SINGAPORE INSTITUTE OF TECHNOLOGY
Design Project Report Submission Form
Declaration of Authorship

Name of candidate: ***Rishivaran S/O Veerappan***	Student ID Number: ***1801441***
Degree: ***BEng Telematics (Intelligent Transportation Systems Engineering)***	
Design Project Title: ***Obstacle Avoidance using AI***	
Academic Supervisor(s): ***Assistant Professor Peter Waszecki***	Cluster: ***ICT***
Industry Supervisor: (*if applicable*)	Organization: (*if applicable*)

I hereby confirm that:

1. this work was done wholly or mainly while in candidature for a degree programme at SIT;
2. where any part of this design project has been previously submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. I have acknowledged the use of all resources in the preparation of this report;
4. the **report contains** / ~~**does not contain**~~ patentable or confidential information *(strike through whichever does not apply)*;
5. the work was conducted in accordance with the research integrity policy and ethics standards of SIT and that the research data are presented honestly and without prejudice. The SIT Institutional Review Board (IRB) approval number is ___________________ *(where applicable)*;
6. I have read and understood the University's definition of plagiarism as stated in the SIT Academic Policies Scheme 14: Academic Integrity.

   Plagiarism is the copying, using or passing off of another's work as one's own work without giving credit to the author or originator, and also includes self-plagiarism. For example, reusing, wholly or partially, one's previous work in another context without referencing its previous use.

[illegible]
Signature of Student

08/08/2021
Date

Design Project - Final Report

Obstacle Avoidance using AI

Rishivaran S/O Veerappan

1801441

Academic Supervisor:
Assistant Professor Peter Waszecki

Submitted as part of the requirements for TLM3001 Design Project

Acknowledgement

I would like to thank my Academic Supervisor A/P Peter Waszecki for his advice, support, and suggestions throughout the project. I would like to thank Dr. Edwin Foo and Dr. Michael Lau for their invaluable guidance and accepting me into their team to work on the Seveur Robot project. It was new experience for me in the world of AI, and I appreciate their knowledge sharing and support throughout the project.

Abstract

This report will discuss the obstacle avoidance feature developed for a developing waiter robot. The obstacle avoidance feature was incorporated with the Artificial Intelligence (AI) concept. The project aimed to provide a dynamic approach to obstacle avoidance when compared to the previous implemented solution. Firstly, an introduction to service robotics and its benefits in modern society will be studied. Furthermore, details on the developing waiter robot and the project's objective and solution will be discussed.

Next, the literature research will be examined on the different dining environments, the current implementations of waiter robots in eateries, and the technologies used in the project's design. After the examination of the research, the conceptual design approach of the project will be further discussed. The obstacle avoidance framework and how it can be incorporated with the robot's movement would be explored.

Further, into the report, the implementation steps will be analysed. How the object detection framework was built, and the integration of the object detection framework with the Robot Operating System (ROS) will be examined. Along with the implementation, the tests conducted will be reviewed. The various tests performed on the object detection framework will be explained, and the output of the tests will be discussed.

The improvements made to the design of the object detection framework will be reviewed further in the report. The improvements made will be compared with the initial tests conducted, and the results will be discussed. Furthermore, the errors made during the implementation stages and the challenges faced throughout the project phase will be explained. Finally, the recommendations for the future work that can be done for the project will be discussed, and the report will end with a conclusion.

Table of Contents

1. Introduction

The world is ever progressing in various ways. Technological advancement is one of the critical advancements in the modern world. Within the technological advancement is the implementation of robotic solutions. Robotics is an increasing trend in various industries as the world moves towards implementing intelligent technologies for the services in their industries. When adopting autonomous robotic solutions, productivity, efficiency, and cost of operation can improve compared to human labour [1]. Service robots are a type of robot used in a more professional setting, and they help alleviate mundane, repetitive, and dangerous tasks currently being done by humans. The use of service robots will free up human resources and allow humans to focus on more complex and cognitive tasks. There are many areas whereby service robots can be implemented, such as healthcare, food industry, construction, and other sectors. According to a report by FortuneBusinessInsight, the market size for service robotics in 2019 was estimated at USD 12 Billion. It will eventually increase further to about USD 47 Billion by 2027 [2]. Figure 1 showcases the increasing trend of the service robotics solution in Europe.

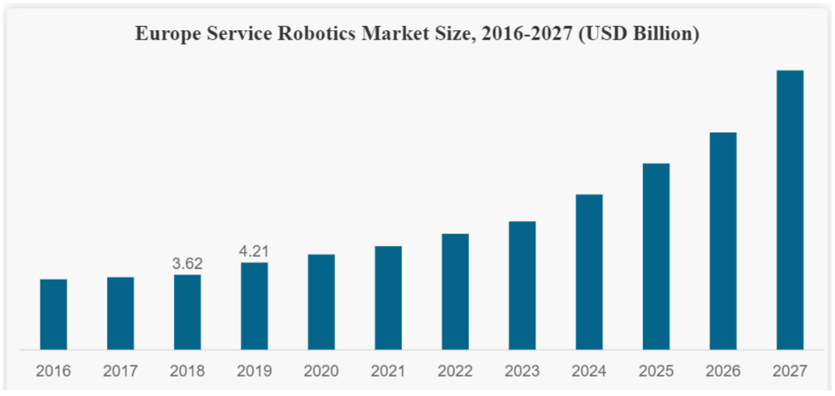


Figure 1. Market trend of service robotics solution

Service robots can be implemented in various industries. The Food and Beverage (F&B) industry is where the robotics solution can benefit users and owners. In the F&B sector, service robots serve food and drinks to patrons in an eatery. Eating outlets in countries like China, Netherlands, Japan have already introduced such robots [3,4]. The robots can be used in dining outlets to serve food and drinks to patrons in a secure manner compared to humans by reducing typical human mistakes, such as stumbling or falling. Another benefit would be reducing the number of wrong orders served by the waiters.

Furthermore, implementing robotic solutions in the food industry will be very useful if the world is experiencing a virus pandemic whereby reducing human interaction is extensively advised. Therefore, the service robots can deliver food in the dining outlet, replacing the need for human waiters to interact with patrons. Introducing such solutions ensures patrons receive an added level of safety and, in the process, increases the outlet's reputation.

1.1 Problem Description

Navigation is an essential aspect of a service robot. Dining locations tend to contain various obstacles for a robot to navigate. With fixed obstacles like tables and chairs and dynamic obstacles such as humans, the robot must avoid colliding with any of the obstacles in order to perform its task correctly. The navigation process for the service robot has to be safe and efficient. Moreover, it must be able to avoid both fixed and dynamic obstacles in its path.

A combined team from NU, SIT, and NYP are pursuing a waiter robot (Serveur Robot), as seen in Figure 2. This robot would be the platform to incorporate the project. The robot was built with a 3-Swedish 90º wheelbase for omnidirectional travel. It was developed with the Robotic Operating System (ROS) for the autonomous systems using Simultaneous Localization and Mapping (SLAM). Obstacle detection is based on Light Detection and Ranging (LiDAR) sensor to detect any obstacle in its path while navigating [5].

The current implementation of the navigation process on the Serveur robot uses the SLAM and sensor capabilities. A challenge with the current implementation is the detection of different kinds of obstacles. In the current implementation, the LiDAR sensor is used to detect any obstacles along the path. However, the sensor is not able to distinguish between the different obstacles. The proposed area of implementation of the robot is in a restaurant environment. Therefore, there would be various kinds of obstacles present in such an environment. By detecting different types of obstacles, the robot's travelling speed can be reduced depending on the type of obstacle it detected. Static obstacles will have a fixed location. Hence, the robot's speed while avoiding need not be too slow. As for the dynamic obstacles, they tend to be on the move. Thus, the velocity of the robot while avoiding can be slower than avoiding static obstacles. The reason for the speed configuration for dynamic obstacle avoidance would be the unpredictable movement of a dynamic obstacle, thus, the higher probability of collision.

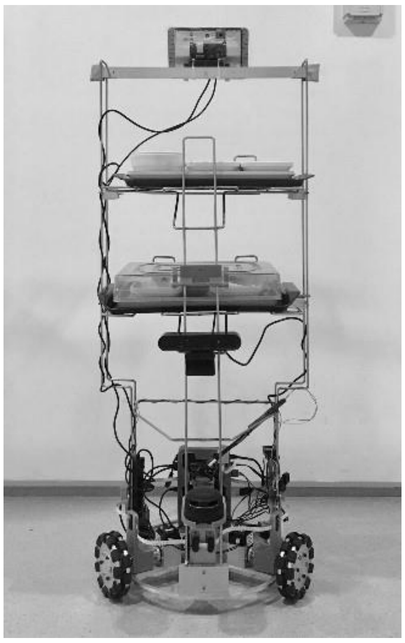


Figure 2. Serveur Robot

1.2 Objective

This project aims to establish dynamic obstacle avoidance using AI, where the feature can be incorporated into a developing waiter robot. With the implementation of the Deep Learning method, a subset of AI, the obstacle avoidance feature will ensure the waiter robot is efficient and intelligent when moving around a dining environment. The Deep Learning framework will have to integrate with ROS to detect different obstacles in its path while navigating. As the waiter robot is based on ROS for the robot navigation, the framework should be able to provide information to the robot movement on the detection with the aim to reduce the speed and avoid the obstacle.

2. Literature Research

2.1 Dining outlets

There are various dining outlets, as seen in Figure 3, such as casual dining, fast-food dining, and fine dining outlets. However, not all outlets can accommodate service robots. The usual roles service robots will partake in the F&B industry are serving, cooking, bartending, and cleaning. Even though the implementation of the robots can help ease specific workloads in a dining outlet, not all outlets may be suited, for example, fine dining restaurants, where the customer relationship is an essential aspect of the fine dining experience. There, patrons may prefer the human touch, which includes explaining the menu items to the patrons, the level of service provided to them, and food prepared by the chef. However, casual dining restaurants or fast-food outlets do not significantly emphasize service and are more flexible and informal. Having less human interaction in terms of service and order would be acceptable in these outlets.

Therefore, introducing the waiter robot to a Casual Dining/Fast-food restaurant would be ideal. The patrons will be able to receive the food in a timely and safe manner. With the majority of people dining in such outlets due to the low cost and quick service of food, traffic will be high. Thus, the Waiter robot can focus on delivering food and beverages, enabling the staff to concentrate more on other activities, such as food preparation.

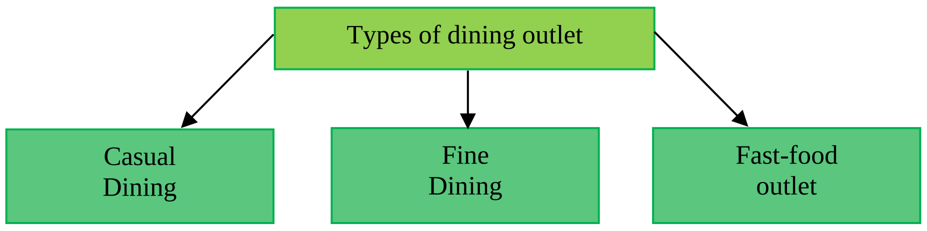


Figure 3. Types of dining outlets

2.2 Current implementation in dining outlets

In 2016, Rong Hong restaurant was one of the first in Singapore to implement robots to serve the customers [6]. Figure 4 shows the robot serving patrons in the restaurant. The robot used magnetic strips for navigation, and sensors were used for obstacle detection with a detection range of 15cm. The robot was able to communicate with the customers when delivering their food and beverages.

The management widely appreciated the implementation of the robot as it was able to ease the cost of operation, hire more staff, and the robots attracted more customers to the restaurant. The service staff appreciated the added help they got from the robots as it eased their workload. The use of magnetic strips for navigation restricts the robot's movement, and maintenance has to be considered for the strips along with the robot.

Figure 4. Waiter robot service at a restaurant in Singapore

At the Royal Palace Renesse restaurant located in the Netherlands, robot waiters were introduced to reduce the human interaction between staff and patrons during the Covid-19 pandemic [7]. As seen in Figure 5, the red and white robots would greet customers, serve food, and pick up dishes. With the introduction of the robots, the restaurant did not need to worry about infection within staff and service being affected as robots cannot be infected.

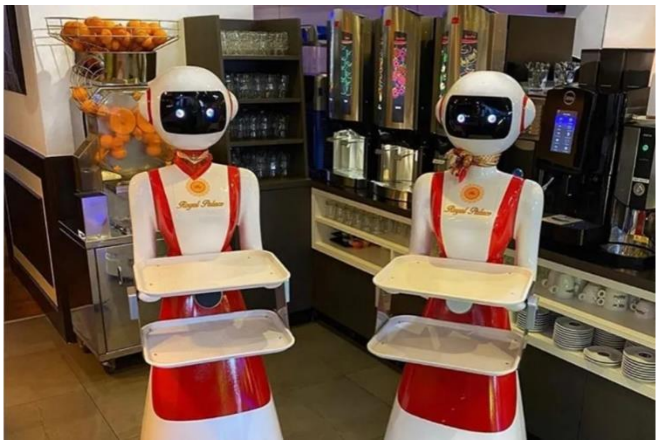


Figure 5. Robot waiters in Royal Palace Renesse restaurant

2.3 System Overview

The System Overview section will discuss the technologies that can be incorporated into the system design.

2.3.1 NVIDIA Jetson Nano Development Kit

The Jetson Nano is a minicomputer, as seen in Figure 6, capable of being used for singlepurpose computing or a handful of tasks with relatively low hardware requirements [8]. The Jetson Nano is built around a 128 Core GPU using Nvidia's Maxwell architecture, capable of 472GFLOPS. It allows running multiple neural networks in parallel for image classification, object detection, segmentation, and speech processing applications. Furthermore, AI algorithms can be efficiently processed with high computational power.

The Jetson Nano implements NVIDIA Jetpack SDK for AI application development. According to the NVIDIA documentation [9], the SDK consists of the OS (UBUNTU Linux OS), libraries, APIs, developer tools, and more. The libraries consist of TensorRT and cuDNN for high-performance deep learning, CUDA for GPU acceleration, and VisionWorks, OpenCV, VPI for visual computing.

Figure 6. Nvidia Jetson Nano

The Jetson Nano can be used as the central hub for the obstacle avoidance feature. Since the feature is based on AI, training the detection model and testing the detection capabilities can be done using the Jetson Nano.

2.3.2 Deep Learning Frameworks

Deep Learning is an AI function that mimics the human brain in processing data and creating patterns for decision making, and decisions are made on unstructured data without supervision [10]. There are many different Deep Learning Frameworks. The two more popular frameworks being used for Deep Learning are TensorFlow and PyTorch.

<u>TensorFlow</u>

Created by the Google Brain team and released in 2015, TensorFlow was developed to conduct machine learning and deep neural networks research. TensorFlow creates dataflow graph structures that detail how data travels through a graph or a series of processing nodes.

Each node represents a mathematical operation, and each connection between nodes represents a tensor (multidimensional data array) [11]. The benefit of TensorFlow is abstraction. The developer can focus on the overall logic of the application since details of implementing algorithms or finding ways to integrate the output of one function to the input of another will be handled by the framework itself.

PyTorch

PyTorch was developed by Facebook's AI research lab and released in 2016, and similar to TensorFlow, it was built for deep learning research. PyTorch is known for its flexibility and speed and is a preferred method to develop project prototypes quickly [11]. It has two main features: Tensor computation (like NumPy) with strong GPU acceleration and automatic differentiation for building and training neural networks [12]. PyTorch defines its computational graph dynamically, unlike TensorFlow. Hence, the graph can be manipulated on the go [13]. PyTorch is integrated with the NVIDIA Jetson Nano via the Jetpack SDK, making it readily accessible to use after the setup was done.

Both frameworks are equally reliable to use. Since the PyTorch framework is integrated with the Jetpack SDK in the Nvidia Jetson Nano, it will make it readily accessible to conduct object detection training for the obstacle avoidance feature.

2.3.3 MobileNet-SSD

MobileNet is an efficient network architecture whereby the model is small, low-powered, and has low latency. It provides high accuracy and is applicable for applications ranging from object detection, classification, tracking, and others. MobileNet model is based on depthwise separable convolution. Depthwise separable convolution is a type of factorized convolution whereby it factorizes a standard convolution to a depthwise convolution, applies a single filter per input channel (input depth), and a pointwise convolution to create a linear combination of the output of the depthwise layer. Additionally, MobileNet uses both batchnorm and ReLU nonlinearities for both layers [14].

Single Shot Multibox Detector (SSD) is a method based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes [15]. It extracts feature maps and applies convolution filters to detect the object. Integrating MobileNet with the SSD framework is the significant benefit of addressing the limitations of running high-resource and power-consuming neural networks on low-end devices in real-time applications.

Figure 7 showcases how the MobileNet-SSD model operates where an image is fed through to the MobileNet base network and then to the SSD network. With the benefit of integrating the MobileNet base network and SSD framework, the model can be applied to object detection training for the project.

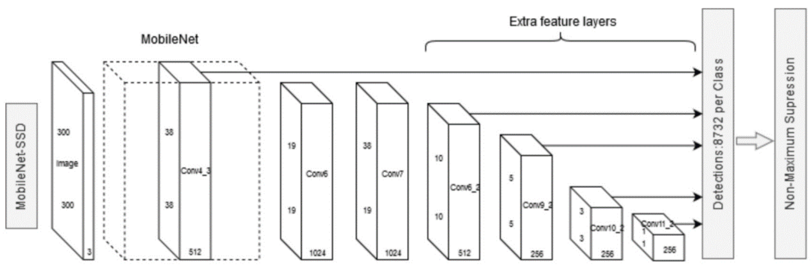


Figure 7. MobileNet-SSD model

2.3.4 ROS

Robot Operating System (ROS) is a framework for robot software [16]. It contains various tools and libraries for building, writing, and obtaining codes. Similar to any operating system, ROS provides services like hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. ROS only runs on Unix-based platforms and is used primarily on Ubuntu and Mac operating systems. Python or C++ languages are used to write the source and scripts, while Extensible Markup Language (XML) is used to write launch files. Launch files are various sources of files, programming scripts clustered into a single file.

ROS nodes are executable files within a ROS package that can communicate with other nodes. The communication channel is either via publishing or subscribing to a topic. The publisher is the node that sends out information of a message type to a topic. At the same time, the subscriber is the node that receives the information from the topic [17]. Figure 8 represents how a publisher and subscriber works by which the publisher node sends out a message with type String (std_msg/String) to the topic example (/example). Subsequently, the various subscriber nodes subscribe to the topic example (/example) and receive the message.

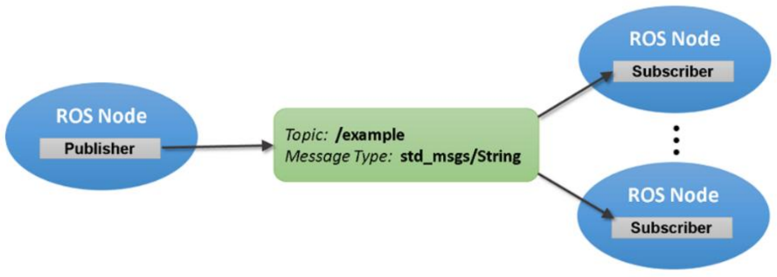


Figure 8. Communication between different nodes

ROS framework can be used to bridge the object detection model and the robot platform. As the information is publishing from a live feed, the ROS framework can feed the information to the robot navigation module. Hence, incorporating the object detection model with ROS can enable robot movement control based on the information provided.

3. Conceptual Design

In this section of the report, the planned design of the system will be discussed.

3.1 Establishing Obstacle Detection

Object detection can be established to distinguish the type of observed obstacle. Objects can be categorized into two kinds, dynamic and static. Dynamic objects will define obstacles that are not stationary and tend to move around, such as waiters or patrons. Static objects will define obstacles that are stationary, such as tables and chairs in a restaurant.

A custom dataset has to be generated in order to establish a custom object detection to determine dynamic and static obstacles. Upon gathering the images for the custom dataset, annotation (labelling) must be done on the images. Annotating the Image will produce a bounding box on the feature of interest in the Image. 2 classes will be created and tagged to the bounding box (Dynamic obstacle and Static obstacle). It will ensure the training algorithm recognises the features based on the bounding box and its tagged class name. After annotating the images, the dataset has to be put through the training process to establish the custom model. Once the training is done, live detection can be done with a camera and observe the categorized obstacles. While running the live detection with a camera, the output screen window should display a dynamic object bounding box when it observes a human. It should display a static object bounding box if it observes tables and chairs. Figure 9 summarises the process of establishing and testing the custom object detection model.

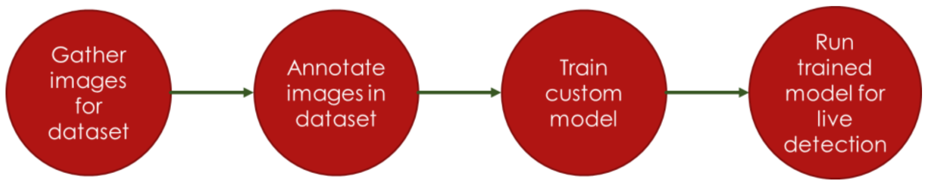

Gather images for dataset
Annotate images in dataset
Train custom model
Run trained model for live detection

Figure 9. Overview of Custom Obstacle Detection establishment

3.2 Integration with ROS

The trained custom model has to integrate with the ROS module. A custom detection script has to be established. When the camera picks up either object, the detected object's coordinates have to be retrieved and published to the navigation module. With the subscribed information of the coordinates, the speed can be adjusted depending on the type of obstacle. Static obstacles remain in a fixed location and are hardly moved around. Dynamic obstacle, on the other hand, has unpredictable movement. The robot will know that the static obstacle is in a fixed location, avoiding the obstacle at a reduced speed. However, when the robot detects a dynamic obstacle, it will not know the movement pattern of the obstacle. Therefore, the robot's speed when avoiding dynamic obstacles will be reduced further when avoiding static obstacles. Figure 10 shows an overall view of the ROS integration with the object detection model. Speed X and Y depicts the different speed reduction the robot will make when detecting either obstacle.

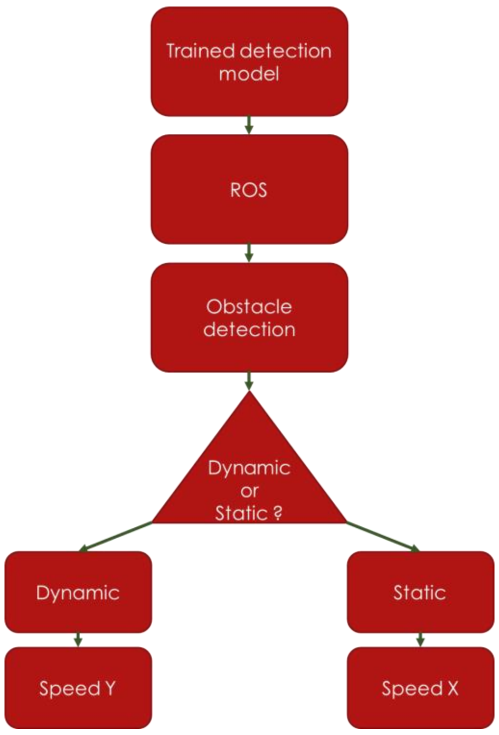


Figure 10. Overview of the integration with robot navigation

4. Implementation

In this section of the report, the implementation process will be discussed.

4.1 Testing on Object Detection

To better understand Deep Learning with PyTorch on the Jetson Nano, image recognition, semantic segmentation, and object detection were tested out [18]. For object detection, a custom dataset of fruit pictures from the Open Images dataset was used. The dataset consists of eight classes and about 6500 images. The dataset was limited to around 1000 images with four classes of Apple, Orange, Banana, and Strawberry to reduce training time. Annotations were already done, and the dataset just had to be put through the training. An example output of the detection can be seen in Figures 11 and 12 for the apple and orange with a confidence level of about 80%. The confidence level relates to the predictability of the detection on the type of class.

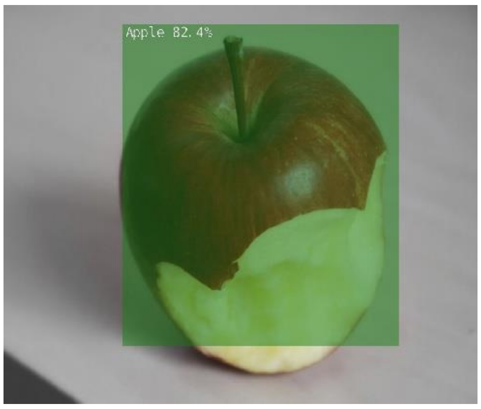


Figure 11. Apple detection 82.4%

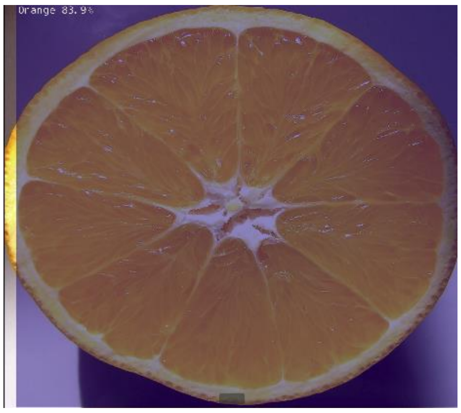


Figure 12. Orange detection 83.9%

4.2 Custom Dataset

A custom dataset had to be formed based on a dining outlet environment. The images collected had to be based on dynamic and static categorization. Images of humans (dynamic obstacle), tables and chairs (static obstacle) were sourced and incorporated into the dataset. Figures 13 and 14 are examples of the images used for dynamic and static category respectively. In order to test out the detection, 100 images were used at first to create the dataset. Not all 100 images were used for the training, as some images were used for the validation process and test.

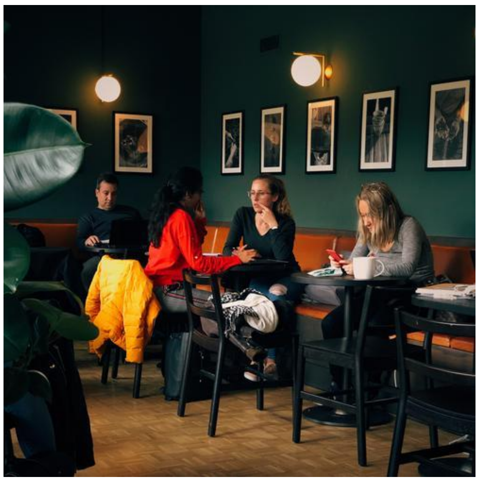


Figure 13. Dynamic image

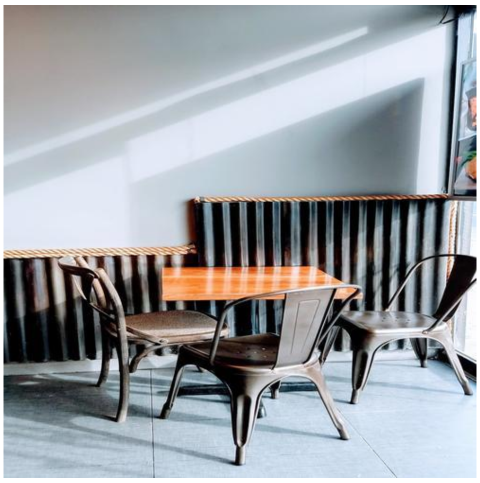


Figure 14. Static image

4.3 Annotation

After creating a custom dataset, the images had to be put through the annotation (labelling) process. In order to conduct labelling, the LabelImg graphical annotation tool was used [19]. Two classes were defined, dynamic object and static object. Bounding boxes were then drawn on every Image to highlight the dynamic and static object. Appendix A.1 and A.2

shows the static and dynamic object labelling examples. The annotated images were saved in the PASCAL VOC format. Appendix A.3 shows the XML file of the annotated dynamic Image. Coordinates of every bounding box were stored in the bounding box property, a child node of the object property parent node. Along with the coordinates, the label name associated with the bounding box was also included in the object property.

4.4 Training

4.4.1 Transfer Learning

The training process of the custom dataset made use of the Transfer Learning technique. Transfer learning is a method in which a model developed for a specific task will be reused as the starting point for a model on a second task, effectively retraining the model to recognise different higher level features [20]. Conducting Transfer Learning will result in reduced training time, and it is preferable for small datasets and consumes less resource during training.

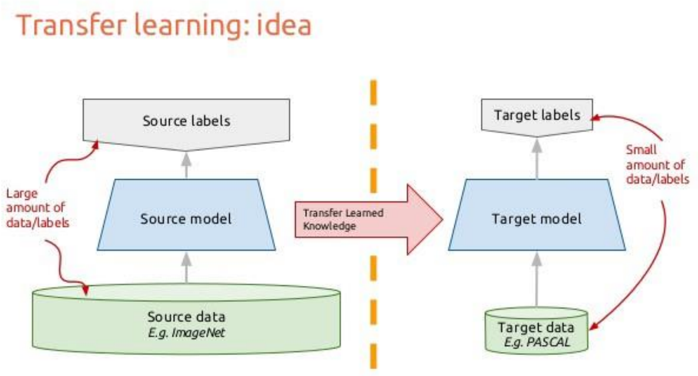


Figure 15. Transfer Learning overview

4.4.2 Training process

The annotated custom dataset had to be put through the Transfer Learning process and retrain the MobileNet-SSD model. After the training process, running the newly trained model will detect dynamic and static obstacles as declared in the annotation process.

Appendix B.1 shows the parameter declaration for the dataset. For the dataset type, VOC was declared since the custom dataset is in the PASCAL VOC format. Along with the dataset type declaration, the path to the dataset location also had to be declared. Appendix B.2 shows the declaration of the pre-trained model. The base-net was declared as a pre-trained model so that the MobileNet-SSD model can be used for the Transfer Learning training. The path to the model location in the file structure was declared to import the model. Training parameters, as seen in Appendix B.3, were also declared. The batch size refers to dividing the dataset by a certain size and training the dataset by batches. Since the dataset is relatively small, the batch size chosen was one. The number of workers is an attribute that informs the data loader on how many sub-processes are to be used for data loading. Memory consumption directly relates to the number of workers used, where a higher number of workers equates to higher memory consumption. Hence, to reduce the memory consumption, the number of workers was set at one. The epoch attribute dictates how many times will the dataset run through the training algorithm [21]. The epoch was set at 30 for the initial training process.

Compute Unified Device Architecture (CUDA) is a special framework that enables parallel computing on the GPU. The training will be conducted using the dedicated Graphics Processor Unit (GPU) instead of the Central Processing Unit (CPU) by enabling CUDA. Running the training process via the GPU will significantly reduce the training time. Therefore, CUDA was enabled during the training process, as seen in Appendix B.4. Appendix B.5 shows the training function where each Image runs through, and Appendix B.6 shows how the training function ran for the number of epochs defined.

4.5 Live detection & ROS integration

To conduct live detection based on the trained custom dataset model, the DetectNet vision primitive was used. The DetectNet primitive is inherited from the TensorNet object, consisting of ImageNet and SegNet primitive used for image classification and segmentation, respectively. The DetectNet script accepts an image as input and outputs a list of coordinates of the detected bounding boxes and their classes and confidence values. Appendix C.1 shows the script used to conduct the live detection, which uses the DetectNet primitive and takes in the image feed from the web camera (/video0) and output the detection result in a display tab.

The object detection feature had to be integrated with the ROS interface. DetectNet was adapted into ROS as a detection node. By doing so, live detection would be achievable via the ROS platform. The raw image feed coming from the camera would be published to the detection node (Appendix C.2). After subscribing to the image feed, the detection node would make use of the model parameters (Appendix C.3) and conduct an image conversion to overlay the images with the detection data. The overlayed image feed, along with information on each image frame, would be published out (Appendix C.4).

In order for the robot movement to understand when to reduce the speed and by how much, the movement program needs information such as the coordinates of the detection and the type of obstacle detected. From the detection node, the coordinates of the bounding box and the detected obstacle's ID have to be published. Appendix C.5 shows each detected object's coordinates and ID, retrieved from the DetectNet class, sending a ROS message to the output to observe the results. Furthermore, the class ID is used to categorize the type of obstacle it detected and send an output message to visualize the speed reduction. Speed X and Y are arbitrary constants to differentiate the reduction of speed into two types where speed X is faster than speed Y. Class ID 1 represents static obstacles. In contrast, class ID 2 represents dynamic obstacles. Therefore, when ID 1 is detected, the speed will be reduced to Speed X, and when ID 2 is detected, the speed will be reduced to Speed Y.

5. Test and improvements

The tests conducted on the object detection framework and the results of the test will be discussed. Furthermore, the improvements implemented to the framework will also be addressed in this section of the report.

5.1 Stationary detection

Once the custom dataset model was trained, it was put to the test using DetectNet to conduct live detection. In the initial test run, both the dynamic and static obstacles were able to be detected. As seen in Figure 16, the camera detected and classified me as a dynamic object with a confidence level of 89.2%. Additionally, in Figure 17, the camera detected the lab table and classified it as a static object with a confidence level of 66.3%. The confidence level returned from the detection for both classes varied due to the different angles of detection. The confidence level returned for the dynamic object class was around 40% for the lowest and 89% for the highest. The confidence level returned for the static object class was approximately 35% for the lowest and 70% for the highest. The frame rate achieved for the detection was around 40 frames per second (FPS), a reasonable frame rate used for live detection.

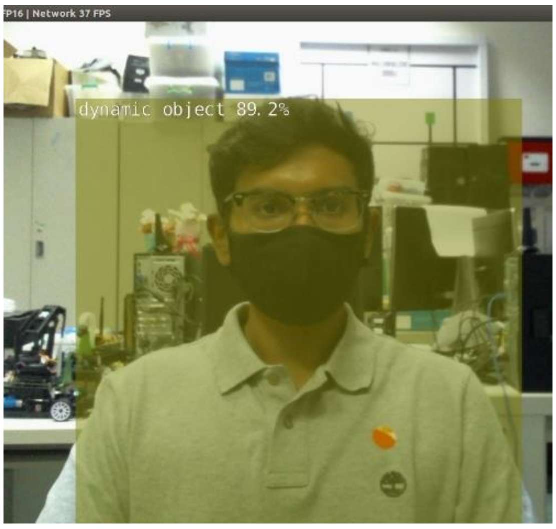


Figure 16. Dynamic object detection

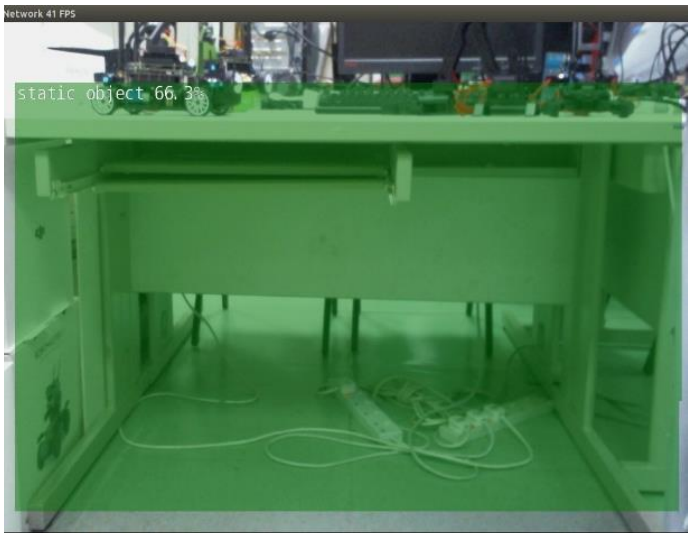


Figure 17. Static object detection

5.2 Movement based detection

Even though the stationary detection was able to detect the obstacles, the implementation of the object detection framework was built with the waiter robot integration in consideration. Therefore, stationary detection was not enough and movement based detection had to be tested too. The Serveur robot was unavailable to test the object detection framework as the new movement program of the robot was still being pursued and was not finished. Hence, another method of testing the object detection framework was required. Therefore, to test the framework on the move, an improvised testing platform was used, requiring manual movement to test the detection capabilities. Figure 18 shows the improvised test platform, which included the traversable table as the robot platform and a long extension cord for the power supply. The Jetson Nano was connected with a small display, web camera, keyboard, and mouse to observe and control the device. The camera was attached at the front of the table to simulate how the camera would be placed on the waiter robot.

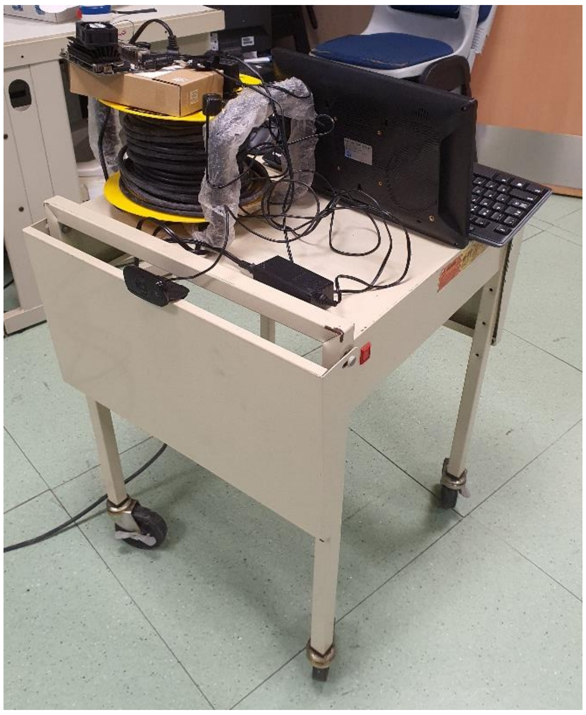


Figure 18. Improvised movement based testing

The test was conducted in a lab that had tables and chairs situated around the room. The room was conducive for the trial, with the tables and chairs to represent the static obstacles. Furthermore, a lab staff was generous to act as the dynamic obstacle during the test process.

Since the movement based testing for the object detection framework was done manually, to ensure the integrity of the test to simulate the robot's movement had to be considered. The initial implemented movement program for the Seveur robot enabled it to traverse at a speed of 0.5 m/s. Therefore, the speed at which the test platform was to traverse had to match between 0.5 m/s. To achieve the speed manually, the distance of the test range was measured, and the value was computed with the robot's speed. Equation 1 shows the time formula used in which the distance and speed values were used. After computing the distance and speed with the formula, the time was calculated. The test platform had to ensure it completes the test range within the stipulated time with the known time. By doing so, the test platform would be able to mimic the travelling speed of the Seveur robot.

$$Time\ (s) = \frac{Distance\ (m)}{Speed\ (m/s)}$$

Equation 1. Distance, Time, Speed formula

While running the detection script, the platform was traversed manually across the room, and the detections were observed. In the initial tests, the static obstacle detection was poor as the detection failed to pick up most of the tables and chairs while on the move. As for the dynamic obstacle, the detection was sporadic. It was having issues detecting when the dynamic obstacle was walking. In Figure 19, right before the platform moved, the detection framework was able to capture the dynamic obstacle when seated, with a confidence level of around 72%. However, it did not detect any static obstacle at that point. Figure 20 showed the output when the platform was on the move. The dynamic obstacle was also on the move, but the detection was sporadic when the obstacle was walking away from the seat. The static obstacle detection was worse, and it was only able to detect the chair that the dynamic obstacle was sitting on when the camera was close to it. The detection could not classify the tables as obstacles, and no bounding boxes formed on the tables.

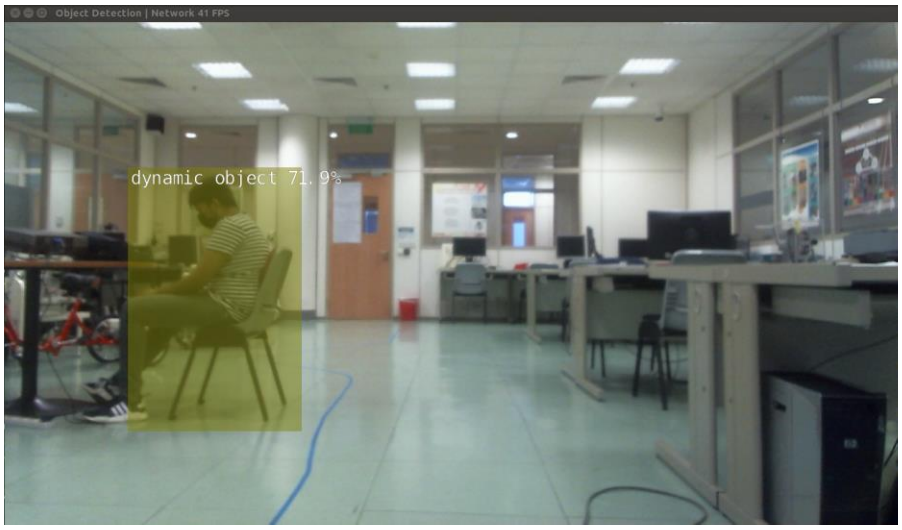


Figure 19. Dynamic obstacle initial detection

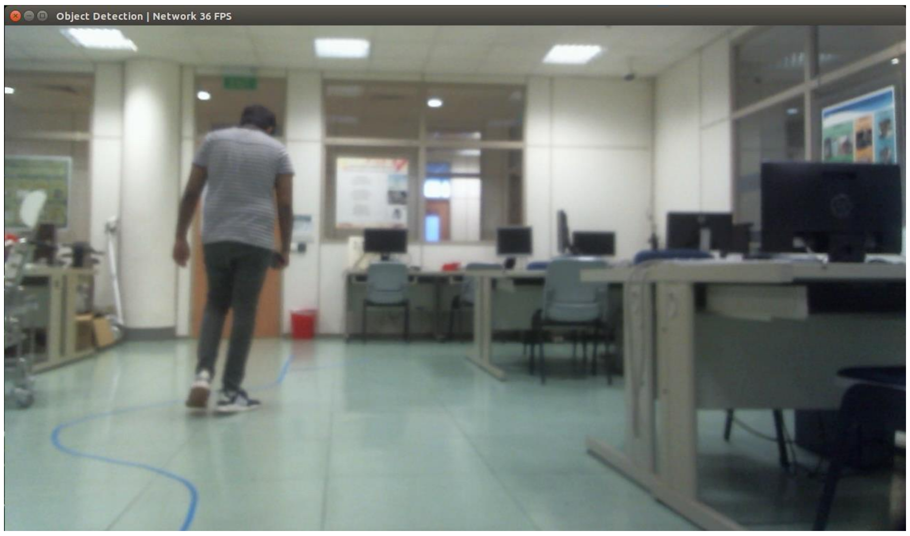


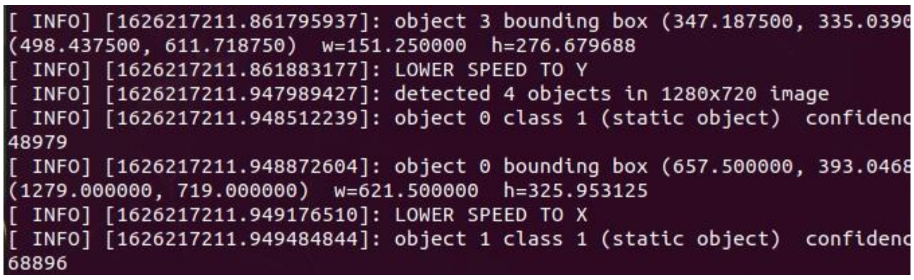
Figure 20. Failed to detect any obstacle

After running the test a few times, the output was similar. The dynamic obstacle was more detectable than the static obstacles. The dynamic obstacle was still detectable when seated and when it is indifferent orientation while seated. As for other positions, such as standing and walking, the detectability was lesser at different orientations. As for the static obstacles, chairs were more detectable than tables. However, the detectability for chairs at different positions was imperfect. Chairs positioned straight at the camera or positioned at an angle towards the camera were captured easier than chairs facing the camera from the back. The frame rate for the movement based detection remained around 40 FPS.

The detection capability was also tested with ROS using the detection node. For the detection node to run, the roscore had to run first for all nodes to communicate. The roscore contains pre-requisites of ROS based systems which are a collection of nodes and programs. Once the roscore was running, the detection node launch file was started. Launch files are XML files that launch multiple ROS nodes at the same time. Additionally, parameters can be set in the launch file, which the respective nodes will adapt. The detection node launch file consisted of the detection node, video source node, and the video output node, as seen in Appendix C.6.

Once the launch file started, the display output appeared, and the detection began. The major difference observed when compared with the tests conducted without ROS integration was the frame rate. In DetecNet, the frame rate ran about 40 FPS, but the frame rate dropped to

8 FPS – 10 FPS in ROS. The frame rate difference was more than half, but the detection was similar to the tests conducted without ROS, and both classes were detectable. Figure 21 shows the output of a static and dynamic obstacle with the detection coordinates and the speed reduction output. Figure 22 shows the graph plot of the different nodes publishing and subscribing. The graph plot was taken from the ROS rqt framework. The framework implements many different Graphical User Interface (GUI) tools in a plugin form, and it can be used for debugging. The plot was taken after the launch file was executed, and so it showed the ongoing publishing and subscribing nodes. The oval shapes in the plot represented the nodes, and there are three ovals since three nodes are communicating with each other. The /video_source/raw and /detectnet/overlay are topics being published by the /video_source and /detecnet nodes respectively.



```
[ INFO] [1626217211.861795937]: object 3 bounding box (347.187500, 335.0390
(498.437500, 611.718750)  w=151.250000  h=276.679688
[ INFO] [1626217211.861883177]: LOWER SPEED TO Y
[ INFO] [1626217211.947989427]: detected 4 objects in 1280x720 image
[ INFO] [1626217211.948512239]: object 0 class 1 (static object)  confidenc
48979
[ INFO] [1626217211.948872604]: object 0 bounding box (657.500000, 393.0468
(1279.000000, 719.000000)  w=621.500000  h=325.953125
[ INFO] [1626217211.949176510]: LOWER SPEED TO X
[ INFO] [1626217211.949484844]: object 1 class 1 (static object)  confidenc
68896
```

Figure 21. Detection output

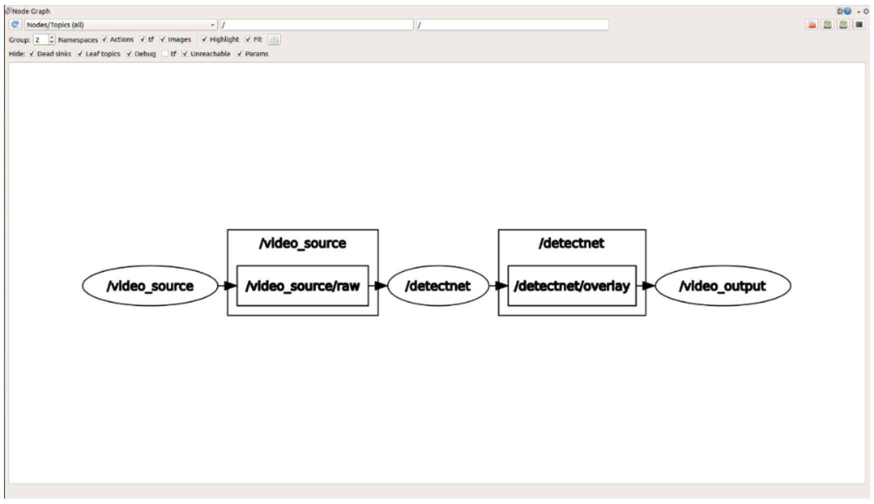


Figure 22. rqt plot of the ROS integration

Even though the object detection framework could detect obstacles, improvements had to be done to enhance detectability and predictability. By conducting the improvements, it would ensure the object detection framework to be more robust and reliable.

5.3 Improvements

In this section, the improvements made to the detection framework along with the results will be discussed.

5.3.1 Increase dataset size & change in annotation

The initial dataset consisted of around 100 images. However, the images were not distributed equally in terms of the classes. As some images had a mixture of annotated classes, the label count would be different between both classes. With an uneven distribution within both classes, the detectability of one class would be better than the other. Therefore, to alleviate this issue and improve the predictability level of both classes, another 100 images were added to the prevailing custom dataset. The images acquired were distributed evenly for both classes.

Additionally, the acquired images consisted of obstacles in a different orientation. For dynamic objects, the images acquired had a mix of seated, standing, and walking human poses, which were in different orientations were chosen. For static objects, different kinds of chairs and tables situated at different orientations were chosen. After acquiring the new images for the dataset, annotations had to be done. In addition to annotating the latest images, the previous annotations were relabeled to ensure a balance between both classes.

After the new custom dataset was annotated, it was put through the training process, and the output was observed. The confidence levels for both classes improved. At different orientations, the confidence level range was higher when compared to the initial tests.

An attempt was made to test out a different method for annotating the custom dataset. The idea was that the detection algorithm would detect an eating environment with people as one class and an empty eating environment as another class. By doing so, the annotation would be more straightforward in which the whole Image could be labelled as a single class (Appendix D.1). In order to test this method, one class was tried out. The images with people in an eating environment were labelled with an additional class. After retraining with the new class, the output was observed. Even though detection was possible, the output only appeared a few times. The probable reason for the imperfect detection could be the custom dataset since the images were varied and the dining area concepts were different.

5.3.2 Higher Epoch and Threshold

In the initial training conducted for the custom dataset, the epoch level was set at 30. Therefore, the training algorithm would run through the dataset 30 times. Since deep learning is an iterative process, running the training algorithm through the dataset several times will provide an optimal result [22]. Figure 23 shows the three different types of learning rate curve fit, resulting in the output achieved after training. No fixed epoch number will provide an optimal result, and it is based on the type of dataset used. Therefore, the custom dataset had to be trained again with different epoch levels to find the optimal fit. As the custom dataset was already trained with an epoch level of 30, the dataset was trained again with the epoch levels of 50 and 80 to observe if the detectability of the trained model improved.

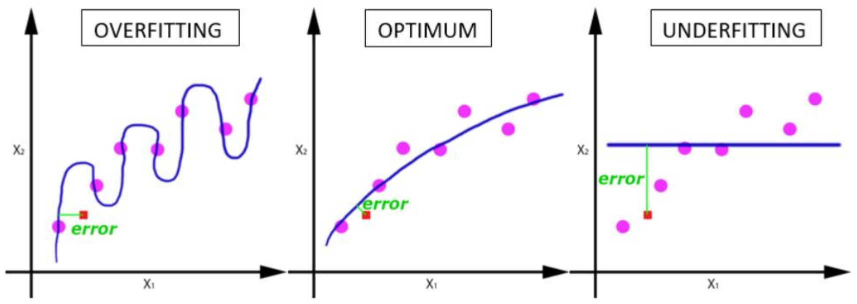


Figure 23. Different learning rate curve fit

After the training was done, the detectability results were compared. At epoch level 50, the detection had improved significantly. Obstacles were more detectable when compared with the training done with 30 epochs. However, the difference was minimal when the detections were compared to both epoch levels 50 and 80. In terms of detectability and confidence levels, both differences were minimal. Therefore, training the custom dataset at an epoch level of 80 was accepted as optimum.

Further training with a higher epoch level could lead to overfitting or overlearning. If the dataset overlearns during training, it will detect obstacles wrongly. Figure 24 shows the detectability of the static obstacles after the training was done at epoch level 80.

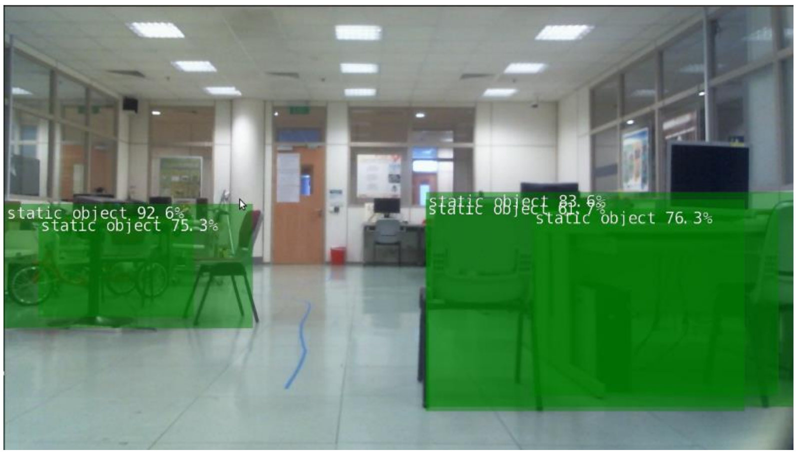


Figure 24. Detection after training with 80 epochs

The increase in epoch levels for the training also led to detection at low light. Previously with the custom dataset trained at epoch level 30, detection at low light was not possible. Figure 25 shows the low light test environment, and detection was not possible. After the increase in epoch level in training, low light detection was possible. However, the detection was sporadic and only was possible a few times.

Furthermore, dynamic obstacles were more challenging to be detected compared to static obstacles. A possible reason could be that the static obstacle's features were more observable than the dynamic obstacle since the features were more detectable. Figures 26 and 27 shows the detection of both obstacle classes in low light.

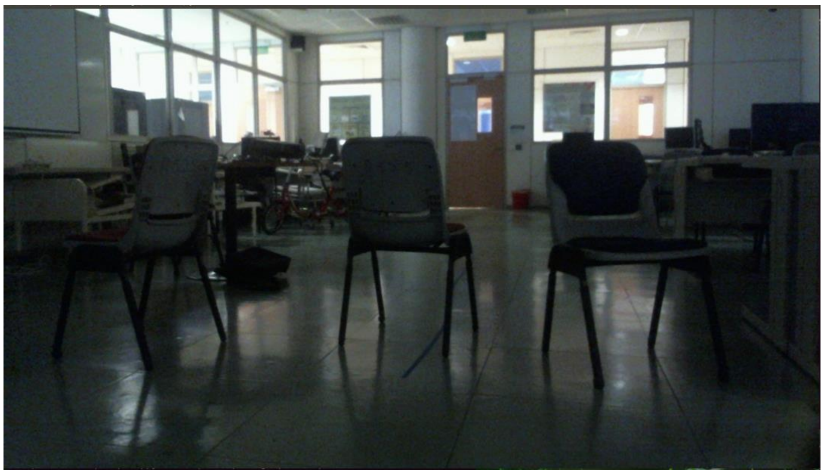


Figure 25. Low light detectability with epoch 30

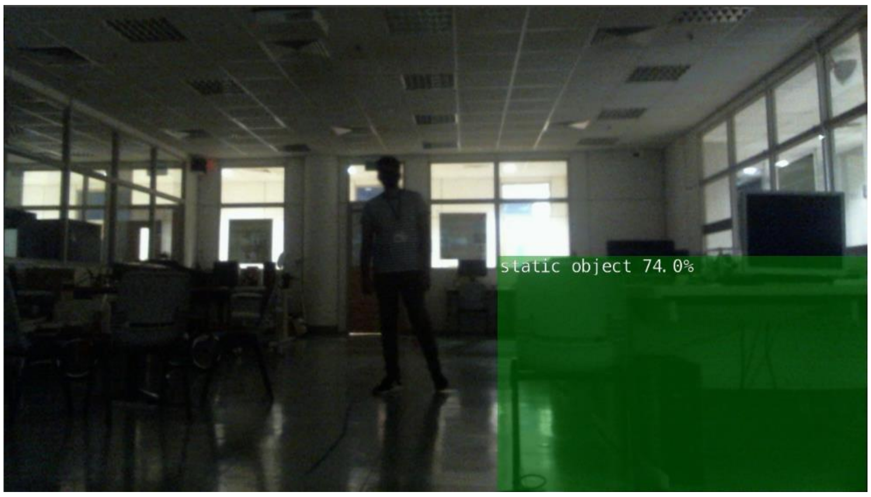


Figure 26. Static obstacle detection at low light

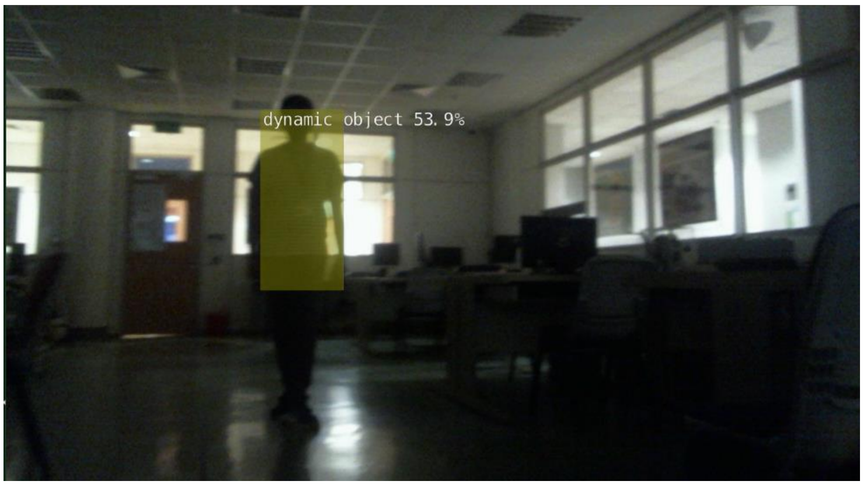


Figure 27. Dynamic obstacle detection at low light

The improvement of the detection with higher epoch also led to more clustered detection. Clustered detection refers to many detections of the same class at one time. By adjusting the threshold value, the clustered detections will vary. Initially, the value was at 0.9, and the number of detections was too many. Figure 28 shows an example of the clustered detection with a threshold value of 0.9. The detection was too many, and the focus on the obstacles nearby were off at the right side. To analyze the best possible threshold, two other values were tested out. A threshold of 0.7 was tested out to observe the difference. The clustered detection had reduced when compared with the previous threshold value of 0.9. Although there were improvements, the clustered detection could be improved. Hence, another threshold value of 0.55 was used. This time the clustered detection improved significantly, whereby it could easily detect the nearer obstacles and the output could be seen. This value provided a focused detection and ensured the closer obstacles were captured.

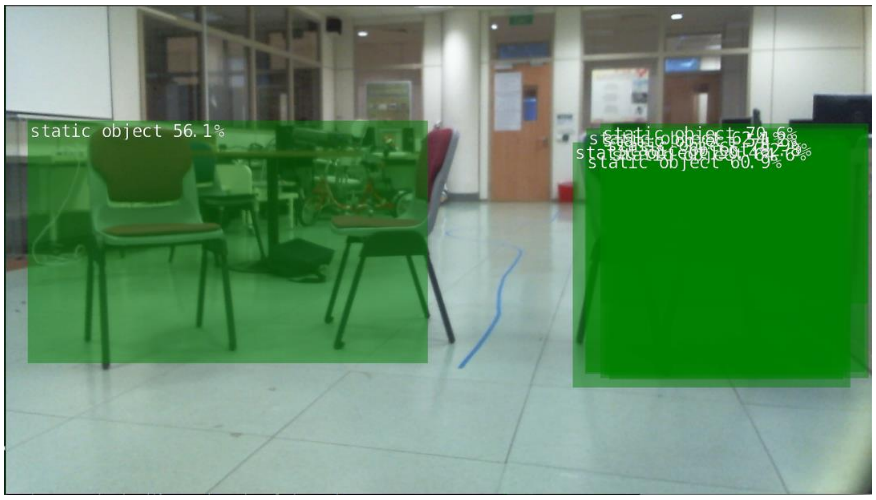


Figure 28. Threshold 0.9

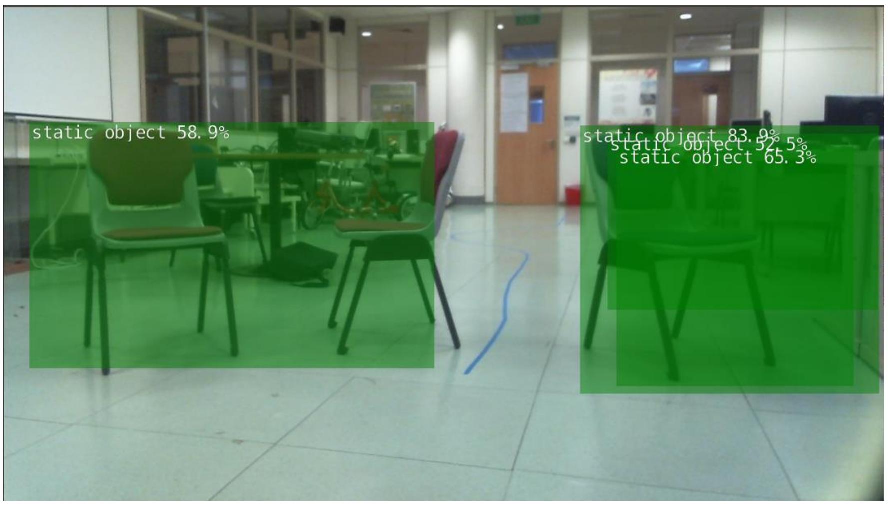


Figure 29. Threshold 0.7

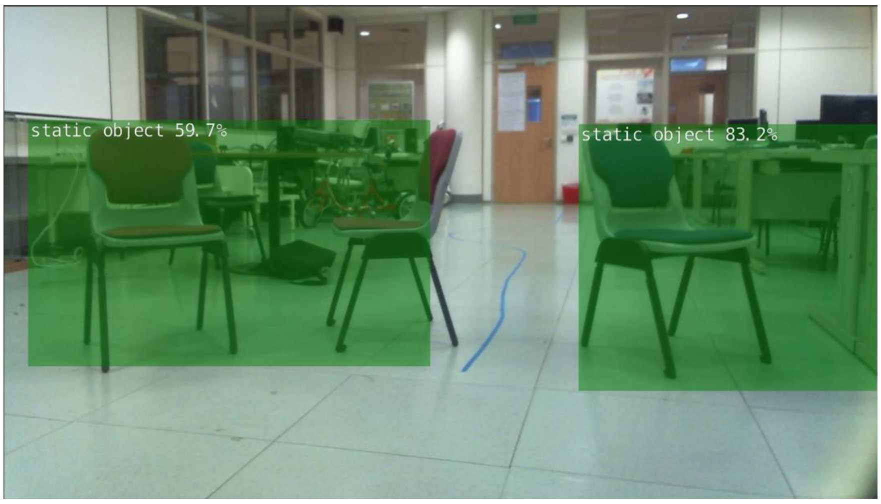


Figure 30. Threshold 0.55

5.3.3 Data Augmentation

Data augmentation is when new data is synthesized from an already available set of data, manipulating the current dataset and providing the impression of a larger dataset [23]. Some augmentations done were photometric distortion and mirroring.

Photometric distortion is manipulating the pixel values of the Image. Appendix D.2 shows the photometric distortion class, and Appendices D.3 to D.5 shows how the contrast, saturation, hue, colour, lighting, and brightness were manipulated. Unlike photometric distortion, mirroring and expanding augmentations manipulated the image dimensions rather than pixels. Appendix D.6 shows the expand and random mirror augmentations, where the dataset images were expanded and mirrored along with the bounding boxes. After the different augmentations were done, the images, class labels, and bounding boxes were returned and fed to the training process.

5.3.4 Summary

Overall, the combined improvement methods resulted in the object detection model being significantly better when compared to the previous tests. The integration of the improved object detection framework with ROS also provided better detection capability, and obstacles were detectable.

6. Discussion and Conclusion

In this section of the report, the discussion on the errors made, challenges, limitations, and future work will be discussed along with the conclusion.

6.1 Errors made

6.1.1 PASCAL VOC formatting

The annotated dataset was saved in the PASCAL VOC format. When conducting the training after the annotation process, an error resulted while running the training script. The error was the training algorithm unable to find the images in the dataset and the annotation files pertaining to the images. Therefore, the VOC dataset script was analysed to see why the dataset and annotation were not retrievable. The VOC data script will gather the images, annotation, and the image set ID (the images name) from specified folders (Annotation, ImageSets, JPEGImages, SegmentationClass, and SegmentationObject) declared in the root.

Appendix E.1 shows the declaration of the path library with root. The ID of each Image was retrieved from the ImageSets files, and the files were retrieved based on the root (Appendix E.2). The ID was then used to retrieve the annotations and images (Appendices E.3 and E.4). After restructuring the file format, as seen in Figure 31, the error was rectified. Under the Annotations sub-directory, the image labelling files (XML format) were saved there. As for the ImageSets sub-directory, the Image set IDs are sorted accordingly into training and validation. The JPEGImages sub-directory holds all the dataset images.

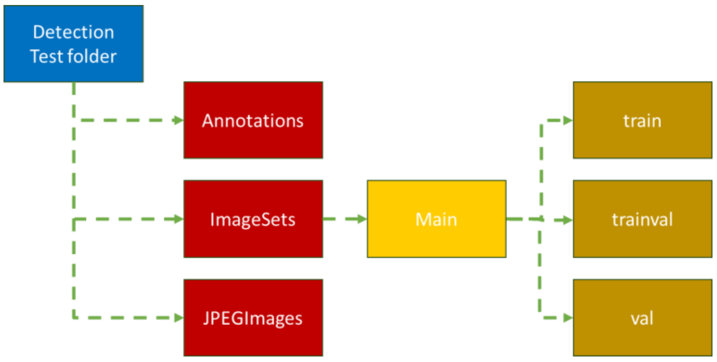


Figure 31. PASCAL VOC folder structure

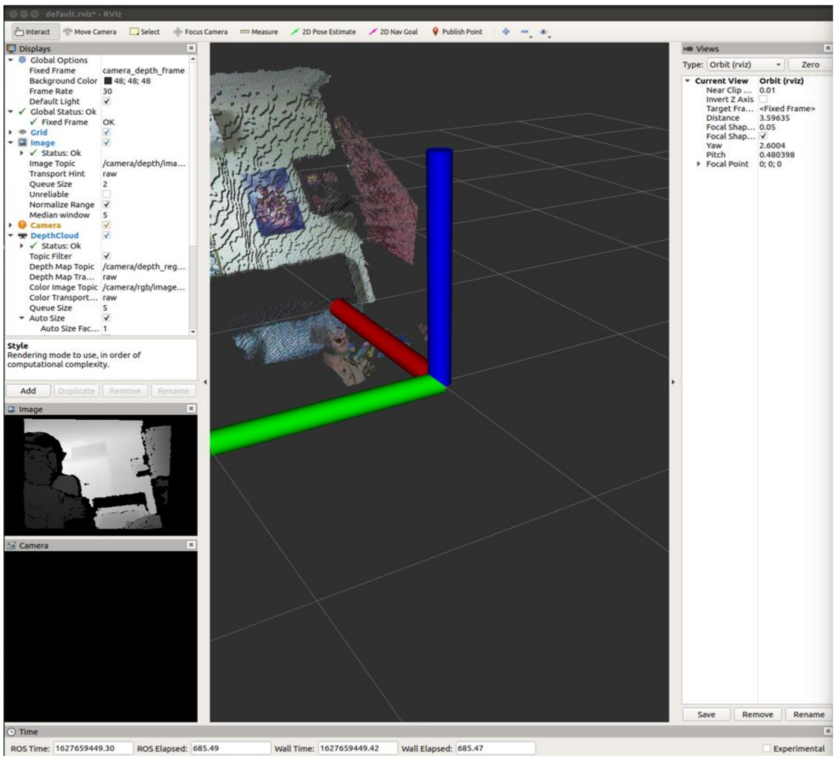
6.1.2 Image resizing

After collecting images for the custom dataset was done, labelling was done immediately without resizing all the images collected. According to Figure 7 above, when the dataset is put through training with the MobileNet-SSD model, the images will be downsized to 300x300 before being fed into the MobileNet network as the Deep Neural Network (DNN) runs at a lower resolution. However, after the annotation of the custom dataset was done, conducting the training yielded errors. One error displayed a low memory warning, which hinted that the images used were too big for the Jetson Nano as loading large images consumes a lot of memory. The images collected were around the size of 5033x3355. A resizing script had to be established (Appendix E.5) to resize all the images in the folder. The images were resized to a dimension of 500x500 to ensure lower memory consumption when loading. After resizing, the images were annotated again, and the newly annotated dataset was able to train successfully.

6.2 Challenges and Limitations

Conducting Deep Learning techniques and working on ROS was new for me. With no prior experience in these technologies, I had a steep learning curve to understand and utilize the frameworks to the best of my capabilities. Modules like Object-Oriented Programming (OOP) and Systems and Software Engineering (SSE) taught in my course of study helped in the understanding and was used throughout the project phase. As Deep Learning utilizes both Python language and C++ language, I had to learn the Python language and refresh my C++ knowledge taught in OOP. Furthermore, with the given timeframe of the project, I had to implement and debug any errors faced throughout the project timeframe. The black-box testing method, taught in SSE, was used to test the object detection framework throughout the project phase to observe the output, ensure it works, and the quality of the output.

Due to the lack of knowledge on ROS and resources, certain improvement attempts made were not possible. In order to obtain the distance of the obstacles detected from the object detection framework, a depth camera was tried to replace the web camera. The depth camera used was the Astra depth camera from Orbbec. The camera's depth range is 8 meters, and the image resolution is 640x480 at 30 FPS. The camera was operable in the ROS environment by running the camera package. Figure 32 shows the visualization of the depth camera output in rviz, a 3D visualization tool for ROS. By running the depth topic in the camera's package, the 3D environment scan was able to visualize my surrounding area, and I was able to observe myself. However, the depth camera was not able to be integrated with the object detection framework. The detection node failed to gather image input from the depth camera, and no output image was shown. The integration of the depth camera with the object detection framework proved to be a challenge, but if it could be implemented, it would enhance the detection capabilities. Figure 33 shows a potential implementation of the depth camera without the direct integration with the object detection framework. The distance of the obstacle can be obtained from the depth camera, and it can be passed to the movement program along with the obstacle coordinates retrieved from the object detection framework.



Global Options	
Fixed Frame	camera_depth_frame
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
Image	✓
✓ Status: Ok	
Image Topic	/camera/depth/ima...
Transport Hint	raw
Queue Size	2
Unreliable	☐
Normalize Range	✓
Median window	5
Camera	✓
DepthCloud	✓
✓ Status: Ok	
Topic Filter	✓
Depth Map Topic	/camera/depth_reg...
Depth Map Tra...	raw
Color Image Topic	/camera/rgb/image...
Color Transport...	raw
Queue Size	5
Auto Size	✓
Auto Size Fac...	1

Style
Rendering mode to use, in order of computational complexity.

Type: Orbit (rviz)

Current View	**Orbit (rviz)**
Near Clip ...	0.01
Invert Z Axis	☐
Target Fra...	<Fixed Frame>
Distance	3.59635
Focal Shap...	0.05
Focal Shap...	✓
Yaw	2.6004
Pitch	0.480398
Focal Point	0; 0; 0

ROS Time: 1627659449.30 ROS Elapsed: 685.49 Wall Time: 1627659449.42 Wall Elapsed: 685.47

Figure 32. Running depth camera in rviz

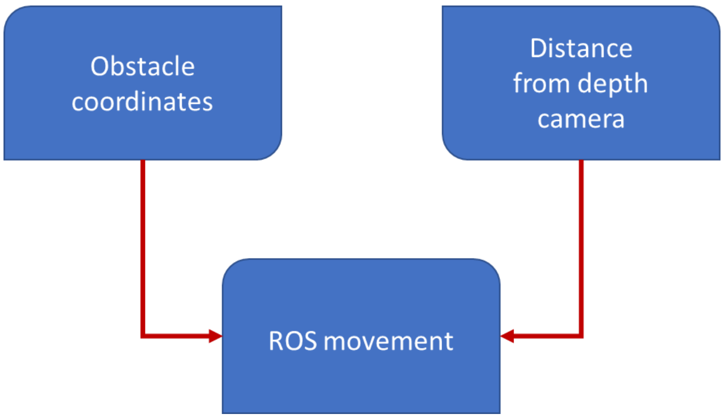


Figure 33. integration of depth distance with object detection framework and movement program

With the unavailability of the Seveur robot to test the object detection framework with the movement program, the manual movement method was used. There may be a possibility that the movement will not travel at a constant speed since it is done manually. There may be a short burst of acceleration or deceleration while traversing. Therefore, the object detection output may not wholly reflect the robot movement. However, to ensure this limitation would be minimized, further tests were conducted at a lower and higher speed compared to the robot's initial speed declared at 0.5m/s. At those speeds, obstacles were still detectable and proved the conducted tests would nearly reflect how the waiter robot detected the obstacles.

6.3 Future improvements

Integrating AI into the Seveur robot would allow for more possibilities for the waiter robot to be enhanced. The object detection framework could be improved by introducing more varied data to the custom dataset to enhance detectability further. The object detection framework could detect more diverse classes apart from dynamic and static obstacles, which could detect different dynamic obstacles such as adults and children. With the detection of children, the robot's movement could be even slower due to the rash conduct of children compared to adults. Apart from the obstacle avoidance aspect while serving, the varied detection could also integrate into an ordering system. The waiter robot could detect either a child or adult and output different menus to cater to them. Additionally, payment could be made via contactless or QR code payment services. It could eliminate the need for human interaction from ordering to the service of the food.

6.4 Conclusion

In conclusion, the implementation of AI with regards to obstacle avoidance was to provide the Seveur waiter robot with a means to traverse its intended environment dynamically. By doing so, the robot would be more intelligent and open more opportunities for AI implementations in the robot.

From the experimental results, the current outcome of the project was able to prove the concept of using AI for obstacle avoidance. The design of the object detection framework would allow dynamic obstacle detection capabilities by categorizing the type of obstacle observed. Furthermore, with the integration of the object detection framework with the ROS middleware, the necessary output data was able to be retrieved for use in the movement program. With the data, the movement program would manipulate the travelling speed while avoiding the obstacle.

Reference

1. Ravensbergen, R. (2021, March 19). What problems will autonomous robots really solve? TechTalks. https://bdtechtalks.com/2021/03/19/autonomous-robots-applications/.
2. Service Robotics Market Size, Report: Industry Forecast [2020-2027]. Service Robotics Market Size, Report | Industry Forecast [2020-2027]. (2019, October). https://www.fortunebusinessinsights.com/industry-reports/service-robotics-market101805.
3. Meisenzahl, M. (2020, June 04). A reopened Dutch restaurant is using robots to implement social distancing by serving and seating customers - see how it works. Retrieved May 2, 2021, from https://www.businessinsider.com/robot-serves-food-takes-temperatures-covid-19-in-the-netherlands-2020-6
4. Nussey, S. (2020, September 28). SoftBank brings food service robot to labour-strapped Japan. Retrieved May 2, 2021, from https://www.reuters.com/article/softbank-group-robotics-idUSKBN26J0GC
5. Wan AYS, Foo E, Lai ZY, Chen HH, Lau WSM (2019) Waiter Bots for Casual Restaurants. Int J Robot Eng 4:018
6. Singapore restaurant 'hires' robot waiters. AsiaOne. (2016, February 1). https://www.asiaone.com/singapore/singapore-restaurant-hires-robot-waiters.
7. Lyons, K. (2020, May 31). A restaurant in the Netherlands is using creepy robot waiters for social distancing. The Verge. https://www.theverge.com/2020/5/31/21276318/restaurant-netherlands-robot-waiters-social-distancing-pandemic.
8. Assuma, M. (2019, June 6). Nvidia Jetson Nano: What is it, and what can it do? https://www.newegg.com/insider/nvidia-jetson-nano-dev-kit-makers-overview-what-can-it-do/.
9. JetPack. NVIDIA Developer Documentation. (n.d.). https://docs.nvidia.com/jetson/jetpack/introduction/index.html#:~:text=NVIDIA%20JetPack%20SDK%20is%20the,%2C%20developer%20tools%2C%20and%20documentation.
10. Hargrave, M. (2021, April 6). How Deep Learning Can Help Prevent Financial Fraud. Investopedia. https://www.investopedia.com/terms/d/deep-learning.asp#:~:text=Deep%20learning%20is%20a%20subset,learning%20or%20dee

p%20neural%20network.

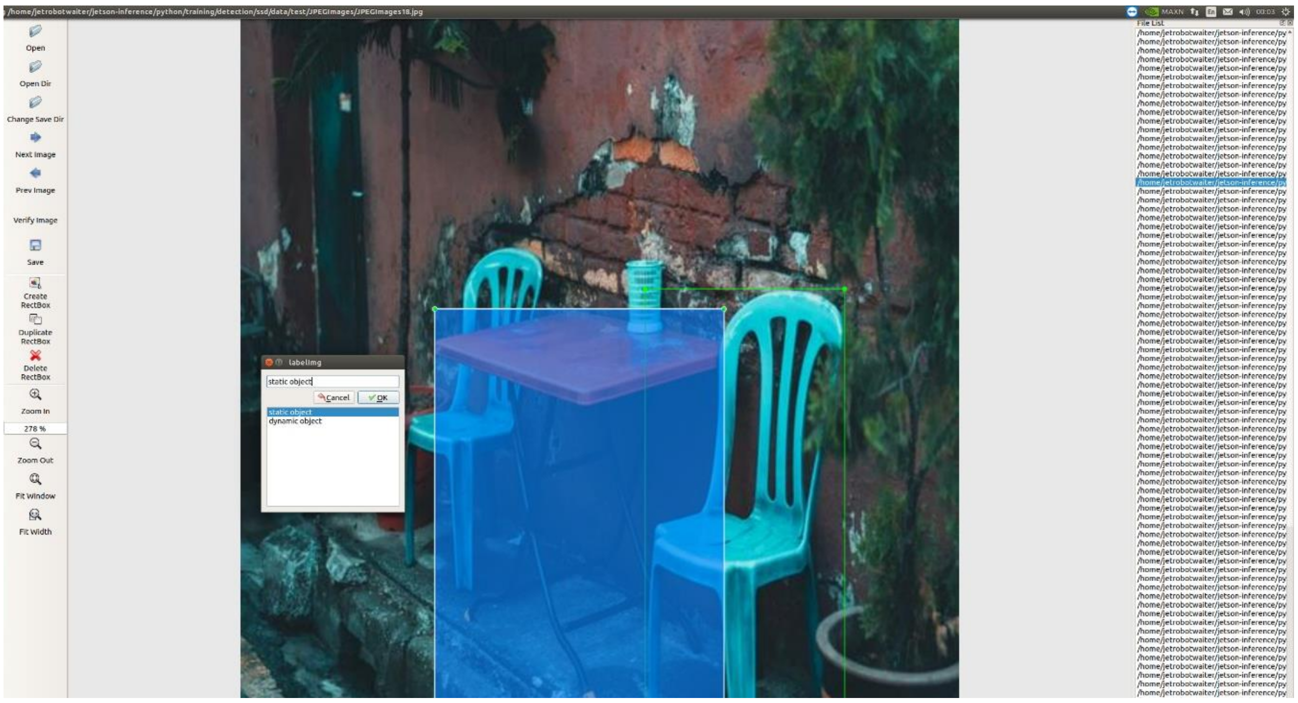
11. Yegulalp, S. (2019, June 18). What is TensorFlow? The machine learning library explained. InfoWorld. https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html.
12. Mwiti, D. (2021, April 9). Deep Learning with PyTorch: An Introduction. Medium. https://heartbeat.fritz.ai/introduction-to-pytorch-for-deep-learning-5b437cea90ac#:~:text=PyTorch%20uses%20a%20technique%20called,near%20zero%20but%20not%20zero.
13. India, U. (2018, November 14). Tensorflow or PyTorch: The force is strong with which one? Medium. https://medium.com/@UdacityINDIA/tensorflow-or-pytorch-the-force-is-strong-with-which-one-68226bb7dab4#:~:text=Tensorflow%20is%20based%20on%20Theano,frameworks%20define%20the%20computational%20graphs.&amp;text=But%20in%20PyTorch%2C%20you%20can,graph%20on%2Dthe%2Dgo.
14. Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. CoRR, abs/1704.04861, 2017. URL: http://arxiv.org/abs/1704.04861, arXiv:1704.04861.
15. Pokhrel, S. (2020, August 26). Real-Time Vehicle Detection with MobileNet SSD and Xailient. Xailient. https://www.xailient.com/post/real-time-vehicle-detection-with-mobilenet-ssd-and-xailient#:~:text=MobileNet%20is%20a%20light%2Dweight,mobiles%20and%20embedded%20vision%20applications.&text=Single%20Shot%20object%20detection%20or,multiple%20objects%20within%20the%20image.
16. Wiki. ros.org. (n.d.). http://wiki.ros.org/ROS/Introduction.
17. Anwar, A. (2021, February 15). Part 3: Create Your First ROS Publisher and Subscriber Nodes. Medium. https://medium.com/swlh/part-3-create-your-first-ros-publisher-and-subscriber-nodes-2e833dea7598.
18. Franklin, D. (2021, February 26). dusty-nv/jetson-inference. GitHub. https://github.com/dusty-nv/jetson-inference/blob/master/README.md#hello-ai-world.
19. Tzutalin. LabelImg. Git code (2015). https://github.com/tzutalin/labelImg
20. Le, J. (2020, May 10). The 10 deep learning methods ai practitioners need to apply.

Medium. https://data-notes.co/the-10-deep-learning-methods-ai-practitioners-need-to-apply-885259f402c1.

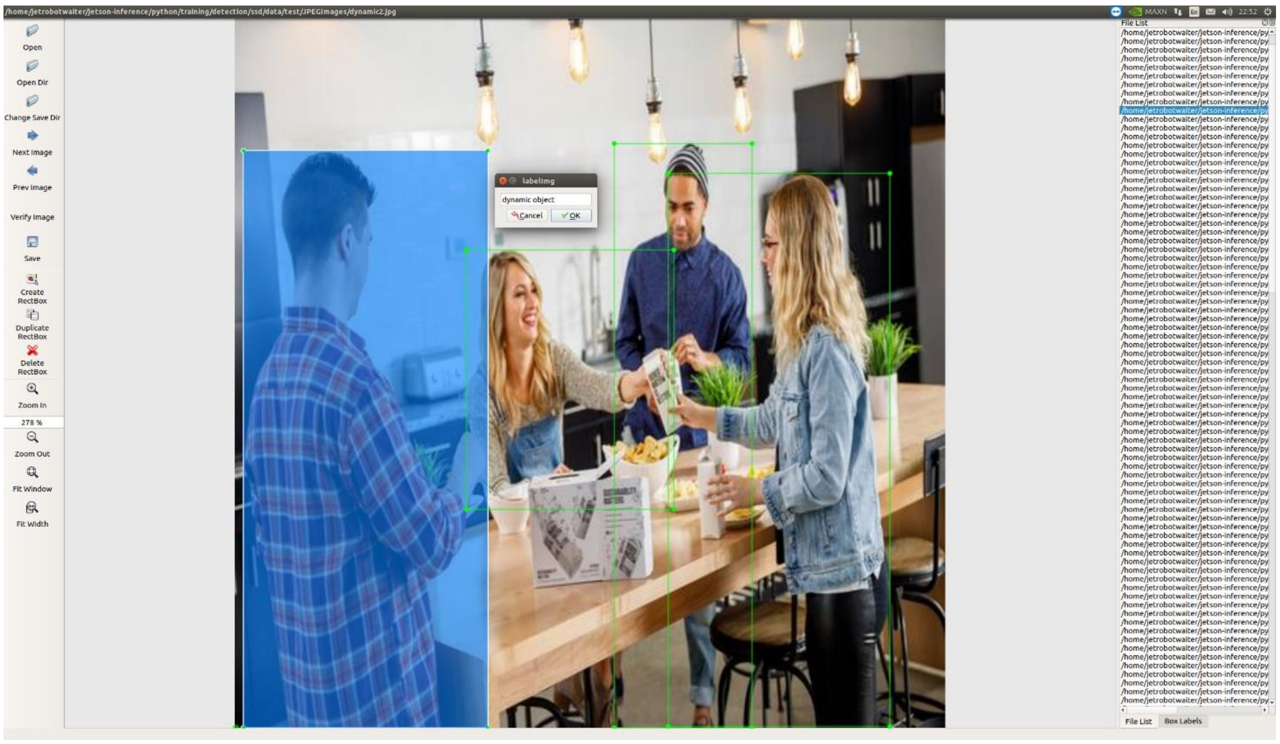
21. Brownlee, J. (2019, October 25). Difference between a batch and an epoch in a neural network. Machine Learning Mastery. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/.
22. Sharma, S. (2019, March 5). Epoch vs batch size vs iterations. Medium. https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9.
23. Gandhi, A. (2021, May 20). Data Augmentation: How to use deep learning when you have limited data. AI &amp; Machine Learning Blog. https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/.

Appendix

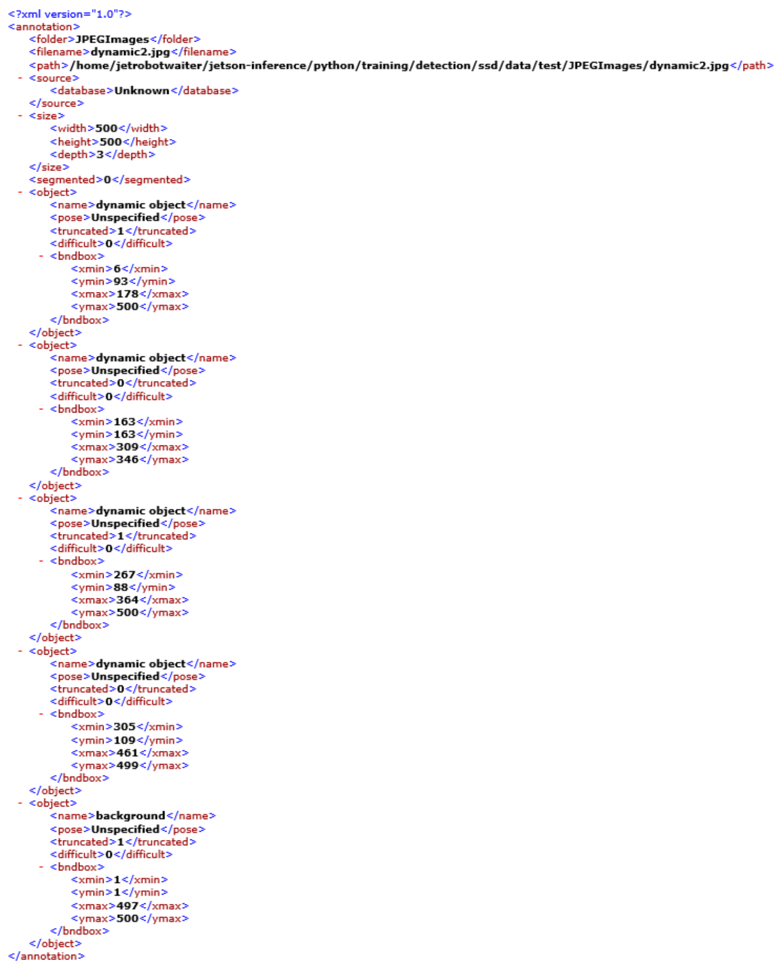
Appendix A LabelImg graphical annotation tool



Appendix A.1 static object labelling



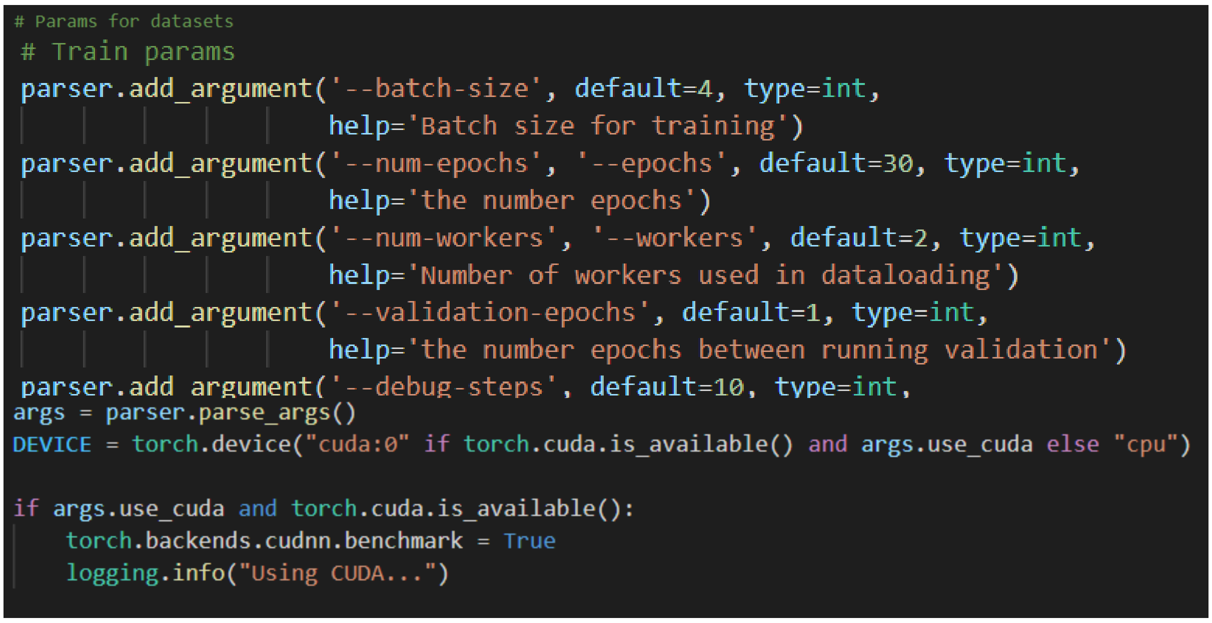
Appendix A.2 dynamic object labelling



```xml
<?xml version="1.0"?>
<annotation>
   <folder>JPEGImages</folder>
   <filename>dynamic2.jpg</filename>
   <path>/home/jetrobotwaiter/jetson-inference/python/training/detection/ssd/data/test/JPEGImages/dynamic2.jpg</path>
 - <source>
      <database>Unknown</database>
   </source>
 - <size>
      <width>500</width>
      <height>500</height>
      <depth>3</depth>
   </size>
   <segmented>0</segmented>
 - <object>
      <name>dynamic object</name>
      <pose>Unspecified</pose>
      <truncated>1</truncated>
      <difficult>0</difficult>
    - <bndbox>
         <xmin>6</xmin>
         <ymin>93</ymin>
         <xmax>178</xmax>
         <ymax>500</ymax>
      </bndbox>
   </object>
 - <object>
      <name>dynamic object</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
    - <bndbox>
         <xmin>163</xmin>
         <ymin>163</ymin>
         <xmax>309</xmax>
         <ymax>346</ymax>
      </bndbox>
   </object>
 - <object>
      <name>dynamic object</name>
      <pose>Unspecified</pose>
      <truncated>1</truncated>
      <difficult>0</difficult>
    - <bndbox>
         <xmin>267</xmin>
         <ymin>88</ymin>
         <xmax>364</xmax>
         <ymax>500</ymax>
      </bndbox>
   </object>
 - <object>
      <name>dynamic object</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
    - <bndbox>
         <xmin>305</xmin>
         <ymin>109</ymin>
         <xmax>461</xmax>
         <ymax>499</ymax>
      </bndbox>
   </object>
 - <object>
      <name>background</name>
      <pose>Unspecified</pose>
      <truncated>1</truncated>
      <difficult>0</difficult>
    - <bndbox>
         <xmin>1</xmin>
         <ymin>1</ymin>
         <xmax>497</xmax>
         <ymax>500</ymax>
      </bndbox>
   </object>
</annotation>
```

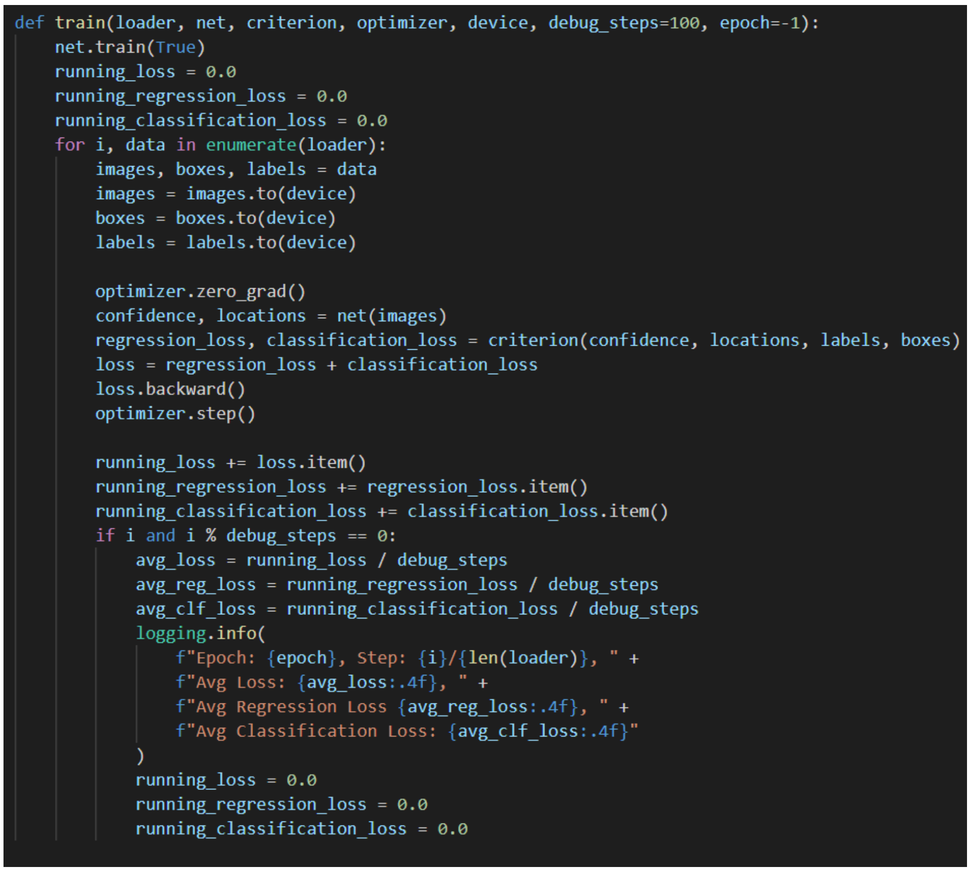
Appendix A.3 XML file of annotated dynamic image

Appendix B Training script



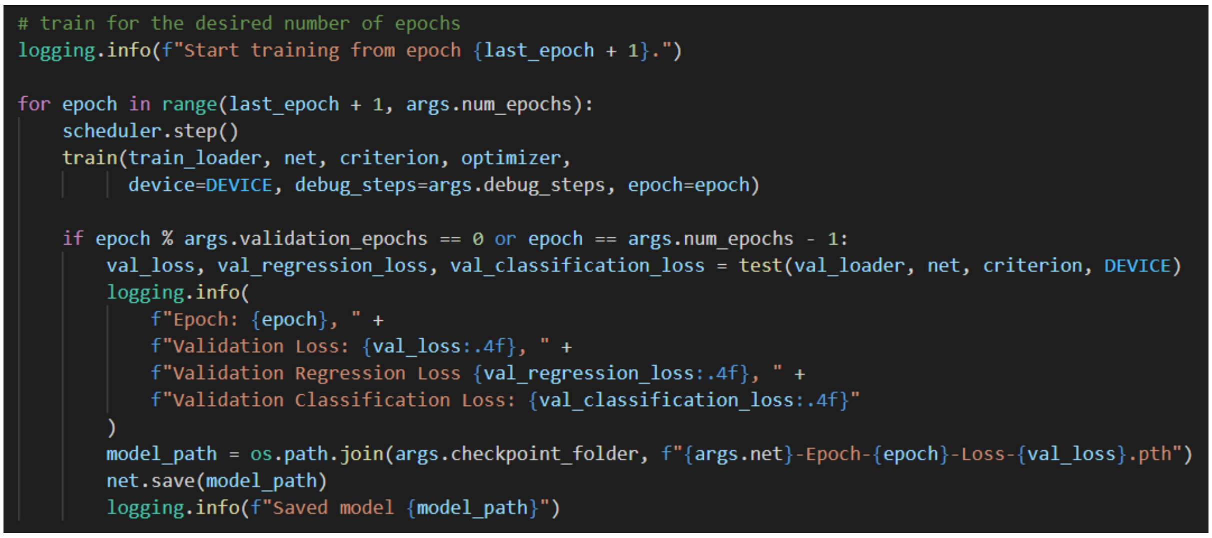
```python
# Params for datasets
# Train params
parser.add_argument('--batch-size', default=4, type=int,
                    help='Batch size for training')
parser.add_argument('--num-epochs', '--epochs', default=30, type=int,
                    help='the number epochs')
parser.add_argument('--num-workers', '--workers', default=2, type=int,
                    help='Number of workers used in dataloading')
parser.add_argument('--validation-epochs', default=1, type=int,
                    help='the number epochs between running validation')
parser.add_argument('--debug-steps', default=10, type=int,
args = parser.parse_args()
DEVICE = torch.device("cuda:0" if torch.cuda.is_available() and args.use_cuda else "cpu")

if args.use_cuda and torch.cuda.is_available():
    torch.backends.cudnn.benchmark = True
    logging.info("Using CUDA...")
```

Appendix B.4 enabling CUDA



```python
def train(loader, net, criterion, optimizer, device, debug_steps=100, epoch=-1):
    net.train(True)
    running_loss = 0.0
    running_regression_loss = 0.0
    running_classification_loss = 0.0
    for i, data in enumerate(loader):
        images, boxes, labels = data
        images = images.to(device)
        boxes = boxes.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
        confidence, locations = net(images)
        regression_loss, classification_loss = criterion(confidence, locations, labels, boxes)
        loss = regression_loss + classification_loss
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        running_regression_loss += regression_loss.item()
        running_classification_loss += classification_loss.item()
        if i and i % debug_steps == 0:
            avg_loss = running_loss / debug_steps
            avg_reg_loss = running_regression_loss / debug_steps
            avg_clf_loss = running_classification_loss / debug_steps
            logging.info(
                f"Epoch: {epoch}, Step: {i}/{len(loader)}, " +
                f"Avg Loss: {avg_loss:.4f}, " +
                f"Avg Regression Loss {avg_reg_loss:.4f}, " +
                f"Avg Classification Loss: {avg_clf_loss:.4f}"
            )
            running_loss = 0.0
            running_regression_loss = 0.0
            running_classification_loss = 0.0
```

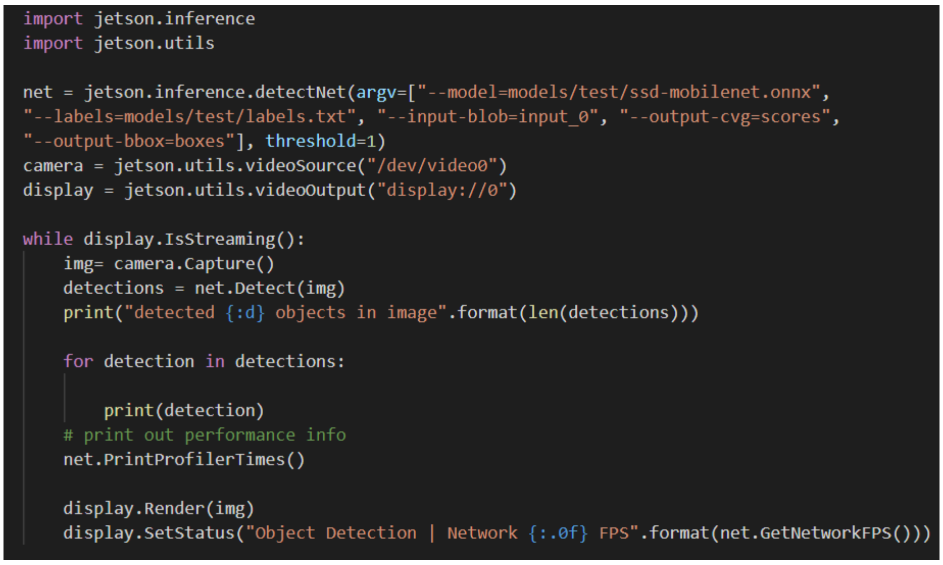
Appendix B.5 training function



```python
# train for the desired number of epochs
logging.info(f"Start training from epoch {last_epoch + 1}.")

for epoch in range(last_epoch + 1, args.num_epochs):
    scheduler.step()
    train(train_loader, net, criterion, optimizer,
          device=DEVICE, debug_steps=args.debug_steps, epoch=epoch)

    if epoch % args.validation_epochs == 0 or epoch == args.num_epochs - 1:
        val_loss, val_regression_loss, val_classification_loss = test(val_loader, net, criterion, DEVICE)
        logging.info(
            f"Epoch: {epoch}, " +
            f"Validation Loss: {val_loss:.4f}, " +
            f"Validation Regression Loss {val_regression_loss:.4f}, " +
            f"Validation Classification Loss: {val_classification_loss:.4f}"
        )
        model_path = os.path.join(args.checkpoint_folder, f"{args.net}-Epoch-{epoch}-Loss-{val_loss}.pth")
        net.save(model_path)
        logging.info(f"Saved model {model_path}")
```

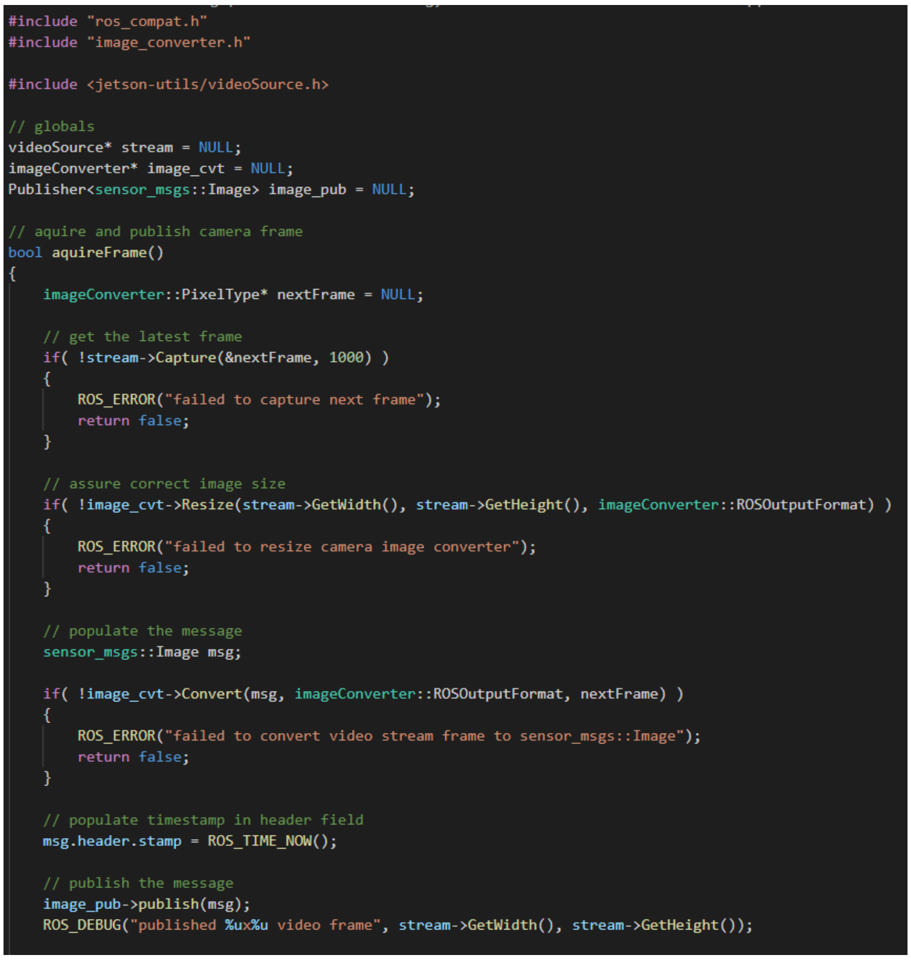
Appendix B.6 training for number of epochs

Appendix C DetectNet and ROS



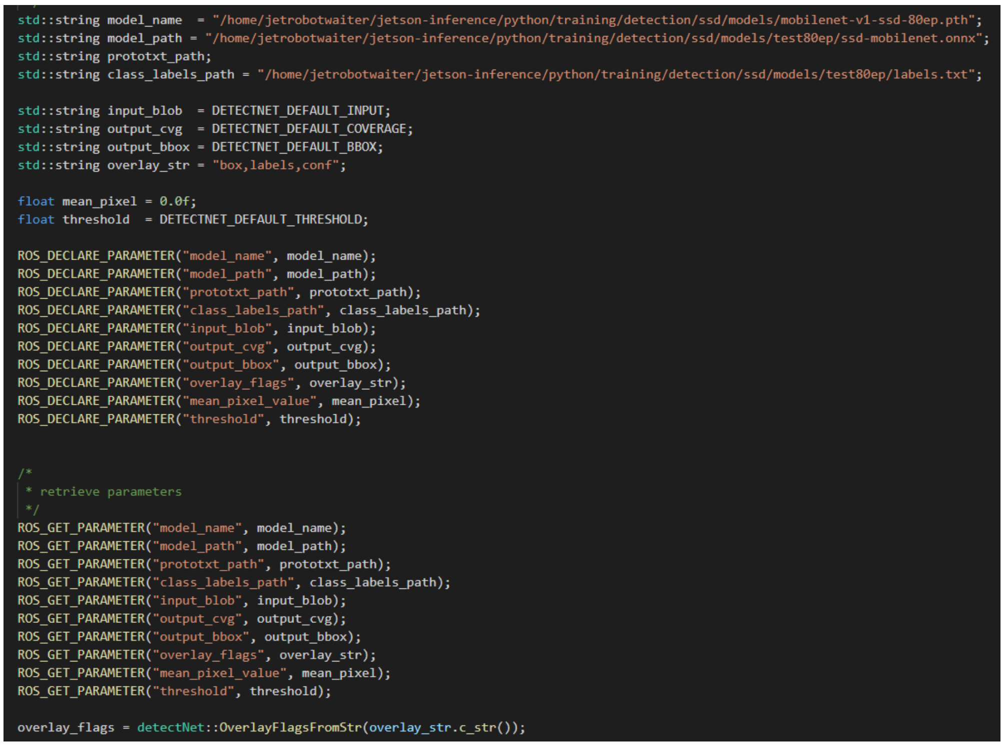
```python
import jetson.inference
import jetson.utils

net = jetson.inference.detectNet(argv=["--model=models/test/ssd-mobilenet.onnx",
"--labels=models/test/labels.txt", "--input-blob=input_0", "--output-cvg=scores",
"--output-bbox=boxes"], threshold=1)
camera = jetson.utils.videoSource("/dev/video0")
display = jetson.utils.videoOutput("display://0")

while display.IsStreaming():
    img= camera.Capture()
    detections = net.Detect(img)
    print("detected {:d} objects in image".format(len(detections)))

    for detection in detections:

        print(detection)
    # print out performance info
    net.PrintProfilerTimes()

    display.Render(img)
    display.SetStatus("Object Detection | Network {:.0f} FPS".format(net.GetNetworkFPS()))
```

Appendix C.1 detectNet detection script



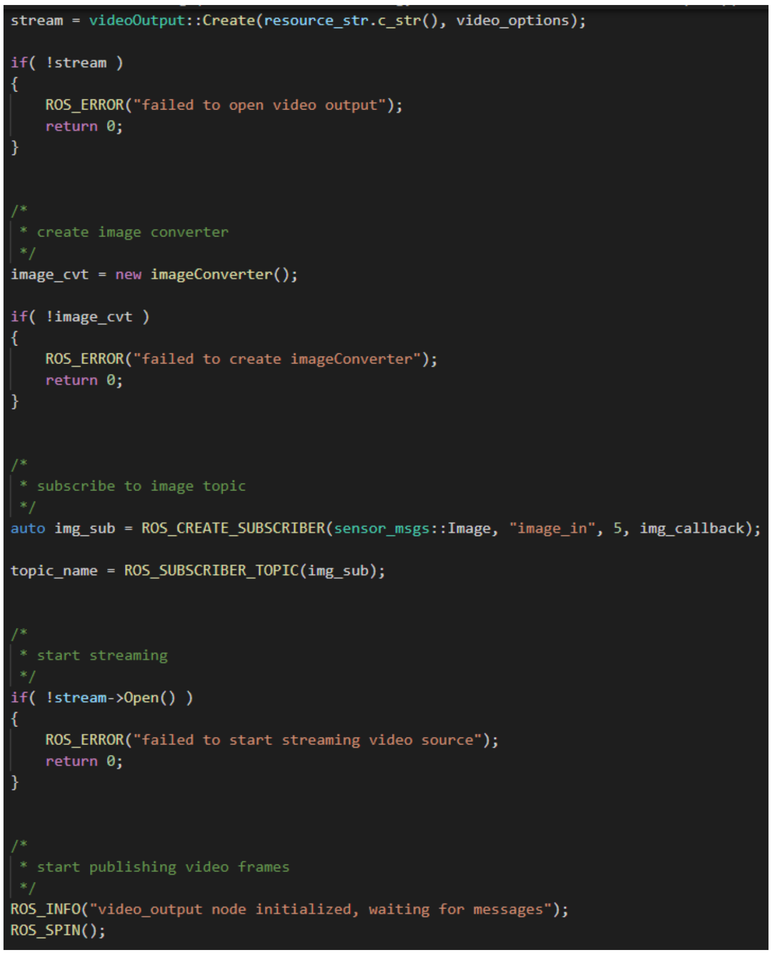
```cpp
#include "ros_compat.h"
#include "image_converter.h"

#include <jetson-utils/videoSource.h>

// globals
videoSource* stream = NULL;
imageConverter* image_cvt = NULL;
Publisher<sensor_msgs::Image> image_pub = NULL;

// aquire and publish camera frame
bool aquireFrame()
{
    imageConverter::PixelType* nextFrame = NULL;

    // get the latest frame
    if( !stream->Capture(&nextFrame, 1000) )
    {
        ROS_ERROR("failed to capture next frame");
        return false;
    }

    // assure correct image size
    if( !image_cvt->Resize(stream->GetWidth(), stream->GetHeight(), imageConverter::ROSOutputFormat) )
    {
        ROS_ERROR("failed to resize camera image converter");
        return false;
    }

    // populate the message
    sensor_msgs::Image msg;

    if( !image_cvt->Convert(msg, imageConverter::ROSOutputFormat, nextFrame) )
    {
        ROS_ERROR("failed to convert video stream frame to sensor_msgs::Image");
        return false;
    }

    // populate timestamp in header field
    msg.header.stamp = ROS_TIME_NOW();

    // publish the message
    image_pub->publish(msg);
    ROS_DEBUG("published %ux%u video frame", stream->GetWidth(), stream->GetHeight());
```

Appendix C.2 ROS video source node acquire image feed



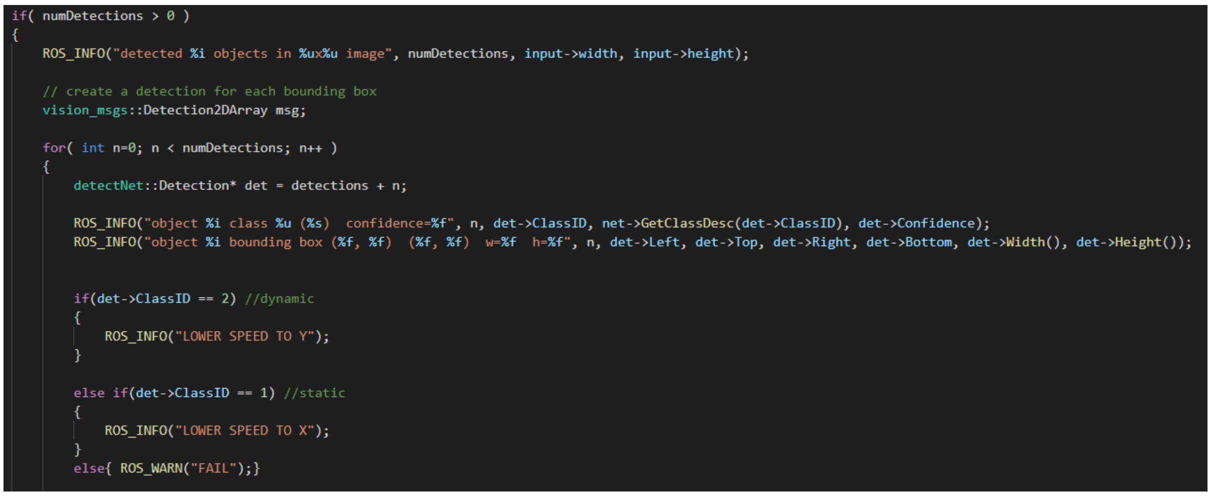
```cpp
std::string model_name  = "/home/jetrobotwaiter/jetson-inference/python/training/detection/ssd/models/mobilenet-v1-ssd-80ep.pth";
std::string model_path = "/home/jetrobotwaiter/jetson-inference/python/training/detection/ssd/models/test80ep/ssd-mobilenet.onnx";
std::string prototxt_path;
std::string class_labels_path = "/home/jetrobotwaiter/jetson-inference/python/training/detection/ssd/models/test80ep/labels.txt";

std::string input_blob  = DETECTNET_DEFAULT_INPUT;
std::string output_cvg  = DETECTNET_DEFAULT_COVERAGE;
std::string output_bbox = DETECTNET_DEFAULT_BBOX;
std::string overlay_str = "box,labels,conf";

float mean_pixel = 0.0f;
float threshold  = DETECTNET_DEFAULT_THRESHOLD;

ROS_DECLARE_PARAMETER("model_name", model_name);
ROS_DECLARE_PARAMETER("model_path", model_path);
ROS_DECLARE_PARAMETER("prototxt_path", prototxt_path);
ROS_DECLARE_PARAMETER("class_labels_path", class_labels_path);
ROS_DECLARE_PARAMETER("input_blob", input_blob);
ROS_DECLARE_PARAMETER("output_cvg", output_cvg);
ROS_DECLARE_PARAMETER("output_bbox", output_bbox);
ROS_DECLARE_PARAMETER("overlay_flags", overlay_str);
ROS_DECLARE_PARAMETER("mean_pixel_value", mean_pixel);
ROS_DECLARE_PARAMETER("threshold", threshold);


/*
 * retrieve parameters
 */
ROS_GET_PARAMETER("model_name", model_name);
ROS_GET_PARAMETER("model_path", model_path);
ROS_GET_PARAMETER("prototxt_path", prototxt_path);
ROS_GET_PARAMETER("class_labels_path", class_labels_path);
ROS_GET_PARAMETER("input_blob", input_blob);
ROS_GET_PARAMETER("output_cvg", output_cvg);
ROS_GET_PARAMETER("output_bbox", output_bbox);
ROS_GET_PARAMETER("overlay_flags", overlay_str);
ROS_GET_PARAMETER("mean_pixel_value", mean_pixel);
ROS_GET_PARAMETER("threshold", threshold);

overlay_flags = detectNet::OverlayFlagsFromStr(overlay_str.c_str());
```

Appendix C.3 ROS acquiring model parameters



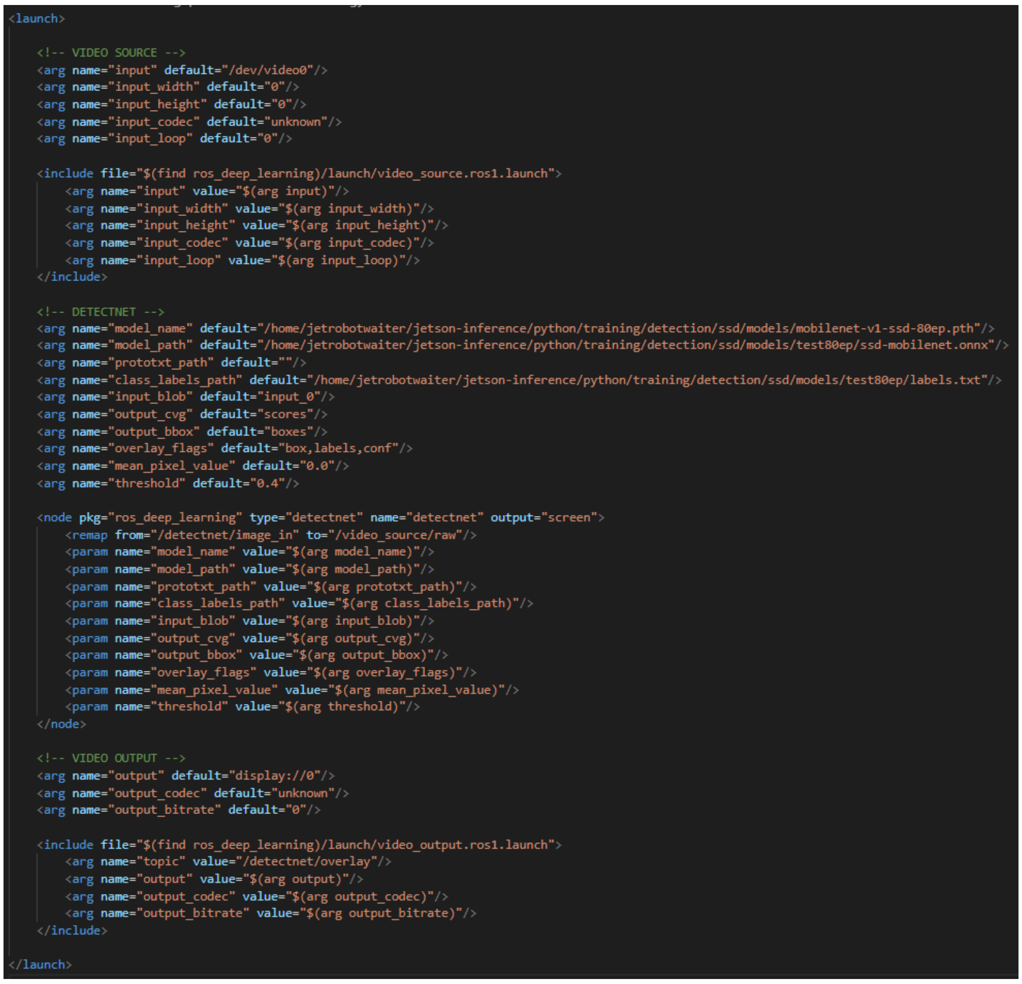
```cpp
stream = videoOutput::Create(resource_str.c_str(), video_options);

if( !stream )
{
    ROS_ERROR("failed to open video output");
    return 0;
}


/*
 * create image converter
 */
image_cvt = new imageConverter();

if( !image_cvt )
{
    ROS_ERROR("failed to create imageConverter");
    return 0;
}


/*
 * subscribe to image topic
 */
auto img_sub = ROS_CREATE_SUBSCRIBER(sensor_msgs::Image, "image_in", 5, img_callback);

topic_name = ROS_SUBSCRIBER_TOPIC(img_sub);


/*
 * start streaming
 */
if( !stream->Open() )
{
    ROS_ERROR("failed to start streaming video source");
    return 0;
}


/*
 * start publishing video frames
 */
ROS_INFO("video_output node initialized, waiting for messages");
ROS_SPIN();
```

Appendix C.4 ROS video output node publishing output



```
if( numDetections > 0 )
{
    ROS_INFO("detected %i objects in %ux%u image", numDetections, input->width, input->height);

    // create a detection for each bounding box
    vision_msgs::Detection2DArray msg;

    for( int n=0; n < numDetections; n++ )
    {
        detectNet::Detection* det = detections + n;

        ROS_INFO("object %i class %u (%s)  confidence=%f", n, det->ClassID, net->GetClassDesc(det->ClassID), det->Confidence);
        ROS_INFO("object %i bounding box (%f, %f)  (%f, %f)  w=%f  h=%f", n, det->Left, det->Top, det->Right, det->Bottom, det->Width(), det->Height());


        if(det->ClassID == 2) //dynamic
        {
            ROS_INFO("LOWER SPEED TO Y");
        }

        else if(det->ClassID == 1) //static
        {
            ROS_INFO("LOWER SPEED TO X");
        }
        else{ ROS_WARN("FAIL");}
```

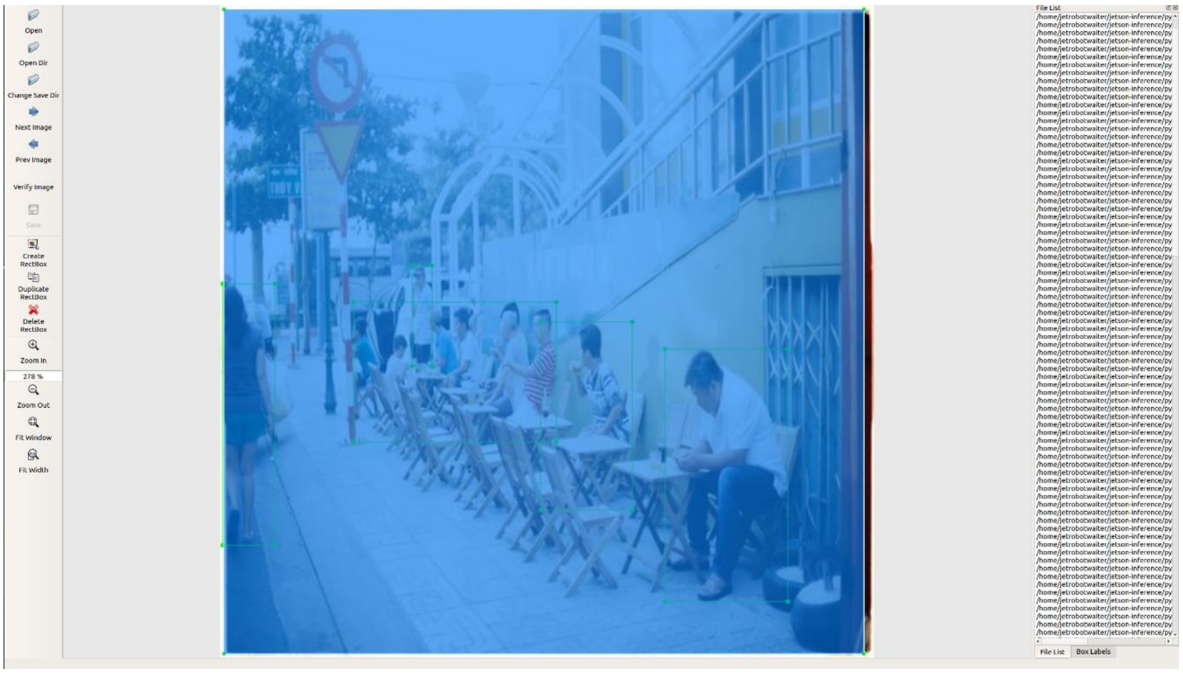
Appendix C.5 Detected obstacle coordinates and ID



```
<launch>

    <!-- VIDEO SOURCE -->
    <arg name="input" default="/dev/video0"/>
    <arg name="input_width" default="0"/>
    <arg name="input_height" default="0"/>
    <arg name="input_codec" default="unknown"/>
    <arg name="input_loop" default="0"/>

    <include file="$(find ros_deep_learning)/launch/video_source.ros1.launch">
        <arg name="input" value="$(arg input)"/>
        <arg name="input_width" value="$(arg input_width)"/>
        <arg name="input_height" value="$(arg input_height)"/>
        <arg name="input_codec" value="$(arg input_codec)"/>
        <arg name="input_loop" value="$(arg input_loop)"/>
    </include>

    <!-- DETECTNET -->
    <arg name="model_name" default="/home/jetrobotwaiter/jetson-inference/python/training/detection/ssd/models/mobilenet-v1-ssd-80ep.pth"/>
    <arg name="model_path" default="/home/jetrobotwaiter/jetson-inference/python/training/detection/ssd/models/test80ep/ssd-mobilenet.onnx"/>
    <arg name="prototxt_path" default=""/>
    <arg name="class_labels_path" default="/home/jetrobotwaiter/jetson-inference/python/training/detection/ssd/models/test80ep/labels.txt"/>
    <arg name="input_blob" default="input_0"/>
    <arg name="output_cvg" default="scores"/>
    <arg name="output_bbox" default="boxes"/>
    <arg name="overlay_flags" default="box,labels,conf"/>
    <arg name="mean_pixel_value" default="0.0"/>
    <arg name="threshold" default="0.4"/>

    <node pkg="ros_deep_learning" type="detectnet" name="detectnet" output="screen">
        <remap from="/detectnet/image_in" to="/video_source/raw"/>
        <param name="model_name" value="$(arg model_name)"/>
        <param name="model_path" value="$(arg model_path)"/>
        <param name="prototxt_path" value="$(arg prototxt_path)"/>
        <param name="class_labels_path" value="$(arg class_labels_path)"/>
        <param name="input_blob" value="$(arg input_blob)"/>
        <param name="output_cvg" value="$(arg output_cvg)"/>
        <param name="output_bbox" value="$(arg output_bbox)"/>
        <param name="overlay_flags" value="$(arg overlay_flags)"/>
        <param name="mean_pixel_value" value="$(arg mean_pixel_value)"/>
        <param name="threshold" value="$(arg threshold)"/>
    </node>

    <!-- VIDEO OUTPUT -->
    <arg name="output" default="display://0"/>
    <arg name="output_codec" default="unknown"/>
    <arg name="output_bitrate" default="0"/>

    <include file="$(find ros_deep_learning)/launch/video_output.ros1.launch">
        <arg name="topic" value="/detectnet/overlay"/>
        <arg name="output" value="$(arg output)"/>
        <arg name="output_codec" value="$(arg output_codec)"/>
        <arg name="output_bitrate" value="$(arg output_bitrate)"/>
    </include>

</launch>
```

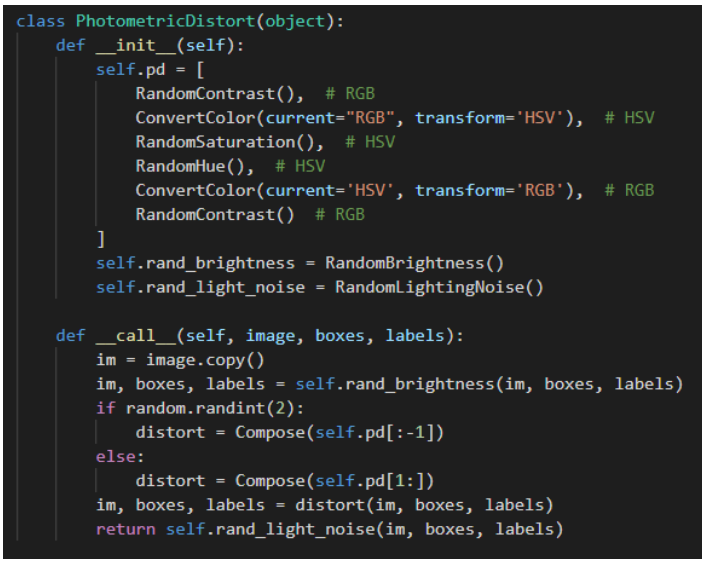
Appendix C.6 detection node launch file in XML

Appendix D Improvements



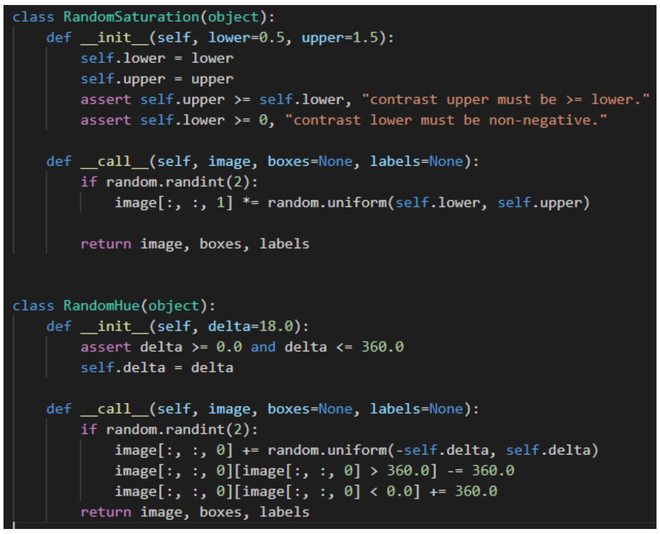
```
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
/home/jetrobotwaiter/jetson-inference/py
```

Appendix D.1 New class labelling



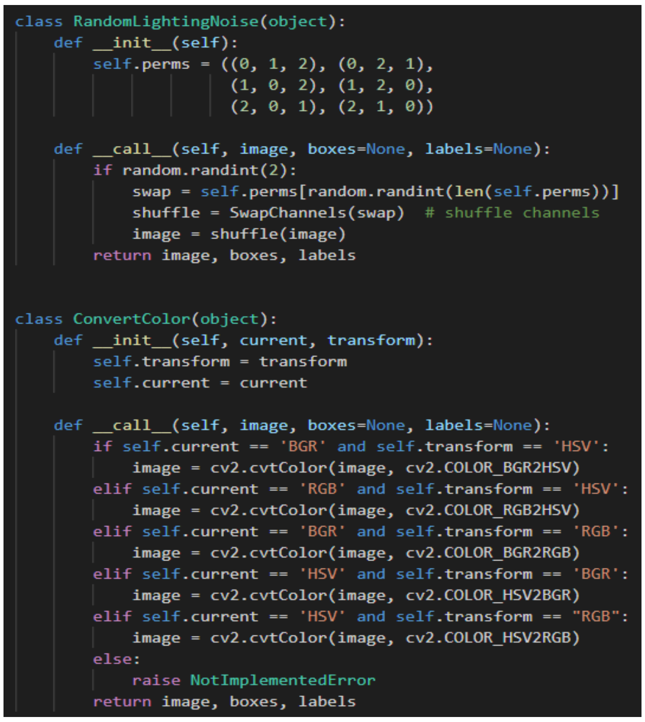
```python
class PhotometricDistort(object):
    def __init__(self):
        self.pd = [
            RandomContrast(),  # RGB
            ConvertColor(current="RGB", transform='HSV'),  # HSV
            RandomSaturation(),  # HSV
            RandomHue(),  # HSV
            ConvertColor(current='HSV', transform='RGB'),  # RGB
            RandomContrast()  # RGB
        ]
        self.rand_brightness = RandomBrightness()
        self.rand_light_noise = RandomLightingNoise()

    def __call__(self, image, boxes, labels):
        im = image.copy()
        im, boxes, labels = self.rand_brightness(im, boxes, labels)
        if random.randint(2):
            distort = Compose(self.pd[:-1])
        else:
            distort = Compose(self.pd[1:])
        im, boxes, labels = distort(im, boxes, labels)
        return self.rand_light_noise(im, boxes, labels)
```

Appendix D.2 Photometric distortion



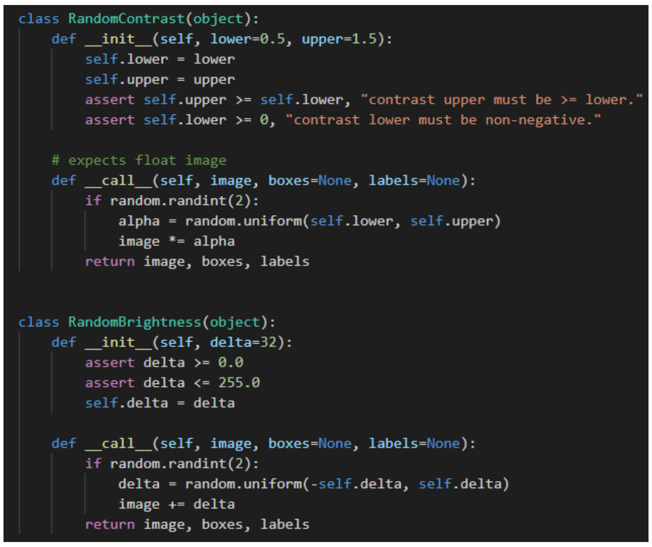
```
class RandomSaturation(object):
    def __init__(self, lower=0.5, upper=1.5):
        self.lower = lower
        self.upper = upper
        assert self.upper >= self.lower, "contrast upper must be >= lower."
        assert self.lower >= 0, "contrast lower must be non-negative."

    def __call__(self, image, boxes=None, labels=None):
        if random.randint(2):
            image[:, :, 1] *= random.uniform(self.lower, self.upper)

        return image, boxes, labels


class RandomHue(object):
    def __init__(self, delta=18.0):
        assert delta >= 0.0 and delta <= 360.0
        self.delta = delta

    def __call__(self, image, boxes=None, labels=None):
        if random.randint(2):
            image[:, :, 0] += random.uniform(-self.delta, self.delta)
            image[:, :, 0][image[:, :, 0] > 360.0] -= 360.0
            image[:, :, 0][image[:, :, 0] < 0.0] += 360.0
        return image, boxes, labels
```

Appendix D.3 random saturation and hue



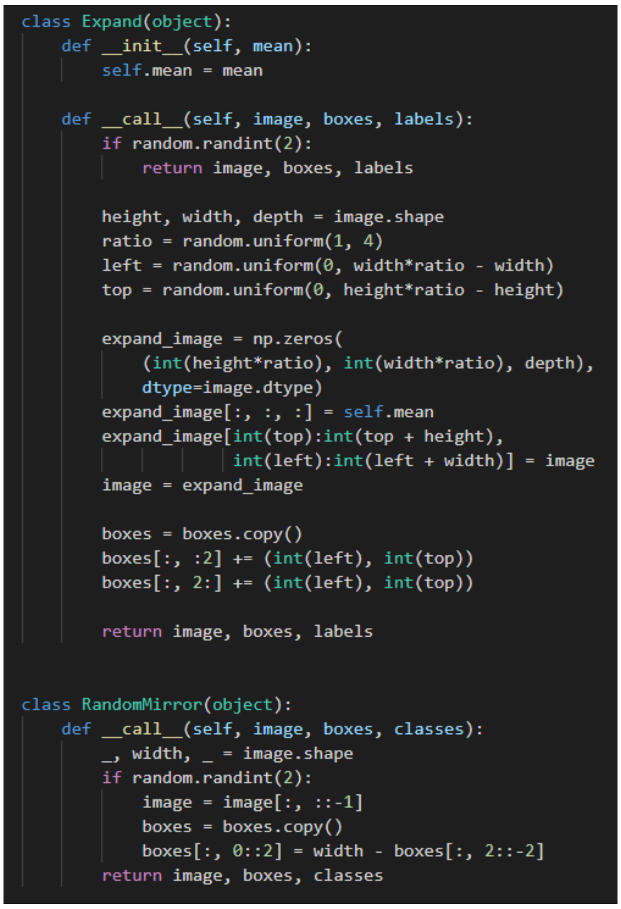
```
class RandomLightingNoise(object):
    def __init__(self):
        self.perms = ((0, 1, 2), (0, 2, 1),
                      (1, 0, 2), (1, 2, 0),
                      (2, 0, 1), (2, 1, 0))

    def __call__(self, image, boxes=None, labels=None):
        if random.randint(2):
            swap = self.perms[random.randint(len(self.perms))]
            shuffle = SwapChannels(swap)  # shuffle channels
            image = shuffle(image)
        return image, boxes, labels


class ConvertColor(object):
    def __init__(self, current, transform):
        self.transform = transform
        self.current = current

    def __call__(self, image, boxes=None, labels=None):
        if self.current == 'BGR' and self.transform == 'HSV':
            image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        elif self.current == 'RGB' and self.transform == 'HSV':
            image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
        elif self.current == 'BGR' and self.transform == 'RGB':
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        elif self.current == 'HSV' and self.transform == 'BGR':
            image = cv2.cvtColor(image, cv2.COLOR_HSV2BGR)
        elif self.current == 'HSV' and self.transform == "RGB":
            image = cv2.cvtColor(image, cv2.COLOR_HSV2RGB)
        else:
            raise NotImplementedError
        return image, boxes, labels
```

Appendix D.4 random lighting noise and colour conversion



```python
class RandomContrast(object):
    def __init__(self, lower=0.5, upper=1.5):
        self.lower = lower
        self.upper = upper
        assert self.upper >= self.lower, "contrast upper must be >= lower."
        assert self.lower >= 0, "contrast lower must be non-negative."

    # expects float image
    def __call__(self, image, boxes=None, labels=None):
        if random.randint(2):
            alpha = random.uniform(self.lower, self.upper)
            image *= alpha
        return image, boxes, labels


class RandomBrightness(object):
    def __init__(self, delta=32):
        assert delta >= 0.0
        assert delta <= 255.0
        self.delta = delta

    def __call__(self, image, boxes=None, labels=None):
        if random.randint(2):
            delta = random.uniform(-self.delta, self.delta)
            image += delta
        return image, boxes, labels
```

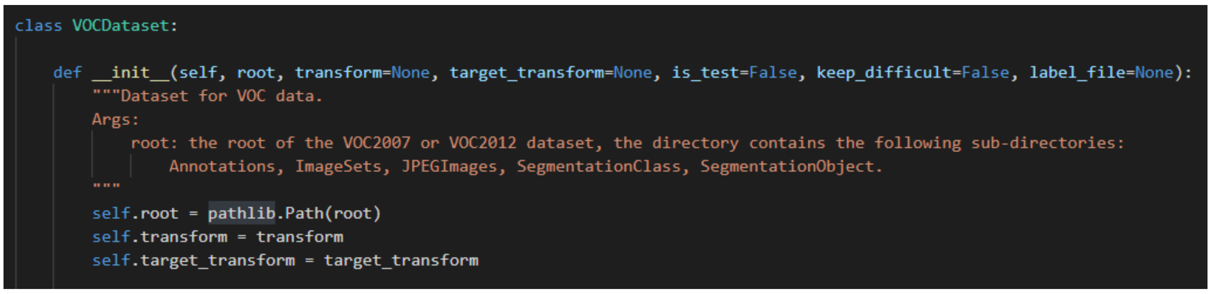
Appendix D.5 random contrast and brightness



```python
class Expand(object):
    def __init__(self, mean):
        self.mean = mean

    def __call__(self, image, boxes, labels):
        if random.randint(2):
            return image, boxes, labels

        height, width, depth = image.shape
        ratio = random.uniform(1, 4)
        left = random.uniform(0, width*ratio - width)
        top = random.uniform(0, height*ratio - height)

        expand_image = np.zeros(
            (int(height*ratio), int(width*ratio), depth),
            dtype=image.dtype)
        expand_image[:, :, :] = self.mean
        expand_image[int(top):int(top + height),
                        int(left):int(left + width)] = image
        image = expand_image

        boxes = boxes.copy()
        boxes[:, :2] += (int(left), int(top))
        boxes[:, 2:] += (int(left), int(top))

        return image, boxes, labels


class RandomMirror(object):
    def __call__(self, image, boxes, classes):
        _, width, _ = image.shape
        if random.randint(2):
            image = image[:, ::-1]
            boxes = boxes.copy()
            boxes[:, 0::2] = width - boxes[:, 2::-2]
        return image, boxes, classes
```

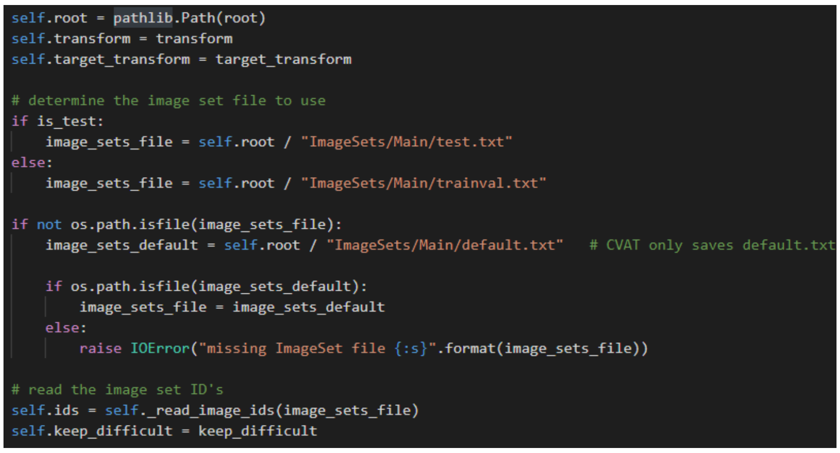
Appendix D.6 Expand and random mirror augmentation

Appendix E File structure and Image resize



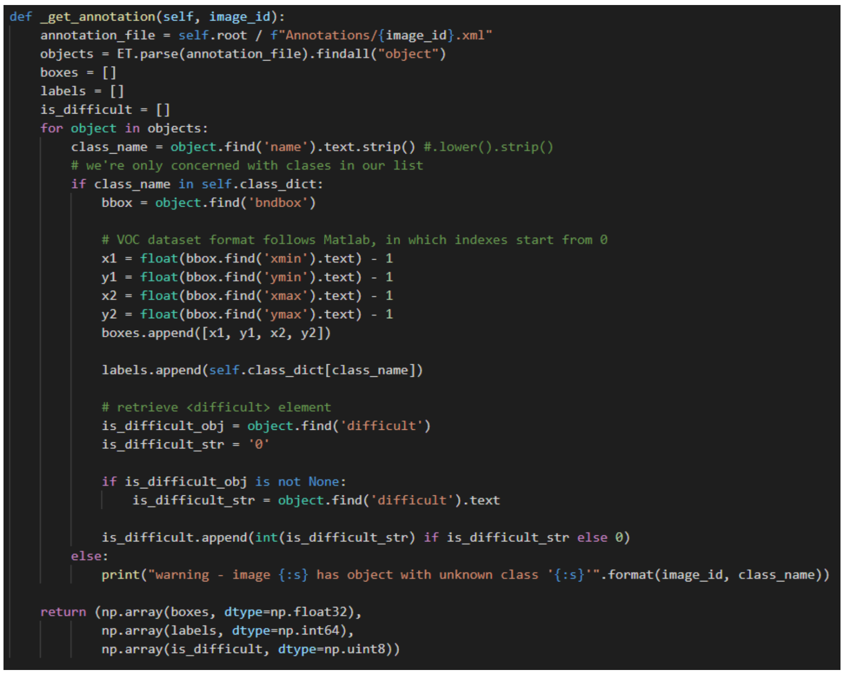
```python
class VOCDataset:

    def __init__(self, root, transform=None, target_transform=None, is_test=False, keep_difficult=False, label_file=None):
        """Dataset for VOC data.
        Args:
            root: the root of the VOC2007 or VOC2012 dataset, the directory contains the following sub-directories:
                Annotations, ImageSets, JPEGImages, SegmentationClass, SegmentationObject.
        """
        self.root = pathlib.Path(root)
        self.transform = transform
        self.target_transform = target_transform
```

Appendix E.1 Directories declared in root



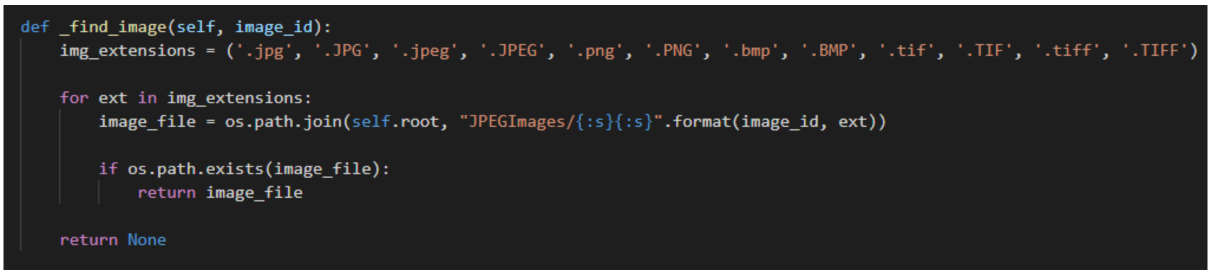
```python
self.root = pathlib.Path(root)
self.transform = transform
self.target_transform = target_transform

# determine the image set file to use
if is_test:
    image_sets_file = self.root / "ImageSets/Main/test.txt"
else:
    image_sets_file = self.root / "ImageSets/Main/trainval.txt"

if not os.path.isfile(image_sets_file):
    image_sets_default = self.root / "ImageSets/Main/default.txt"   # CVAT only saves default.txt

    if os.path.isfile(image_sets_default):
        image_sets_file = image_sets_default
    else:
        raise IOError("missing ImageSet file {:s}".format(image_sets_file))

# read the image set ID's
self.ids = self._read_image_ids(image_sets_file)
self.keep_difficult = keep_difficult
```

Appendix E.2 Gathering image id from ImageSets folders



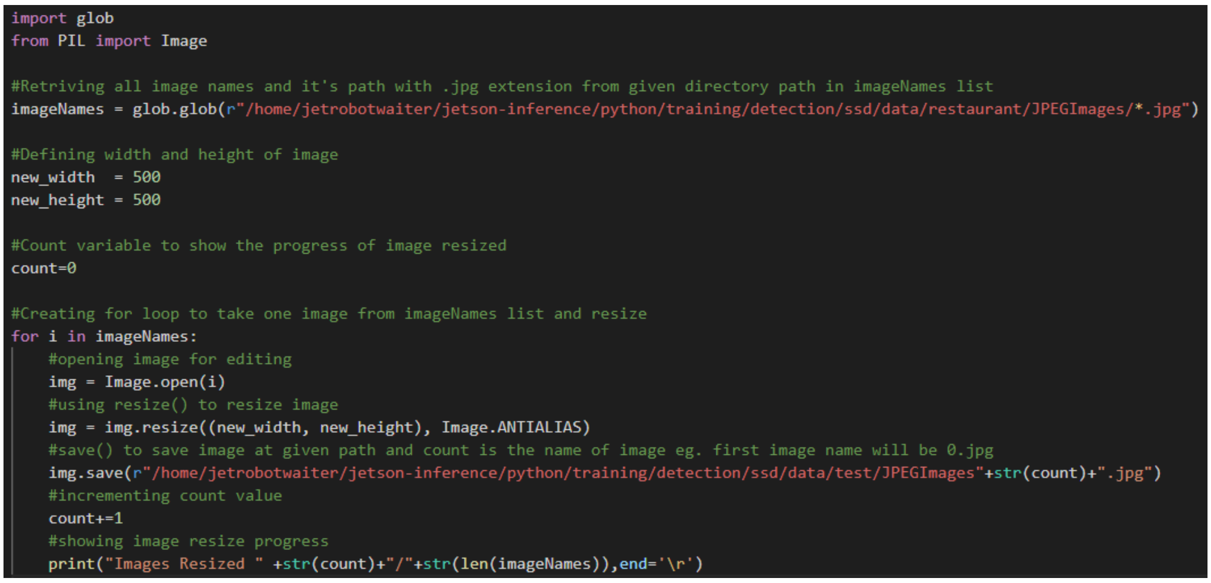
```python
def _get_annotation(self, image_id):
    annotation_file = self.root / f"Annotations/{image_id}.xml"
    objects = ET.parse(annotation_file).findall("object")
    boxes = []
    labels = []
    is_difficult = []
    for object in objects:
        class_name = object.find('name').text.strip() #.lower().strip()
        # we're only concerned with clases in our list
        if class_name in self.class_dict:
            bbox = object.find('bndbox')

            # VOC dataset format follows Matlab, in which indexes start from 0
            x1 = float(bbox.find('xmin').text) - 1
            y1 = float(bbox.find('ymin').text) - 1
            x2 = float(bbox.find('xmax').text) - 1
            y2 = float(bbox.find('ymax').text) - 1
            boxes.append([x1, y1, x2, y2])

            labels.append(self.class_dict[class_name])

            # retrieve <difficult> element
            is_difficult_obj = object.find('difficult')
            is_difficult_str = '0'

            if is_difficult_obj is not None:
                is_difficult_str = object.find('difficult').text

            is_difficult.append(int(is_difficult_str) if is_difficult_str else 0)
        else:
            print("warning - image {:s} has object with unknown class '{:s}'".format(image_id, class_name))

    return (np.array(boxes, dtype=np.float32),
            np.array(labels, dtype=np.int64),
            np.array(is_difficult, dtype=np.uint8))
```

Appendix E.3 Gathering Annotations using image id



```python
def _find_image(self, image_id):
    img_extensions = ('.jpg', '.JPG', '.jpeg', '.JPEG', '.png', '.PNG', '.bmp', '.BMP', '.tif', '.TIF', '.tiff', '.TIFF')

    for ext in img_extensions:
        image_file = os.path.join(self.root, "JPEGImages/{:s}{:s}".format(image_id, ext))

        if os.path.exists(image_file):
            return image_file

    return None
```

Appendix E.4 Gathering images based on the image id



```python
import glob
from PIL import Image

#Retriving all image names and it's path with .jpg extension from given directory path in imageNames list
imageNames = glob.glob(r"/home/jetrobotwaiter/jetson-inference/python/training/detection/ssd/data/restaurant/JPEGImages/*.jpg")

#Defining width and height of image
new_width  = 500
new_height = 500

#Count variable to show the progress of image resized
count=0

#Creating for loop to take one image from imageNames list and resize
for i in imageNames:
    #opening image for editing
    img = Image.open(i)
    #using resize() to resize image
    img = img.resize((new_width, new_height), Image.ANTIALIAS)
    #save() to save image at given path and count is the name of image eg. first image name will be 0.jpg
    img.save(r"/home/jetrobotwaiter/jetson-inference/python/training/detection/ssd/data/test/JPEGImages"+str(count)+".jpg")
    #incrementing count value
    count+=1
    #showing image resize progress
    print("Images Resized " +str(count)+"/"+str(len(imageNames)),end='\r')
```

Appendix E.5 Image resize