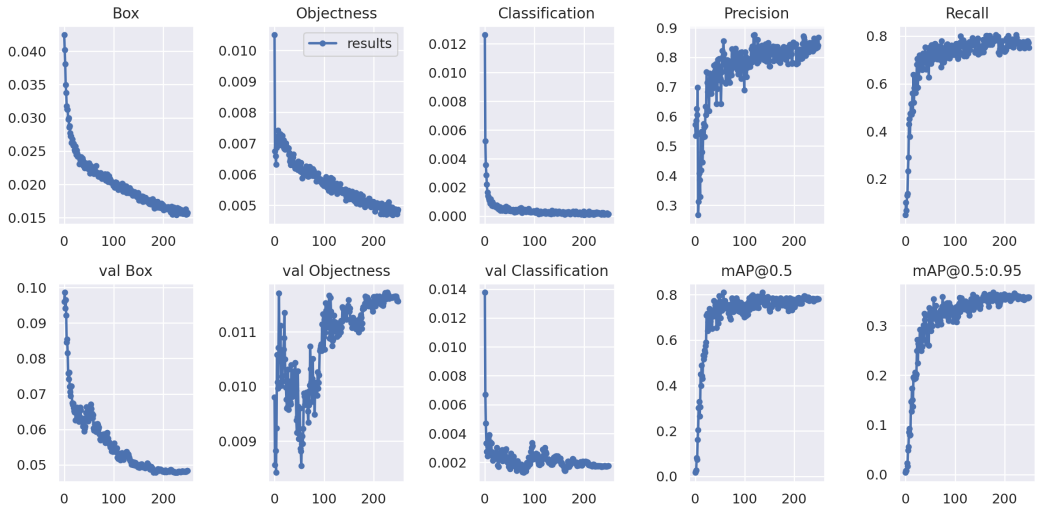


Figure 4.11: Training Results of YOLOv7 (Tiny) model for fire and smoke detection

Confusion Matrix: Similar to the docking station detection, the confusion matrix is plotted for fire and smoke detection with the validation set. A confidence threshold

of 0.4 and an IoU threshold of 0.75 is used for plotting the confusion matrix shown in Fig. 4.12. Unlike the docking station detection task with two classes, here in the given task, there are three classes: fire, smoke, and background (i.e., no fire and no smoke). It is noteworthy that fire and smoke detection pose a relatively more challenging problem for an AI model compared to docking station detection. Consequently, there are some FN predictions for both fire and smoke, as well as FP predictions for these classes. To elaborate, referring to Fig. 4.12, the model fails to detect fire 23% of the time, while it detects fire correctly 77% of the time. Similarly, the model fails to detect smoke 7% of the time, while correctly detecting it 93% of the time. If the FP cases are considered, the model incorrectly detects fire with a 9% probability and smoke with a 31% probability when there is only background. A higher FP rate does not necessarily imply that the model would always generate false predictions; it is simply a score indicating the model's performance on the given validation set. Furthermore, in critical situations such as a fire alert, it is preferable to have a higher FP rate than FN rate. In simpler terms, it is better to incorrectly detect a fire than to miss detecting a fire when it is present.

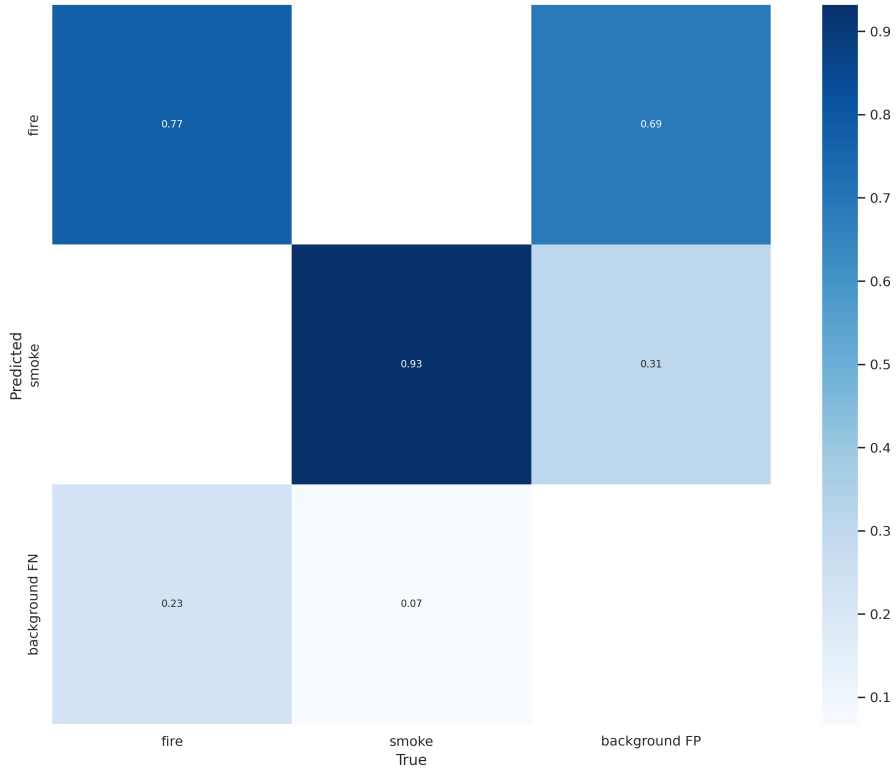


Figure 4.12: Confusion Matrix Plot of the YOLOv7 (Tiny) model for fire and smoke detection

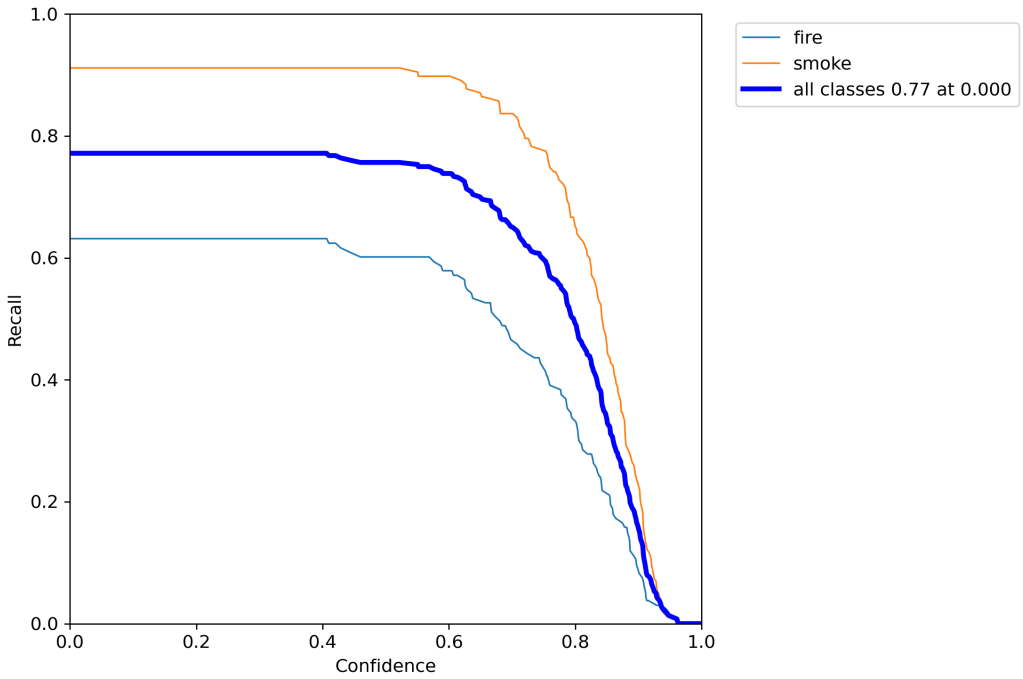


Figure 4.13: R Curve Plot of the YOLOv7 (Tiny) model for fire and smoke detection

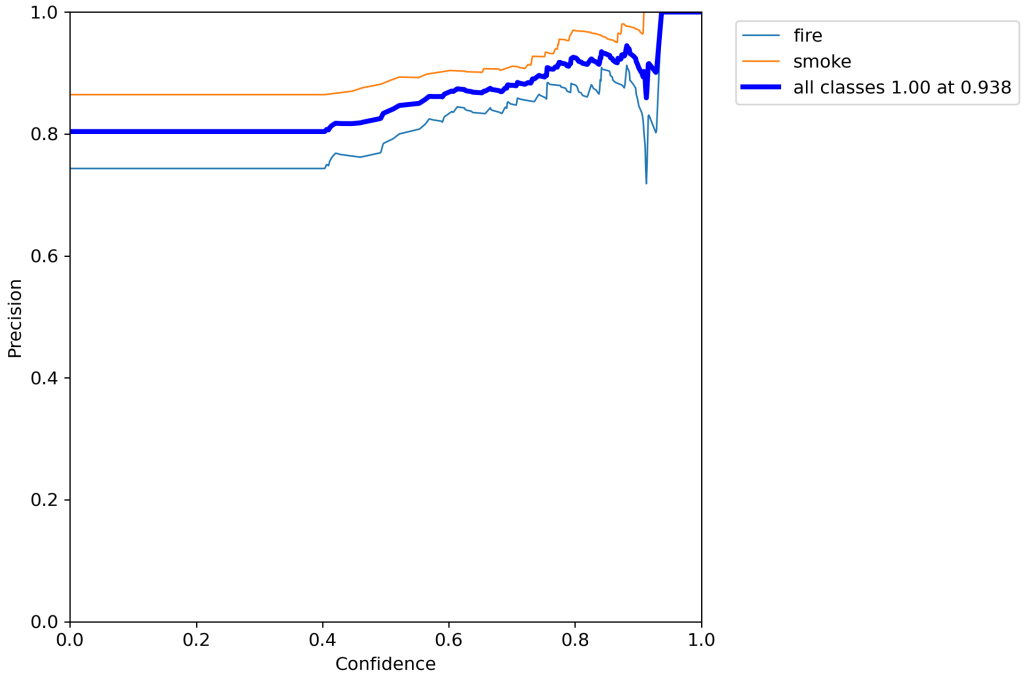


Figure 4.14: P Curve Plot of the YOLOv7 (Tiny) model for fire and smoke detection

F1 Score curve: Referring to Fig. 4.13 and 4.14, the smoke class apparently has a better precision and recall score than fire at different confidence thresholds. Referring to Fig. 4.15, it is also evident that the confidence threshold of 0.4 is suitable for balancing precision-recall.

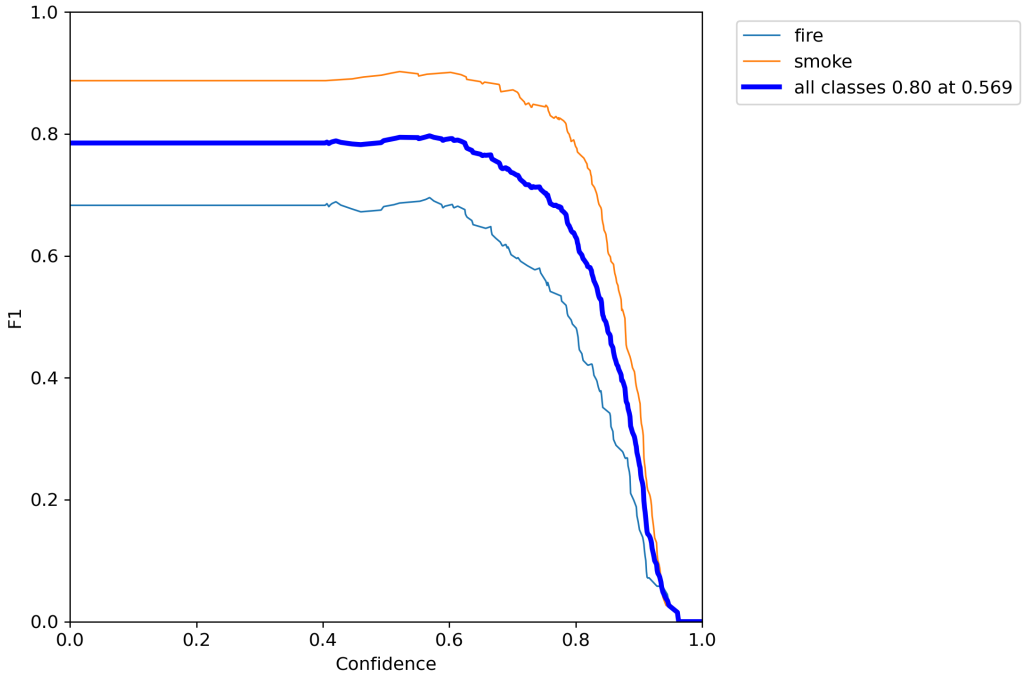


Figure 4.15: F1 Curve Plot of the YOLOv7 (Tiny) model for fire and smoke detection

P-R curve: Referring to Fig. 4.16, it is noticeable that YOLOv7 demonstrates acceptable performance. This is evident from the fire and smoke plots, each having an area under the curve (AUC-PR) of over 50%, with values of 0.562 and 0.885, respectively (as indicated in the legends at the top right corner).

It is noteworthy that all object detection models (docking station detection, fire/smoke, and COCO dataset) are separately trained, optimized, and deployed in a mobile robot. This approach optimizes resource usage, speed, and accuracy for individual use cases, instead of integrating all detection functionalities into one model.

4.3 Scene Segmentation

In the following sections, work developments related to scene segmentation are presented.

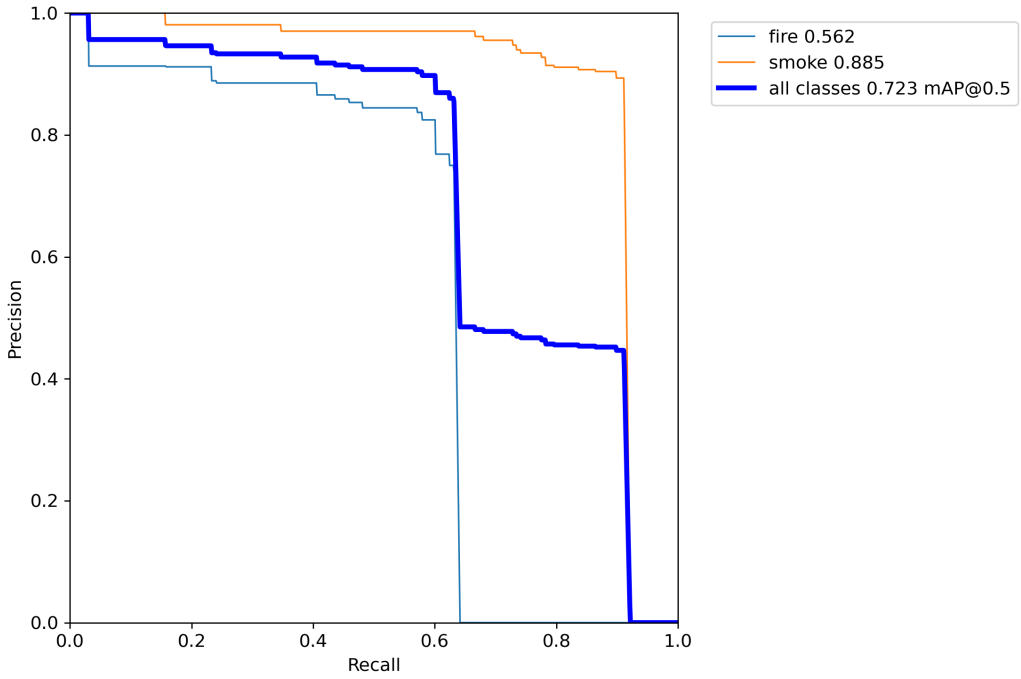
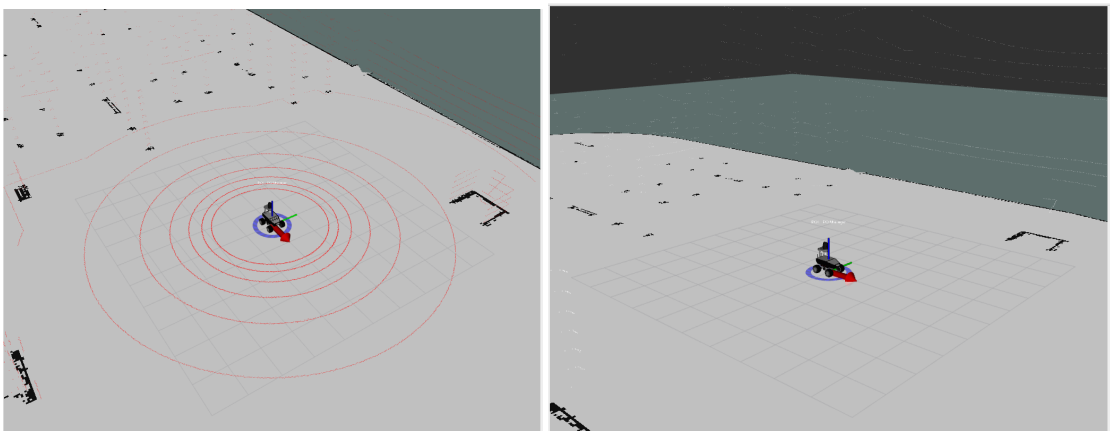


Figure 4.16: PR Curve Plot of the YOLOv7 (Tiny) model for fire and smoke detection

4.3.1 Ground Plane Segmentation

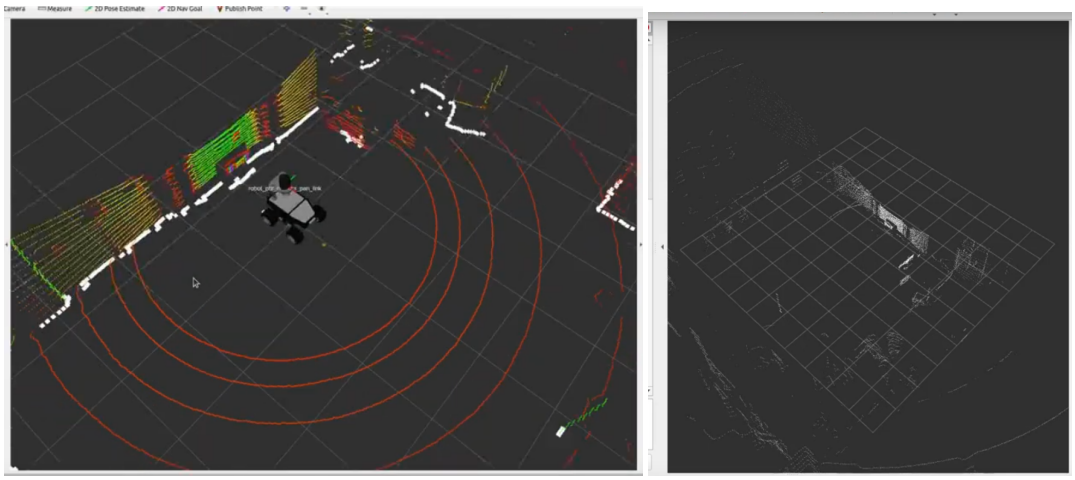
The current methodology employed by Robotnik Automation associates point clouds below a specified height, such as 0.05m, to the floor and disregard (filter out) such data points for mapping and navigation purposes. This approach becomes vulnerable to inaccuracies in certain situations, especially when confronted with sloped surfaces, reflective materials, or small obstacles on the floor. Therefore, the main goal of ground plane segmentation is to improve the identification and filtering of the ground point cloud data to aid obstacle avoidance and navigation of mobile robots at Robotnik Automation. Typically, the segmentation process involves classifying each point in the point cloud as belonging to the ground or other objects. Usually, points belonging to the ground are identified based on geometric characteristics, e.g., relatively flat and having consistent height values. There are various methodologies of clustering, plane fitting, and machine learning for ground plane segmentation.

In the given case, RANSAC (Random Sample Consensus, [65]) is used to filter horizontal planes from point cloud data robustly. RANSAC is an iterative algorithm for estimating the parameters of a mathematical model from a set of observed data that contains outliers. RANSAC follows a four-step process to identify the plane. First, randomly select a subset of data. Second, fit a plane model ($Ax + By + Cz + D = 0$) to the given data using any numerical method. Third, calculate inliers, points whose distance from each other is below a certain threshold. Fourth, repeat the process for a



(a) with unfiltered point cloud (b) with filtered ground plane point cloud

Figure 4.17: A snapshot of Gazebo's Simulation visualized in RViz



(a) unfiltered point cloud (b) filtered floor point cloud

Figure 4.18: A snapshot of RB-Watcher's real-time point cloud visualized in RViz

(a) Correctly segmenting (b) Not correctly segmenting

Figure 4.19: A snapshot of the floor segmentation model with two different elevation angles

certain number of iterations. Lastly, select the best-fitted model with the highest number of inliers. The RANSAC ground segmentation task is implemented via the Open3D library [66], an open-source library for 3D data software development (in C++ and Python). Both simulation (a custom grid station scenario) and real-world (indoor lab Robotnik Automation) demo videos are provided in the results section, and screenshots are added here to aid discussion. An RB-Watcher platform is used to obtain results from RANSAC. It is clear from the simulation results in Fig. 4.17 that RANSAC could segment and filter the ground plane. For example, in Fig. 4.17b, it can be observed that the point clouds at a higher elevation on the top right are still present (in white color), while ground points are entirely filtered. However, when the same method was tested in the real world, the performance was unsatisfactory even after tuning the RANSAC parameters. Specifically, RANSAC could not entirely filter all the ground points, as evident from Fig. 4.18 (some of the ground points in the top right portion of Fig. 4.18b are not filtered). In other words, although RANSAC can filter the ground plane to some extent, it still leaves some ground plane points, especially in far-off regions. Therefore, a more robust approach is necessary for exploration.

The next attempt to solve the ground segmentation problem involves exploring the OpenVINO pre-trained road segmentation model (*road-segmentation-adas-0001*) from Open Model Zoo [67]. This network classifies each pixel into four classes: background, road, curb, and mark. The pre-trained model's mean IoU and mean accuracy over all classes are 0.844 and 0.899, respectively, calculated based on 500 images from the "Mighty AI" dataset. The results are initially tested on the RealSense RGB stream to evaluate the feasibility of subsequent usage. The model correctly segments the floor to some extent in the case of a lower elevation field of view. However, it completely fails to segment the ground when the field-of-view is at a higher elevation, as shown in Fig. 4.19. Therefore, advanced methods of semantic segmentation are explored, motivated by two primary reasons. First, to accurately filter the floor, and secondly, to achieve more segmentation classes that could be helpful for indoor/outdoor navigation, such as vegetation and clutter.

4.3.2 Semantic Segmentation

In the context of a given task, semantic segmentation is used to identify surrounding objects and obstacles such as the floor, ceiling, clutter, humans, etc. This task is of interest to the Software team at Robotnik Automation because it could provide mobile robots with better situational awareness. Specifically, there is a significant interest in filtering the floor and ceiling point cloud data to enhance obstacle avoidance during indoor navigation tasks.

Two possible directions are considered for this semantic segmentation task: RGBD semantic segmentation and point-cloud segmentation. First, the methodology adopted for RGBD semantic segmentation is described, followed by the point-cloud segmentation methodology.

The methodology adopted for the proposed RGBD segmentation is from [68]. There are three specific reasons for using this model: firstly, its acclaimed real-time performance on mobile robots; secondly, it demonstrates improved performance due to the incorporation of additional depth channels; and lastly, the model employs simple neural network operations designed specifically for use with ONNX (in this case, with OpenVINO) [68]. A brief overview of [68] is presented here to provide a better understanding of the aforementioned justifications.

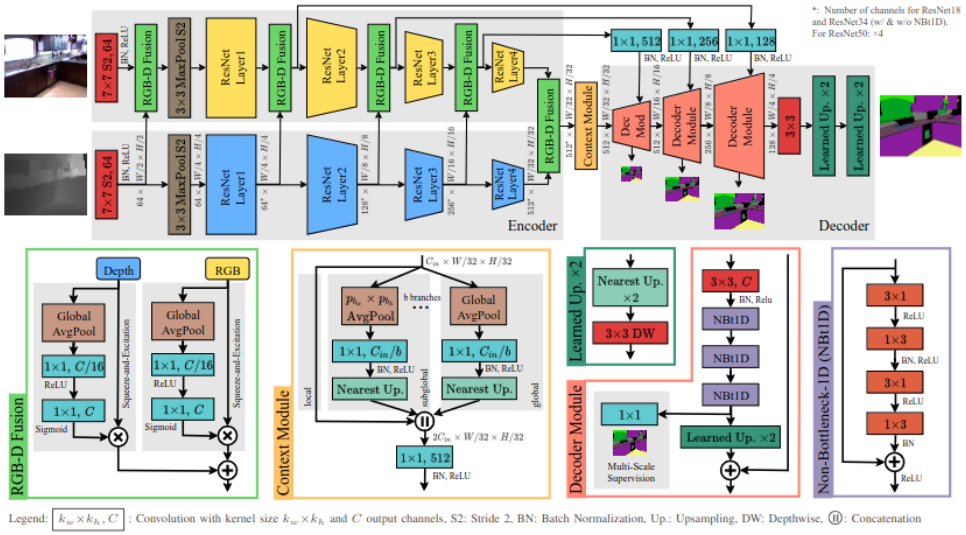


Figure 4.20: A Flow diagram taken from [68] representing RGBD segmentation (top) and specific network parts (bottom)

The ESANET [68] draws inspiration from the RGB segmentation approach of SwiftNet [69]. It follows a similar structure, featuring a shallow encoder based on the pretrained ResNet18 backbone, substantial downsampling, and a context module to aggregate features at different scales to overcome the limited receptive field of ResNet, similar to the Pyramid Pooling Module in PSPNet [29]. Additionally, it uses a shallow decoder with encoder skip connections. However, unlike SwiftNet, which solely focuses on RGB

data, ESANet leverages an additional encoder dedicated to depth data. This supplementary depth encoder captures valuable geometric details, which are then fused into the RGB encoder using an attention mechanism at various stages. The RGBD fusion is one of the key components of this entire architecture. In each of the five resolution stages within the encoders (as shown in Fig. 4.20), depth-related features are integrated into the RGB encoder. First, features from both RGB and depth modalities undergo reweighting via a Squeeze and Excitation (SE) module [70], followed by element-wise summation, highlighted in gray in the RGBD Fusion block in Fig. 4.20. This utilization of a channel attention mechanism allows the model to discern and emphasize specific features from each modality while suppressing others, based on the input provided. The authors demonstrated that their experimental results of this fusion approach significantly enhanced the segmentation performance. Moreover, both encoders are optimized for quicker inference through architectural revisions. The decoder consists of multiple modules, each upscaling the resultant feature maps by a factor of 2, refining features through convolutions, and integrating encoder features. Ultimately, the decoder maps these features to class labels and rescales the class mapping to the original input resolution. The implementation is in PyTorch, with a special emphasis on not using intricate neural architectures and operations to ensure compatibility and practicality for optimized deployment.

The ESANET pre-trained network is utilized with the NYUv2 ResNet34 NBt1D backbone. It maps and segments 40 classes in the scene. The NYUv2 dataset [71] is a semantic segmentation dataset containing a large number of indoor RGBD scenes gathered with a Microsoft Kinect Camera.

The model conversion to IR and the RGBD input-to-inference pipeline are set up in the same way as in earlier steps. However, the asynchronous callback mode, quantization, and the *prepostprocessing* block of OpenVINO are used to run ESANET with minimum latency. There was a notable improvement in performance with the use of the optimization techniques mentioned above (from 2 FPS to 8 FPS). Asynchronous inference is particularly beneficial here because when the OpenVINO Async API is busy with the inference job, the application can perform other tasks in parallel—specifically, gathering RGBD data from the RealSense camera, postprocessing the earlier returned inference, converting RGBD images to point-cloud, and visualizing those. This speeds up the loop computation time, as the application no longer needs to wait for the tasks to complete sequentially. Quantization to FP16 also reduces the computation burden and accelerates the inference by reducing the memory and power consumption (particularly valuable for embedded systems, as in our case). Lastly, instead of using a separate preprocessing function to normalize and scale the RGBD image, this step was incorporated in the IR model itself using the OpenVINO *prepostprocessing* block.

To transform RGBD into a point cloud, the Open3D library is utilized. Specifically, Open3D voxel sampling is employed to reduce the point-cloud size, and point-cloud creation from RGBD images, along with visualization commands, are utilized within the scope of this work. A visual representation of the algorithmic pipeline is shown in Fig. 4.21. Figure 4.22 displays a snapshot of the semantic segmentation. For the real-time

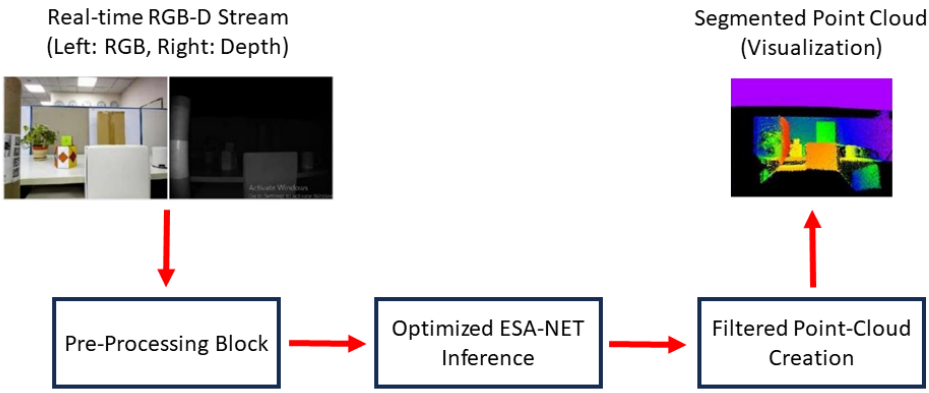


Figure 4.21: Block Level Depiction of RGBD-based point cloud semantic segmentation

demonstration video, please refer to Section 4.4. In Fig. 4.22, each segmented scene color is associated with a semantic label according to NYUv2's metadata format [71]. For example, in the top right corner image of Fig. 4.22, the green (bottom left) represents the floor, blue represents the wall, light green (upper portion of the image) represents the ceiling, and orange represents other structures.

Now, let's discuss the methodologies adopted for point-cloud segmentation. To implement the point-cloud segmentation methodology, Open3D-ML [72] is utilized. Open3D-ML is an extension of Open3D designed to assist users with 3D machine learning tasks, such as semantic cloud segmentation. In essence, it harnesses core Open3D functionalities for 3D data processing and offers various machine-learning frameworks and tools on top of it. The compatibility of Open3D-ML with both PyTorch and TensorFlow adds convenience to integrating Open3D-ML with existing machine-learning implementations. Furthermore, it provides pre-trained models and training pipelines for common 3D machine-learning tasks. Specifically, the implementations of well-known methods like RandLA-Net [73] and KPConv [74] are already included in the Open3D-ML packages.

RandLA-Net and KPConv are two popular models for point cloud processing and analysis. Alongside their benchmarked efficiency and notable performance on various tasks, they exhibit high segmentation accuracies on S3DIS (Stanford 3D Indoor Spaces) and KITTI (Karlsruhe Institute of Technology and Toyota Technological Institute) datasets. Additionally, their open-source availability, commercial licensing, widespread academic and industrial adoption, efficient real-time inference, and cross-dataset compatibility indicate their robustness and generalizability. In most academic settings, both methods are evaluated to find suitable candidates for the given cloud segmentation task due to their underlying differences. Nevertheless, both of these methods represent state-of-the-art approaches in point cloud segmentation, sharing many similarities such as the use of point sampling, local feature aggregation, and computational efficiency.

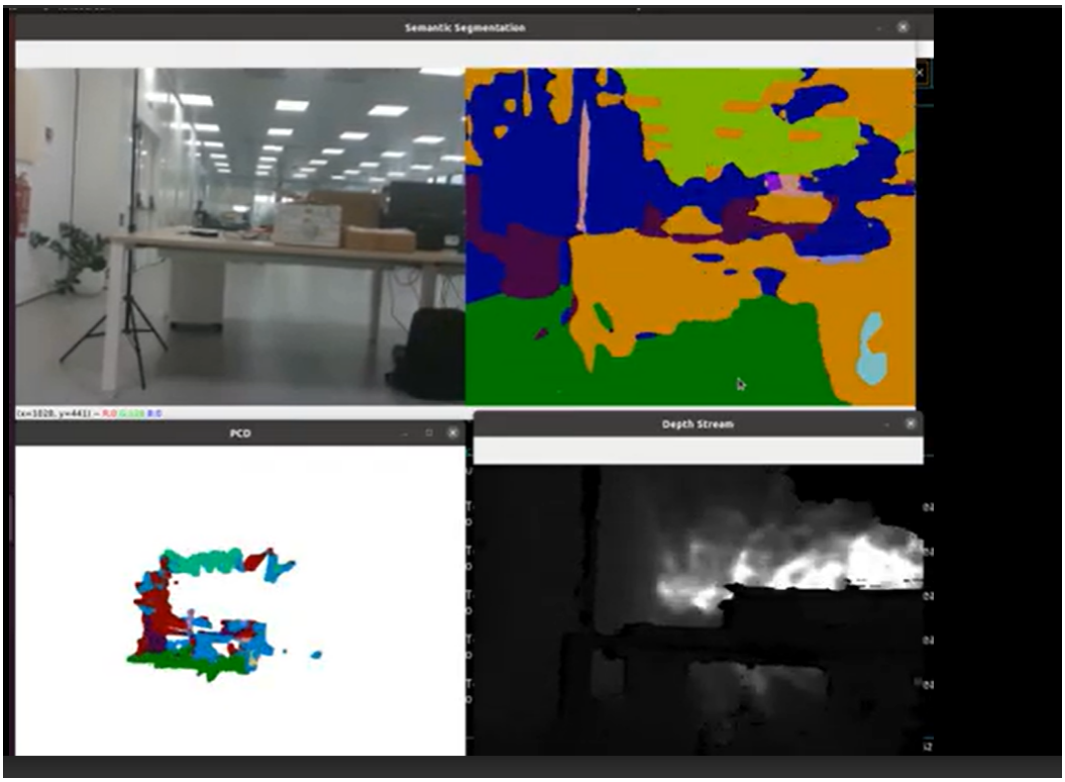


Figure 4.22: A snapshot from the ESANET demo

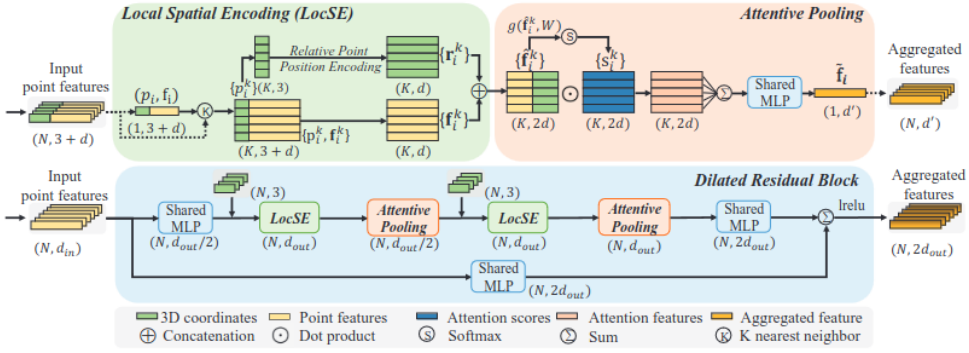


Figure 4.23: Local Feature Aggregation Module Representation [73]

RandLA-Net directly deduces per-point semantics for large-scale point cloud data and does not utilize any intricate point selection method. This approach significantly enhances computation and memory efficiency, although it may unintentionally discard crucial features. One of the key innovations of RandLA-Net addresses this challenge by enabling real-time inference on large-scale point clouds with millions of points spanning up to hundreds of meters. Prior to this method, there was a tradeoff in computational expenses between heuristic or learning-based methods with good coverage versus random sampling-based algorithms.

The RandLA-Net model introduced an innovative local feature aggregating module that gradually expands the receptive field. Specifically, every 3D point is parallelly passed through the local feature aggregation module, comprising Local Spatial Encoding (LocSE), attentive pooling, and dilated residual block. First, LocSE introduces a local context-aware mechanism that enhances the feature representation for each point and generates attention weights based on the local geometric relationships of neighboring points. Second, attentive pooling is used to fuse information from multiple scales with Multi-Layer Perceptrons (MLPs) to generate attention weights for neighboring points within a local region. Unlike earlier approaches that used max/mean pooling [75, 76], this method employs an attention mechanism to capture both fine-grained details and broader context simultaneously. Lastly, the dilated residual block expands the network's receptive field without adding complexity. It utilizes dilated convolutions to efficiently capture broader context, aiding accurate semantic segmentation in complex scenes. A detailed explanation of the local feature aggregating module is provided in Fig. 4.23, taken from [73].

On the other hand, Kernel Point Convolution (KPConv) begins by representing each point in the point cloud as a kernel point (rigid or deformable) and then utilizes these kernel points to convolve and aggregate information from nearby points within a specific user-defined radius. This approach enables the method to capture both local geometric and contextual features in a computationally efficient manner, leading to accurate and faster segmentation of the point cloud data [74].

In the study by [74], two kernel point architectures are introduced: KP-CNN, a five-layered convolutional network designed for classification tasks, and KP-FCNN, a fully convolutional network tailored for segmentation tasks. Both KP-CNN and KP-FCNN involve successive transformations of features during the forward pass, represented by different colors, while points are additionally incorporated to guide and support these operations, as illustrated in Fig. 4.24. The KP-CNN architecture comprises multiple layers, each consisting of two convolutional blocks. In these blocks, the first block is stridden, except for the initial layer. These blocks are structured similarly to bottleneck ResNet blocks [77], incorporating KPConv, batch normalization, and leaky ReLU activation. Following the final layer, features are aggregated using global average pooling, and subsequent fully connected and softmax layers are applied, resembling the architecture of typical image-based CNNs. The network of primary interest, KP-FCNN, employs an encoder similar to that of KP-CNN. To combine features between the encoder and decoder layers, nearest upsampling and skip links are utilized. Concatenated features undergo

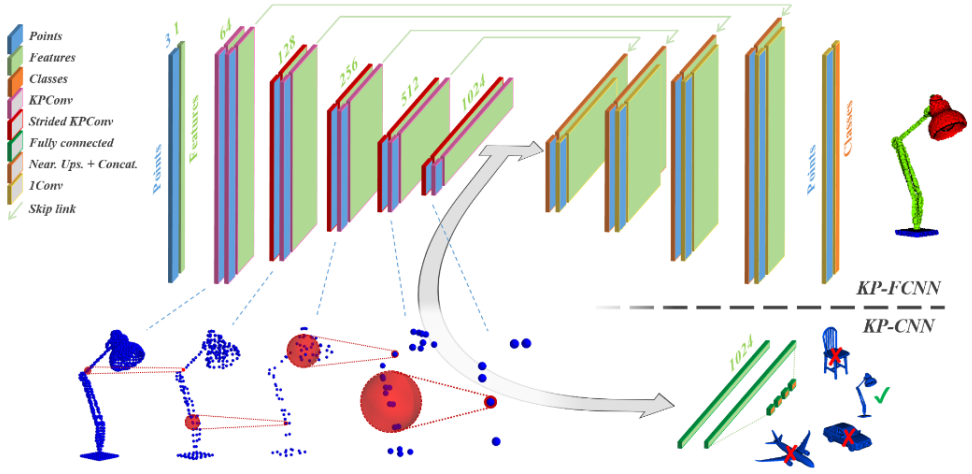


Figure 4.24: The architecture of KP-CNN and KP-FCNN taken from [74]

processing through a unary convolution, similar to a shared MLP or 1x1 convolution.

Pre-trained RandLA-Net and KPConv models on KITTI [78] and S3DIS datasets [79] were selected and utilized in this study due to their impressive publicly-listed evaluation metrics. The KITTI dataset serves as a widely used benchmark for cloud segmentation tasks. This dataset is constructed from real-world 3D point cloud data collected by driving a LIDAR-mounted vehicle in urban environments. It can identify up to 19 classes, including roads, pedestrians, vehicles, and buildings. However, since this dataset focuses on outdoor scenes, models trained on it may not perform optimally in indoor environments. In contrast, the S3DIS dataset serves as a benchmarking dataset for point cloud segmentation tasks, specifically designed to train and evaluate models for indoor scene understanding. It comprises 3D point clouds collected from six indoor areas, including various room types such as offices, classrooms, and conference halls. The points are annotated with 40 different classes including, ceiling, floor, wall, column, beam, and other objects. Unlike the KITTI dataset-trained models, which require only geometric information of each point, models trained on S3DIS require three additional parameters: the R, G, and B values of the point cloud data as input.

Unlike earlier image-based semantic segmentation, here, the dense point cloud is directly captured from the RealSense Camera and then converted into an Open3D-compatible 3D data processing pipeline. The preprocessing involved voxel downsampling and the elimination of incorrect points from the point cloud data. The preprocessed input consists of a three-element key-value pair dictionary (namely, points, feat, and labels). Label values are set to none when passed for inference. Feat (or features) are also set to none in the case of the KITTI dataset. The values of the "point" key are dynamic on the 0th axis, i.e., (?, 3), depending upon the downsampling and density of the point cloud. The "feat" key values would be non-empty and equivalent to the size of the point cloud in case of S3DIS dataset. The output is a dictionary comprising two key-value pairs 'predict_labels' and 'predict_scores,' corresponding to the input

dimension. It is to be noted here that point cloud segmentation requires relatively more computational resources due to several factors such as the large number of points in the point cloud, neighborhood search, high-dimensional features, and a higher number of deep neural network parameters. Therefore, optimizing the real-time performance of the models was of even higher significance. The original speed on Nvidia 3060 GPU was approximately 1 FPS, while after performing OpenVINO-based optimization, it was increased to 3 FPS on iGPU. Utilizing OpenVINO for this purpose was a challenging task as the original implementation of Open3D-ML is incompatible with OpenVINO in terms of model conversion and third-party used packages. The issues related to input/output handling, model tracing, etc., were reported to the authors of the Open3D-ML repository. Nevertheless, self-code debugging and corrections were performed to ensure compatibility within the time constraints of the task. In general, a significant amount of effort was devoted to this activity.

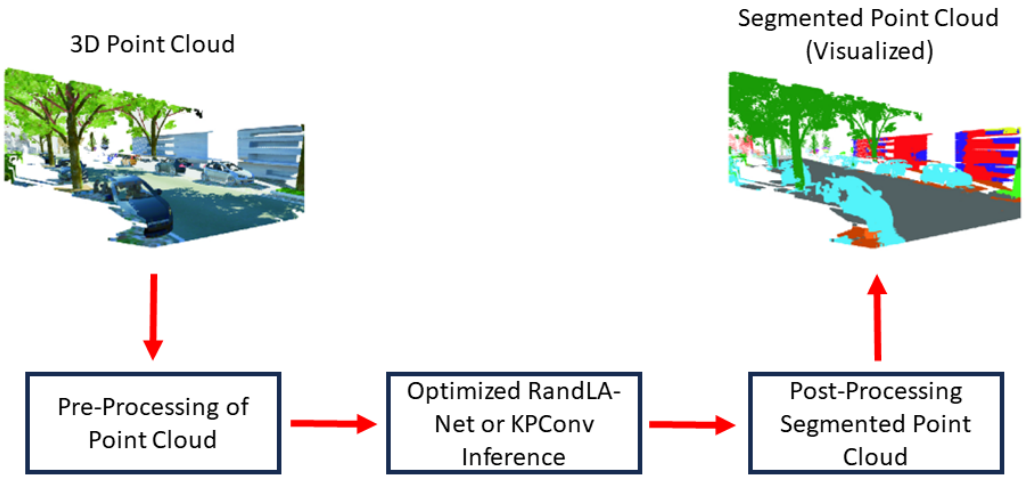


Figure 4.25: Point Cloud semantic segmentation pipeline representation

Figure 4.25 illustrates the point cloud segmentation pipeline with RandLA-Net and KPConv. Figure 4.26 shows a snapshot of the point cloud segmentation (video demonstration is provided in section 4.4). The scene in Fig. 4.26 is the same as in Fig. 4.22, mainly consisting of a table, floor, wall, and ceiling. However, the point cloud data is significantly downsampled to enable real-time processing. It is evident that RandLA-Net has not satisfactorily segmented the scene; only a few blue segmented points are classified as ground. However, it is important to note that the test environment is challenging, featuring reflective floor and ceiling tiles, wall paintings, and clutter.

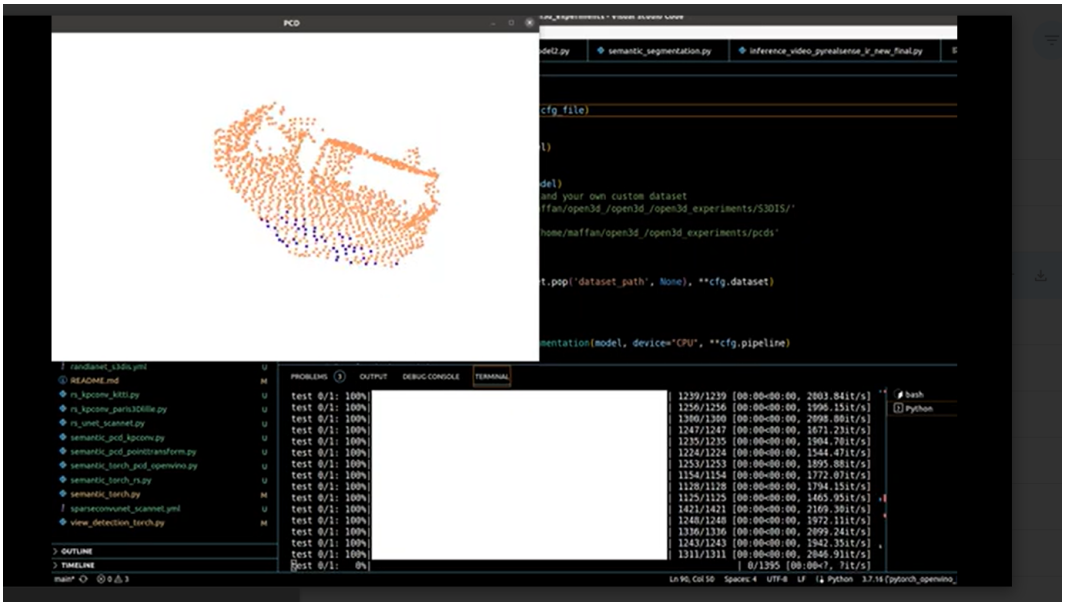


Figure 4.26: A snapshot of the RandLA-Net Demo

4.4 Results

The first objective was to validate the EI for the AMR toolkit. The toolkit was successfully tested; however, it uses Ubuntu 20.04 LTS (or newer), ROS 2 Foxy, and docker-in-docker-based modular packages, which make it infeasible to use with Robotnik's existing software stack. Their software base comprises Ubuntu 18.04, ROS Melodic, and related packages, which were incompatible with the EI for AMR. Likewise, the benefits offered by EI for AMR were not promising enough to justify a complete shift or upgrade to new software versions. However, the key takeaway was the potential applicability of OpenVINO-based pipelines for their mobile robots. The OpenVINO-based methodology was subsequently scaled to the second objective of detecting the docking station, fire, smoke, and objects from the COCO dataset. The simplified pseudocode for object detection ROS wrapper and inference can be referred from Appendices B.1.2 and B.1.2, respectively. All the proposed object detection models were successfully developed and deployed on mobile robots. Object, fire, and smoke detection are entirely new features added to Robotnik's mobile platform. Therefore, their performance is evaluated both via evaluation metrics (visual plots) and empirical testing (demonstration videos). The AI-based docking station detection methodology improves the existing ArUco marker-based approach as it solves the limitations of the conventional image processing-based approach. However, since some of Robotnik's mobile platforms currently utilize the ArUco marker-based approach, a direct real-time performance comparison cannot be provided at this time due to confidentiality reasons. All the video demonstrations of object detection are provided below.

- Docking station detection demo video
- A demo video of object detection from the COCO dataset (Only human detection class is enabled; refer to Section 4.2.2 for more detail)
- Fire detection demo video

The third objective was to filter point cloud data to facilitate indoor/outdoor mobile robot navigation tasks. The initial approaches involved using RANSAC and road-segmentation pre-trained models. These models were tested in real-time with mobile robots at Robotnik Automation, as demonstrated in the first two videos in the following paragraphs.

Upon close analysis of the road segmentation demo, it becomes apparent that real-time results are unsatisfactory under specific conditions. The segmentation is effective only when: 1) the robot moves slowly or remains stationary; its performance deteriorates when the robot moves faster or rotates, 2) the robot is not in a wide open area where neighboring walls are visible or close, and 3) there is a noticeable color difference between the floor and the walls (or other objects on the surface).

As a result, the model's robustness is questionable, making it unsuitable for adoption on a commercial scale. In the second case, RANSAC was integrated and fine-tuned to work in real-time with the target mobile robot. It is important to note that there is an inherent lag between the RGB frame and the point cloud ROS topics of RealSense due to the conversion of the RGBD stream to the point cloud stream.

Upon close analysis of RANSAC's demo, it becomes evident that: 1) points near the ground/floor are not accurately filtered, and 2) small objects on the floor are identified and not filtered along with the floor. In other words, RANSAC can distinguish between objects on the surface and the surface itself (provided the scene is not too close to the robot). However, the results are not satisfactory or reliable enough for commercial-scale use.

This leads us to the topic of semantic segmentation. However, the key consideration in this case was to optimize the methodology sufficiently to run the module in real-time with minimal latency. This objective was achieved through the implementation of ESANet, RandLA-Net, and KPConv-based point cloud segmentation pipelines. The mean Intersection over Union (mIoU) of the pre-trained ESANET was 50.30%, and the original FPS on the CPU was 2, which was enhanced to 8 with OpenVINO. For direct point cloud segmentation, the FPS was doubled, even without quantization and asynchronous code operation. However, the optimized FPS and performance were insufficient to achieve real-time satisfactory results on mobile robots. You can refer to the simplified pseudocode of point cloud segmentation and RGBD Segmentation in Appendices B.2.3 and B.2.4, respectively. Video demonstrations of the segmentation tasks are provided below.

- RANSAC-based dense point-cloud filtering in real-time at Robotnik (Indoor and Warehouse)

- Road Segmentation model-based RGB Video segmentation at Robotnik (Indoor and Warehouse)
- ESANet-based point cloud segmentation
- RandLA-Net point cloud segmentation: Before and After OpenVINO (without quantization and async)

From demo videos of semantic segmentation, it is apparent that ESANET outperforms the other three counterparts and is also compatible with OpenVINO. All the segmented colors displayed in the provided demo videos represent specific semantic labels (as per their original curated format; refer to Section 4.3.2 for more details on each method and their corresponding citations). The scene shown in the RandLA-Net demo video is the same as in ESANET, mainly comprising tables, walls, and floors. It is noticeable that the results, including ESANET, are not ideal for commercial-scale adoption. Therefore, it has been concluded that it is necessary to create Robotnik's own RGBD Segmentation Dataset for scene understanding or segmentation. Point cloud segmentation is an ongoing project at Robotnik. In the future, ESANET models will be trained with custom annotated RGBD images from the robot's field of view to enhance accuracy.

Chapter 5

Conclusions and Future Work

Contents

In this chapter, the conclusions of the work, as well as its future extensions, are shown.

5.1 Conclusions

Working at Robotnik Automation has been an exceptional experience. The problems and challenges encountered during the proposed activities were all real-world scenarios. This opportunity provided me with significant confidence in my skills and insights into the robotics industry's practices, trends, and needs. The collaborative environment enabled me to learn from skilled professionals in different areas of robotics within a limited period. Robotnik Automation provided me with all the necessary resources, including computing hardware, to facilitate my thesis work. However, there were also challenges encountered during this time due to the lack of AI infrastructure at the beginning. In a nutshell, the activities undertaken were the beginning of an era of AI for commercial usage at Robotnik Automation. In addition to my AI-related activities at Robotnik Automation, I also made significant software development contributions at Robotnik Automation in the following areas:

- Migration of the Gazebo simulation of RB-Kairos in ROS Melodic to ROS Noetic using docker container: This involved updating the simulation to work with the

newer version of ROS and dockerizing it so that it could be easily deployed and run on different platforms.

- Containerization of front-end of Open-RMF demos: This involved containerizing the front-end of the Open-RMF demos so that they could be easily deployed and run on different platforms
- Hardware/Software validation and testing of mobile robotic platforms at Robotnik Automation: This involved testing the hardware and software of mobile robotic platforms to ensure that they are working properly and meet the requirements of the customers.

These activities contributed to my overall learning of robotics software engineering by giving me the opportunity to work on a variety of different projects and to learn about the different aspects of developing software for robots.

5.2 Future work

Robotnik Automation is constantly evolving its activities. As of now, the point cloud segmentation methodology is being improved with custom data creation and training. A custom segmentation dataset will be incorporated to enable the segmentation of elevation and small vegetation in outdoor spaces. Work is also underway on broken-fence and open window/door detection, for which a dataset is being created. Specifically, recent YOLO versions and Detectron models will be explored. In short, Robotnik is eager to engage in more and more research and development initiatives focused on robotics and artificial intelligence and to scale them up for commercial use. These projects aim to bridge the gap between different areas of expertise and enable small and medium-sized enterprises (SMEs) to leverage this technology without requiring specialized personnel.

Bibliography

[1] Robotnik webpage. https://robotnik.eu/. Accessed: 2023-08-08.

[2] RB-Watcher. https://robotnik.eu/products/mobile-robots/rb-watcher/. Accessed: 2023-08-08.

[3] RB-Theron. https://robotnik.eu/products/mobile-robots/rb-theron-en/. Accessed: 2023-09-26.

[4] Edge Insights for Autonomous Mobile Robots (EI for AMR). https://www.intel.com/content/www/us/en/developer/topic-technology/edge-5g/edge-solutions/autonomous-mobile-robots.html. Accessed: 2023-10-03.

[5] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[6] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[7] For Intel® distribution of OpenVINO™ toolkit: Intel® distribution of OpenVINO™ toolkit, [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html. Accessed: 2023-10-03.

[8] Verband der Automobilindustrie. Interface for the Communication between Automated Guided Vehicles (AGV) and a Master Control: VDA5050. *VDA: Berlin, Germany*, 2020.

[9] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, 2001.

[10] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005.

[11] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[13] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[15] Ross Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[18] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017.

[19] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single Shot Multibox Detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[20] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[21] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. YOLO by Ultralytics, January 2023.

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[23] Donghao Zhang, Yang Song, Dongnan Liu, Jia Haozhe, Siqi Liu, Yong Xia, Heng Huang, and Weidong Cai. *Panoptic Segmentation with an End-to-End Cell R-CNN for Pathology Image Analysis: 21st International Conference, Granada, Spain, September 16–20, 2018, Proceedings, Part II*, pages 237–244. 09 2018.

[24] Sergi Caelles, Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Laura Leal-Taixé, Daniel Cremers, and Luc Van Gool. One-shot video object segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 221–230, 2017.

[25] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[26] Huikai Wu, Junge Zhang, Kaiqi Huang, Kongming Liang, and Yu Yizhou. Fastfcn: Rethinking dilated convolution in the backbone for semantic segmentation. In *arXiv preprint arXiv:1903.11816*, 2019.

[27] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *IEEE Transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[28] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[29] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[30] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[31] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.

[32] Alexander Kolesnikov, Alexey Dosovitskiy, Dirk Weissenborn, Georg Heigold, Jakob Uszkoreit, Lucas Beyer, Matthias Minderer, Mostafa Dehghani, Neil Houlsby, Sylvain Gelly, Thomas Unterthiner, and Xiaohua Zhai. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[33] Jieneng Chen, Yongyi Lu, Qihang Yu, Xiangde Luo, Ehsan Adeli, Yan Wang, Le Lu, Alan L Yuille, and Yuyin Zhou. Transunet: Transformers make strong encoders for medical image segmentation. *arXiv preprint arXiv:2102.04306*, 2021.

[34] Bowen Cheng, Alex Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation. In *Advances in Neural Information Processing Systems*, volume 34, pages 17864–17875. Curran Associates, Inc., 2021.

[35] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[36] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12159–12168, 2021.

[37] Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. Segmenter: Transformer for semantic segmentation. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7242–7252, 2021.

[38] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. In *Neural Information Processing Systems (NeurIPS)*, 2021.

[39] Mingyu Ding, Zhe Wang, Bolei Zhou, Jianping Shi, Zhiwu Lu, and Ping Luo. Every frame counts: Joint learning of video segmentation and optical flow. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 10713–10720, 2020.

[40] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[41] Gabriel J Brostow, Julien Fauqueur, and Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97, 2009.

[42] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*, pages 102–118. Springer, 2016.

[43] Evan Shelhamer, Kate Rakelly, Judy Hoffman, and Trevor Darrell. Clockwork convnets for video semantic segmentation. In *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III 14*, pages 852–868. Springer, 2016.

[44] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. Deep feature flow for video recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2349–2358, 2017.

[45] Behrooz Mahasseni, Sinisa Todorovic, and Alan Fern. Budget-aware deep semantic video segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1029–1038, 2017.

[46] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, Pablo Martinez-Gonzalez, and Jose Garcia-Rodriguez. A survey on deep learning techniques for image and video semantic segmentation. *Applied Soft Computing*, 70:41–65, 2018.

[47] Tao Wang, Ning Xu, Kean Chen, and Weiyao Lin. End-to-end video instance segmentation via spatial-temporal graph neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10797–10806, 2021.

[48] Shusheng Yang, Xinggang Wang, Yu Li, Yuxin Fang, Jiemin Fang, Wenyu Liu, Xun Zhao, and Ying Shan. Temporally efficient vision transformer for video instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2885–2895, 2022.

[49] Junfeng Wu, Yi Jiang, Song Bai, Wenqing Zhang, and Xiang Bai. Seqformer: Sequential transformer for video instance segmentation. In *European Conference on Computer Vision*, pages 553–569. Springer, 2022.

[50] Intel® RealSense™. *Product Family D400 Series*, 7 2013. Rev. 016.

[51] Robotnik Automation STL. *RB-Watcher*, 6 2023.

[52] OpenCV. Detection of aruco markers. https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html.

[53] Lidar, [Online]. https://en.wikipedia.org/wiki/Lidar, 2023. Accessed: 2023-09-22.

[54] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint arXiv:2207.02696*, 2022.

[55] Chien-Yao Wang, Hong-Yuan Mark Liao, and I-Hau Yeh. Designing network design strategies through gradient path analysis. *arXiv preprint arXiv:2211.04800*, 2022.

[56] Chien-Yao Wang, Hong-Yuan Mark Liao, and I-Hau Yeh. Designing network design strategies through gradient path analysis. *arXiv preprint arXiv:2211.04800*, 2022.

[57] Dwyer B., Nelson (2022) J., Solawetz J., and et. al. Roboflow (Version 1.0) [Software]. Available: https://roboflow.com.

[58] UJI thesis. Docking station detection dataset. https://universe.roboflow.com/uji-thesis/docking-station-detection, feb 2023. [Accessed: 2023-08-22.

[59] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-IoU loss: Faster and better learning for bounding box regression. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 12993–13000, 2020.

[60] Pytorch 1.7.1 documentation. https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html. Accessed: 2023-10-07.

[61] ONNX Runtime developers. Onnx runtime. https://onnxruntime.ai/, 2021. Version: x.y.z.

[62] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. Learning non-maximum suppression. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6469–6477, 2017.

[63] Lutz Roeder. Netron: Visualizer for neural network, deep learning and machine learning models [Software]. https://github.com/lutzroeder/netron.git, 2017.

[64] UJI thesis. Fire dataset. https://universe.roboflow.com/uji-thesis/fire-5lwji, mar 2023. Accessed: 2023-08-22.

[65] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, 1981.

[66] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[67] For Intel's Pre-Trained Models or Public Pre-Trained Models: OpenVINO™ Toolkit - Open Model Zoo repository, [Online]. Available: https://github.com/openvinotoolkit/open_model_zoo. Accessed: 2023-10-07.

[68] Daniel Seichter, Mona Köhler, Benjamin Lewandowski, Tim Wengefeld, and Horst-Michael Gross. Efficient rgb-d semantic segmentation for indoor scene analysis. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 13525–13531, 2021.

[69] Marin Orsic, Ivan Kreso, Petra Bevandic, and Sinisa Segvic. In defense of pre-trained imagenet architectures for real-time semantic segmentation of road-driving images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12607–12616, 2019.

[70] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-Excitation Networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018.

[71] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor Segmentation and Support Inference from RGBD Images. In *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part V 12*, pages 746–760. Springer, 2012.

[72] Open3D-ML: An extension of open3d to address 3d machine learning tasks. https://github.com/isl-org/Open3D-ML, 2018.

[73] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11108–11117, 2020.

[74] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas Guibas. KPConv: Flexible and Deformable Convolution for Point Clouds. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6410–6419, 2019.

[75] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution On X-Transformed Points. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[76] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[77] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[78] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets Robotics: The KITTI Dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[79] I. Armeni, A. Sax, A. R. Zamir, and S. Savarese. Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *ArXiv e-prints*, February 2017.

APPENDIX A

Other considerations

This appendix addresses miscellaneous aspects that have not been included in the main sections of the document.

A.1 Software Dependencies

Below are the developed ROS packages' dependencies on other ROS Noetic Catkin packages.

- Private packages (custom libraries of Robotnik Automation): rcomponent, robotnik_msgs, and yolov7_class
- Public packages: vision_msgs, std_msgs, cv_bridge, sensor_msgs, geometry_msgs, roscpp, rospy

Below is the list of the object detection package dependencies at a level of Python development environment.

- six==1.16.0
- absl-py==1.4.0
- actionlib==1.14.0
- addict==2.4.0
- angles==1.9.13
- argcomplete==1.8.1
- astor==0.8.1
- asttokens==2.2.1
- atomicwrites==1.1.5
- attrs==19.3.0
- autobahn==17.10.1
- Automat==0.8.0

- autopep8==2.0.1
- awscli==1.27.77
- Babel==2.11.0
- backcall==0.2.0
- bcrypt==3.1.7
- beautifulsoup4==4.8.2
- blinker==1.4
- bloom==0.11.2
- bondpy==1.8.6
- botocore==1.29.77
- breezy==3.0.2
- build==0.10.0
- cachetools==5.3.0
- camera-calibration==1.17.0
- catkin==0.8.10
- catkin-pkg==0.5.2
- cbor==1.0.0
- certifi==2019.11.28
- chardet==3.0.4
- charset-normalizer==3.0.1
- Click==7.0
- colcon-argcomplete==0.3.3
- colcon-bash==0.4.2
- colcon-cd==0.1.1
- colcon-cmake==0.2.27
- colcon-common-extensions==0.3.0
- colcon-core==0.12.1
- colcon-defaults==0.2.8
- colcon-devtools==0.2.3
- colcon-library-path==0.2.1
- colcon-metadata==0.2.5
- colcon-notification==0.2.15
- colcon-output==0.2.13
- colcon-override-check==0.0.1
- colcon-package-selection==0.2.10
- colcon-parallel-executor==0.2.4
- colcon-pkg-config==0.1.0
- colcon-powershell==0.3.7
- colcon-python-setup-py==0.2.8
- colcon-recursive-crawl==0.2.1
- colcon-ros==0.3.23
- colcon-test-result==0.3.8
- colcon-zsh==0.4.0
- colorama==0.4.3
- coloredlogs==15.0.1
- comm==0.1.2
- configobj==5.0.6
- constantly==15.1.0
- contourpy==1.0.7
- controller-manager==0.19.6
- coremltools==6.2
- cov-core==1.15.0
- coverage==4.5.2
- cryptography==2.8

- cv-bridge==1.16.2
- cycler==0.10.0
- Cython==0.29.14
- dbus-python==1.2.16
- debugpy==1.6.6
- decorator==5.1.1
- defusedxml==0.7.1
- Deprecated==1.2.7
- diagnostic-analysis==1.11.0
- distlib==0.3.0
- docutils==0.16
- dynamic-reconfigure==1.7.3
- empy==3.3.2
- executing==1.2.0
- fastjsonschema==2.16.3
- filelock==3.9.0
- flatbuffers==23.1.21
- fonttools==4.38.0
- funniest==0.1
- gazebo_plugins==2.9.2
- google-auth==2.16.1
- google-auth-oauthlib==0.4.6
- grpcio==1.51.3
- httplib2==0.14.0
- huggingface-hub==0.13.1
- humanfriendly==10.0
- idna==2.8
- image-geometry==1.16.2
- importlib-metadata==6.0.0
- importlib-resources==5.12.0
- ipykernel==6.21.3
- ipython==7.34.0
- ipython-genutils==0.2.0
- jedi==0.18.2
- Jinja2==3.1.2
- jmespath==1.0.1
- joblib==1.2.0
- joint-state-publisher==1.15.1
- jsonschema==4.17.3
- jstyleson==0.0.2
- jupyter_client==8.0.3
- jupyter_core==5.2.0
- keybert==0.7.0
- keyring==18.0.1
- kiwisolver==1.0.1
- launchpadlib==1.10.13
- lazr.restfulclient==0.14.2
- lazr.uri==1.0.3
- lockfile==0.12.2
- Markdown==3.4.1
- markdown-it-py==2.2.0
- MarkupSafe==2.1.2
- matplotlib==3.7.0
- matplotlib-inline==0.1.6

- mdurl==0.1.2
- more-itertools==4.2.0
- mpmath==1.2.1
- nbformat==5.8.0
- nest-asyncio==1.5.6
- networkx==2.8.8
- nltk==3.8.1
- numpy==1.23.4
- oauthlib==3.1.0
- onnx==1.13.1
- onnxruntime==1.14.1
- onnxsim==0.4.17
- opencv-contrib-python==4.7.0.72
- opencv-python==4.7.0.72
- openvino==2022.3.0
- openvino-dev==2022.3.0
- openvino-telemetry==2022.3.0
- osrf-pycommon==2.0.2
- packaging>=20.3
- pandas==1.3.5
- parso==0.8.3
- pbr==5.4.5
- pexpect==4.6.0
- pickleshare==0.7.5
- Pillow==9.4.0
- pip-autoremove==0.10.0
- pkgutil_resolve_name==1.3.10
- platformdirs==3.1.1
- prettytable==3.6.0
- prompt-toolkit==3.0.37
- protobuf==3.20.3
- psutil==5.5.1
- psycopg2==2.9.5
- pure-eval==0.2.2
- py==1.8.1
- py-cpuinfo==9.0.0
- py-ubjson==0.14.0
- pyasn1==0.4.2
- pyasn1-modules==0.2.1
- pycairo==1.16.2
- Pygments==2.14.0
- PyHamcrest==1.9.0
- PyJWT==1.7.1
- pymacaroons==0.13.0
- pyparsing==3.0.9
- pypng==0.0.20
- PyQRCode==1.2.1
- PyQt5==5.14.1
- pyrealsense2==2.53.1.4623
- pyrsistent==0.19.3
- pytest==4.6.9
- python-dateutil==2.8.2
- python-qt-binding==0.4.4
- PyTrie==0.2

- pytz==2022.7.1
- pyxdg==0.26
- PyYAML==5.4.1
- pyzmq==25.0.1
- qt-dotgraph==0.4.2
- qt-gui==0.4.2
- qt-gui-cpp==0.4.2
- qt-gui-py-common==0.4.2
- regex==2022.10.31
- reportlab==3.5.34
- requests==2.28.2
- requests-oauthlib==1.3.1
- requests-unixsocket==0.2.0
- resource_retriever==1.12.7
- rich==13.3.1
- roman==2.0.0
- rosbag==1.16.0
- rosboost-cfg==1.15.8
- rosclean==1.15.8
- roscreate==1.15.8
- rosdep==0.22.2
- rosdistro==0.9.0
- rosgraph==1.16.0
- rosinstall==0.7.8
- rosinstall-generator==0.1.23
- roslaunch==1.16.0
- roslib==1.15.8
- roslint==0.12.0
- roslz4==1.16.0
- rosmake==1.15.8
- rosmaster==1.16.0
- rosmsg==1.16.0
- rosnode==1.16.0
- rosparam==1.16.0
- rospkg==1.5.0
- rospy==1.16.0
- rosservice==1.16.0
- rostest==1.16.0
- rostopic==1.16.0
- rosunit==1.15.8
- roswtf==1.16.0
- rqt-image-view==0.4.17
- rqt-moveit==0.5.10
- rqt-reconfigure==0.5.5
- rqt-robot-dashboard==0.5.8
- rqt-robot-monitor==0.5.14
- rqt-rviz==0.7.0
- rqt_action==0.4.9
- rqt_bag==0.5.1
- rqt_bag_plugins==0.5.1
- rqt_console==0.4.11
- rqt_dep==0.4.12
- rqt_graph==0.4.14
- rqt_gui==0.5.3

- rqt_gui_py==0.5.3
- rqt_launch==0.4.9
- rqt_logger_level==0.4.11
- rqt_msg==0.4.10
- rqt_nav_view==0.5.7
- rqt_plot==0.4.13
- rqt_pose_view==0.5.11
- rqt_publisher==0.4.10
- rqt_py_common==0.5.3
- rqt_py_console==0.4.10
- rqt_robot_steering==0.5.12
- rqt_runtime_monitor==0.5.9
- rqt_service_caller==0.4.10
- rqt_shell==0.4.11
- rqt_srv==0.4.9
- rqt_tf_tree==0.6.3
- rqt_top==0.4.10
- rqt_topic==0.4.13
- rqt_web==0.4.10
- rsa==4.7.2
- rviz==1.14.20
- s3transfer==0.6.0
- scipy==1.10.1
- seaborn==0.12.2
- SecretStorage==2.3.1
- sensor-msgs==1.13.1
- sentence-transformers==2.2.2
- sentencepiece==0.1.97
- smach==2.5.1
- smach-ros==2.5.1
- smclib==1.8.6
- sympy==1.11.1
- termcolor==2.2.0
- testresources==2.0.1
- texttable==1.6.7
- tf==1.13.2
- tf-conversions==1.13.2
- tf2-geometry-msgs==0.7.6
- tf2-kdl==0.7.6
- tf2-py==0.7.6
- tf2-ros==0.7.6
- thop==0.1.1.post2209072238
- threadpoolctl==3.1.0
- tokenizers==0.13.2
- tomli==2.0.1
- topic-tools==1.16.0
- torch>=1.7.0,!=1.12.0
- torchvision>=0.8.1,!=0.13.0
- #torch==1.11.0+cu113
- #torchvision==0.12.0+cu113
- tornado==6.2
- tqdm==4.64.1
- traitlets==5.9.0
- transformers==4.26.1

- typing_extensions==4.5.0
- urllib3==1.25.8
- vcstool==0.3.0
- vcstools==0.1.42
- wadllib==1.3.3
- wcwidth==0.1.8
- Werkzeug==2.2.3
- xacro==1.14.15
- zipp==3.14.0
- zope.interface==4.7.1

Below is the list of the scene segmentation package dependencies at a level of Python development environment. It is to be noted here that many of the package dependencies are conflictive between object detection and scene segmentation, while some are overlapping. Therefore. a Conda virtual environment is used for development purposes.

- absl-py==0.8.1
- actionlib==1.14.0
- addict==2.4.0
- aiofiles==22.1.0
- aiosqlite==0.19.0
- angles==1.9.13
- anyio==3.7.1
- appdirs==1.4.4
- argon2-cffi==21.3.0
- argon2-cffi-bindings==21.2.0
- astor==0.8.0
- astroid==2.3.2
- asttokens==2.2.1
- attrs==19.3.0
- Babel==2.12.1
- backcall==0.2.0
- base-local-planner==1.17.3
- beautifulsoup4==4.12.2
- bleach==6.0.0
- bondpy==1.8.6
- camera-calibration==1.17.0
- catkin==0.8.10
- catkin-virtualenv==0.6.1
- certifi==2023.5.7
- cffi==1.13.1
- chardet==3.0.4
- charset-normalizer==3.2.0
- cityscapesScripts==1.5.0
- cloudpickle==1.2.2
- controller-manager==0.19.6
- controller-manager-msgs==0.19.6
- cv-bridge==1.16.2
- cycler==0.10.0
- Cython==0.29.36
- cytoolz==0.10.1
- dask==2.7.0
- debugpy==1.6.7
- decorator==4.4.1

- defusedxml==0.7.1
- deprecation==2.1.0
- devtools==0.11.0
- diagnostic-analysis==1.11.0
- diagnostic-updater==1.11.0
- dill==0.3.1.1
- dynamic-reconfigure==1.7.3
- entrypoints==0.4
- exceptiongroup==1.1.2
- executing==1.2.0
- fastjsonschema==2.17.1
- future==0.18.2
- gast==0.2.2
- gazebo-plugins==2.9.2
- gazebo-ros==2.9.2
- gencpp==0.7.0
- geneus==3.0.0
- genlisp==0.4.18
- genmsg==0.6.0
- gennodejs==2.0.2
- genpy==0.6.15
- google-pasta==0.1.8
- grpcio==1.25.0
- h5py==2.9.0
- idna==2.8
- image-geometry==1.16.2
- imageio==2.6.1
- importlib-metadata==6.7.0
- importlib-resources==5.12.0
- interactive-markers==1.12.0
- ipykernel==6.16.2
- ipython==7.34.0
- ipython-genutils==0.2.0
- ipywidgets==8.0.7
- isort==4.3.21
- jedi==0.18.2
- Jinja2==3.1.2
- joblib==1.3.1
- joint-state-publisher==1.15.1
- json5==0.9.14
- jsonschema==4.17.3
- jstyleson==0.0.2
- jupyter-client==7.4.9
- jupyter-core==4.12.0
- jupyter-events==0.6.3
- jupyter-packaging==0.12.3
- jupyter-server==1.24.0
- jupyter-server-fileid==0.9.0
- jupyter-server-ydoc==0.8.0
- jupyter-ydoc==0.2.4
- jupyterlab==3.6.5
- jupyterlab-pygments==0.2.2
- jupyterlab-server==2.23.0
- jupyterlab-widgets==3.0.8

- Keras-Applications==1.0.8
- Keras-Preprocessing==1.1.0
- kiwisolver==1.1.0
- laser-geometry==1.6.7
- lazy-object-proxy==1.4.2
- Markdown==3.1.1
- MarkupSafe==2.1.3
- matplotlib==3.1.1
- matplotlib-inline==0.1.6
- mccabe==0.6.1
- message-filters==1.16.0
- mistune==3.0.1
- mkl-fft==1.0.14
- mkl-random==1.1.0
- mkl-service==2.3.0
- mock==4.0.1
- moveit-commander==1.1.12
- moveit-core==1.1.12
- nbclassic==1.0.0
- nbclient==0.7.4
- nbconvert==7.6.0
- nbformat==5.8.0
- nest-asyncio==1.5.6
- netron==7.0.3
- networkx==2.4
- notebook==6.5.4
- notebook-shim==0.2.3
- numpy==1.21.6
- olefile==0.46
- onnx==1.6.0
- onnxruntime==1.0.0
- open3d==0.13.0
- opencv-python==4.8.0.74
- openvino==2022.3.1
- openvino-dev==2022.3.1
- openvino-telemetry==2023.0.0
- opt-einsum==3.1.0
- packaging==23.1
- pandas==1.3.5
- pandocfilters==1.5.0
- parso==0.8.3
- pexpect==4.8.0
- pickleshare==0.7.5
- Pillow==8.2.0
- pkgutil-resolve-name==1.3.10
- prometheus-client==0.17.1
- promise==2.3
- prompt-toolkit==3.0.39
- protobuf==3.10.0
- psutil==5.9.5
- ptyprocess==0.7.0
- pycparser==2.19
- Pygments==2.15.1
- pylint==2.4.3

- pyparsing==2.4.2
- pyrealsense2==2.53.1.4623
- pyrsistent==0.19.3
- python-dateutil==2.8.0
- python-json-logger==2.0.7
- python-qt-binding==0.4.4
- pytorch-ignite==0.2.1
- pytz==2019.3
- PyWavelets==1.1.1
- PyYAML==6.0
- pyzmq==25.1.0
- qt-dotgraph==0.4.2
- qt-gui==0.4.2
- qt-gui-cpp==0.4.2
- qt-gui-py-common==0.4.2
- requests==2.31.0
- resource-retriever==1.12.7
- rfc3339-validator==0.1.4
- rfc3986-validator==0.1.1
- rosbag==1.16.0
- rosboost-cfg==1.15.8
- rosclean==1.15.8
- roscreate==1.15.8
- rosgraph==1.16.0
- roslaunch==1.16.0
- roslib==1.15.8
- roslint==0.12.0
- roslz4==1.16.0
- rosmake==1.15.8
- rosmaster==1.16.0
- rosmsg==1.16.0
- rosnode==1.16.0
- rosparam==1.16.0
- rospy==1.16.0
- rosservice==1.16.0
- rostest==1.16.0
- rostopic==1.16.0
- rosunit==1.15.8
- roswtf==1.16.0
- rqt-action==0.4.9
- rqt-bag==0.5.1
- rqt-bag-plugins==0.5.1
- rqt-console==0.4.11
- rqt-dep==0.4.12
- rqt-graph==0.4.14
- rqt-gui==0.5.3
- rqt-gui-py==0.5.3
- rqt-image-view==0.4.17
- rqt-launch==0.4.9
- rqt-logger-level==0.4.11
- rqt-moveit==0.5.10
- rqt-msg==0.4.10
- rqt-nav-view==0.5.7
- rqt-plot==0.4.13

- rqt-pose-view==0.5.11
- rqt-publisher==0.4.10
- rqt-py-common==0.5.3
- rqt-py-console==0.4.10
- rqt-reconfigure==0.5.5
- rqt-robot-dashboard==0.5.8
- rqt-robot-monitor==0.5.14
- rqt-robot-steering==0.5.12
- rqt-runtime-monitor==0.5.9
- rqt-rviz==0.7.0
- rqt-service-caller==0.4.10
- rqt-shell==0.4.11
- rqt-srv==0.4.9
- rqt-tf-tree==0.6.3
- rqt-top==0.4.10
- rqt-topic==0.4.13
- rqt-web==0.4.10
- rviz==1.14.20
- scikit-image==0.14.2
- scikit-learn==1.0.2
- scipy==1.7.3
- Send2Trash==1.8.2
- sensor-msgs==1.13.1
- six==1.12.0
- smach==2.5.1
- smach-ros==2.5.1
- smclib==1.8.6
- sniffio==1.3.0
- soupsieve==2.4.1
- srdfdom==0.6.4
- tensorboard==1.15.0
- tensorboardX==1.9
- tensorflow-datasets==1.3.2
- tensorflow-estimator==1.15.1
- tensorflow-gpu==1.15.0
- tensorflow-metadata==0.21.0
- termcolor==1.1.0
- terminado==0.17.1
- texttable==1.6.7
- tf==1.13.2
- tf-conversions==1.13.2
- tf2-geometry-msgs==0.7.6
- tf2-kdl==0.7.6
- tf2-py==0.7.6
- tf2-ros==0.7.6
- tf2-sensor-msgs==0.7.6
- threadpoolctl==3.1.0
- tinycss2==1.2.1
- tomli==2.0.1
- tomlkit==0.11.8
- toolz==0.10.0
- topic-tools==1.16.0
- torch==1.3.1
- torchvision==0.4.2

- tornado==6.2
- tqdm==4.65.0
- traitlets==5.9.0
- typing-extensions==4.7.1
- urdfdom-py==0.4.6
- urllib3==1.25.7
- wcwidth==0.2.6
- webencodings==0.5.1
- websocket-client==1.6.1
- Werkzeug==0.16.0
- widgetsnbextension==4.0.8
- wrapt==1.11.2
- xacro==1.14.16
- y-py==0.5.9
- ypy-websocket==0.8.4
- zipp==3.15.0

Appendix B

Source code

Due to the confidential/Commercial nature of the work, the source code of the implemented solutions can not be provided. However, for reference, a simplified pseudocode is provided.

B.1 Object detection

Algorithm B.1.1 *ROS wrapper class for YOLO-V7 inference*

1. Initialize the ROS node and the YOLOv7 inference engine.
2. Subscribe to the input image topic.
3. In the image callback function,
4. Convert the ROS image message to a NumPy array.
5. Run the YOLOv7 inference engine on the image.
6. Get the bounding boxes, scores, and classes of the detected objects.
7. Create a ROS message for the bounding boxes.
8. Publish the bounding boxes message and the image message.

Algorithm B.1.2 *Yolo v7 inference class*

1. Initialize the YOLOv7 inference class with the following parameters
   img_size ▷ The size of the input image
   conf_thres ▷ The confidence threshold for object detection
   iou_thres ▷ The intersection-over-union threshold for object detection
   device ▷ The device on which to run the inference
   detection_type ▷ The type of object detection model to use
   COI ▷ The list of classes of interest

2. Load the YOLOv7 weights and model files.

3. Create a dictionary of colors for each class of interest.

4. Convert the ROS image message to a NumPy array.

5. For each input image:

   a) Convert the image to a NumPy array.

   b) Perform letterboxing on the image to resize it to the desired input size.

   c) Transpose of the image and expand its dimensions to make it compatible with the YOLOv7 model.

   d) Normalize the image to the range [0, 1].

   e) Pass the image through the YOLOv7 model to obtain the detections.

   f) For each detection, if the class of the detection is in the list of classes of interest, then:

      i. Calculate the bounding box coordinates of the detection.

      ii. Draw the bounding box and the class label on the image.

6. Return the image with the bounding boxes drawn on it.

B.2 Semantic Segmentation

Algorithm B.2.3 *Point cloud Segmentation*

1. Import necessary libraries and modules, including *pyrealsense*2, *numpy*, *numpy*,and *open*3*d*.
2. Define a dictionary $COLOR\_MAP$ that maps class labels to their corresponding RGB colors.
3. Create a dictionary of colors for each class of interest.
4. Define a $s3dis\_labels$ dictionary for labeling point cloud data.
5. Load ML configuration from a YAML file
   ▷ Configuration details of dataset, model, and pipeline
6. Initialize the RandLANet/KPConv model.
7. Load and Optimized the RandLANet/KPConv model via OpenVINO.
8. Initialize the Realsense context and query connected devices.
9. Create a Realsense pipeline and configure it for depth and color streams.
   a) Wait for frames from the Realsense camera.
   b) Get depth and color frames from the pipeline.
   c) Create an Open3D RGBDImage from the color and depth frames.
   d) Create a PointCloud from the RGBDImage using intrinsic parameters.
      ▷ This step is specific to Realsense as it is used to create pointcloud
   e) Downsample and transform the PointCloud for processing.
   f) Prepare the PointCloud for inference by removing NaNs and infinities.
   g) Run inference on the prepared data using the RandLANet/KPConv model.
   h) Color the PointCloud based on predicted labels using the $COLOR\_MAP$.
   i) Initialize a visualizer and add the colored PointCloud.
   j) Update the visualizer and display the PointCloud.
10. Handle pipeline termination and window destruction.

Algorithm B.2.4 *RGBD-based Segmentation*

1. Import necessary libraries and modules, including *pyrealsense*2, *numpy*, *cv*2, and *open*3*d*.
2. Define global variables and functions for visualization and post-processing.
3. Set up the Realsense pipeline, configure the camera streams, and enable alignment of depth to the color.
4. Initialize the OpenVINO core and load the compiled model for inference.
5. Define a custom preprocessing function for input samples.
6. Create an asynchronous inference queue with a callback function.
7. Enter a loop for real-time processing of frames from the camera.
   a) Wait for color and depth frames from the Realsense camera.
   b) Preprocess the frames by applying custom normalization and preparing input tensors.
   c) Start asynchronous inference using the compiled OpenVINO model.
   d) Perform post-processing to obtain segmented images.
   e) Convert the segmented image to a PointCloud using Open3D.
   f) Visualize or publish the segmented image and PointCloud in a window.
   g) Display the visualized images and PointCloud and handle user input.
8. Handle pipeline termination and window destruction.