Object Detection and Tracking in Computer Vision using Deep Learning

M. Engg. In Information Technology
Individual Project
Kushal Prakash
Kushal.prakash@stud.fra-uas.de
1429800
By Dr. Peter Nauth

Abstract- **In the rapidly advancing field of computer vision, the ability to accurately detect and track objects in a dynamic environment is a critical task with applications ranging from autonomous vehicles to surveillance systems. This paper presents a comprehensive study and implementation on the utilization of the You Only Look Once (YOLO) algorithm** [1]**, a state-of-the-art deep learning-based approach for real-time object detection and extends its application for robust object tracking with 3-D depth camera. We focus on the integration of multi-modal sensory data, specifically RGB (Red, Green, Blue) and depth information, to improve the detection accuracy and tracking stability to give consolidated result in diverse and challenging scenarios for people as subject** [2]**.**

Keywords- **Object Detection, Object Tracking, YOLO, Deep Learning, Computer Vision, Depth Sensors, Intel RealSense, Multi-modal Sensing, Autonomous intelligence.**

I. INTRODUCTION

Object detection and tracking are two of the most fundamental and challenging tasks in the field of computer vision. Object detection involves identifying and locating objects within an image or video frame, while object tracking is the process of following the detected objects as they move across frames over time.

The advent of deep learning has revolutionized the approach to object detection and tracking, leading to significant advancements and the development of highly accurate algorithms. Among these, the You Only Look Once (YOLO) family of algorithms has emerged as a prominent solution for real-time object detection. The YOLO algorithm reframes object detection as a single regression problem, directly predicting bounding boxes and class probabilities in one evaluation. This approach contrasts with earlier methods that typically involved a separate pipeline for generating region proposals followed by classification [3].

To improve the accuracy of detection and tracking in various scenarios we are employing a depth camera along with RGB camera. This improves the algorithms performance under different lighting conditions [2].

These tasks are pivotal for numerous applications, including autonomous driving, video surveillance, robotics, face recognition, autonomous in-store operations and augmented reality.

II. METHODOLOGY

In this section we will further discuss on Intel RealSense camera sensor, data collection, training the algorithm and confusion matrix.

A. Intel RealSense Depth camera

We are using the Intel RealSense Depth camera D435 for our project. The good range for the working of sensor is between 0.3 meter(s) to 3.0 meter(s). RGB sensor tops out at 30 frames per second (FPS), hence the entire project is using 30 FPS even when depth camera can do it even more. The RGB camera has the 2 Megapixel sensor. The field of view is 69” x 42”. We need faster USB 3.1 Gen1 cable to extract its full

capabilities, else we will not be able to get the complete data. The power is supplied via the USB cable.

There are two (InfraRed)IR cameras on either side of IR dot matrix projector which maps the surroundings in 3-D (3 dimensions). The difference in the distance of each IR dot is used to map the depth of the object and map the surroundings. The RGB camera is at the end, which is used to capture 2-D colour images [2].

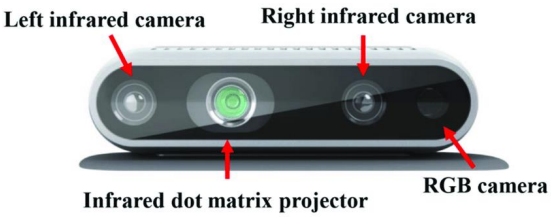

Figure 1: Intel RealSense camera

B. Measurement methods using RealSense Depth camera

Linux environment was used to interface with the sensor for capturing the data. As there was no RealSense depth sensor data available easily for training the algorithm on people images, we created our own dataset to train the YOLO algorithm. We collected data with different postures, multiple number of people of varied height, body shape, different lighting conditions, hairstyles, objects in hands, objects blocking people and so on at various distances.

Coco dataset was used to train the algorithm on RGB camera dataset.

C. Confusion Matrix

A confusion matrix is a table that is used to define the performance of a classification or machine learning algorithm. By comparing the predicted labels of a collection of data to the actual labels, a confusion matrix table that summarizes the performance of a classification method is obtained. An example of confusion matrix is shown in Figure 2.

		Predicted	
		Negative (**N**) -	Positive (**P**) +
Actual	Negative -	True Negatives (**TN**)	False Positives (**FP**) **Type I error**
	Positive +	False Negatives (**FN**) **Type II error**	True Positives (**TP**)

Figure 2: Confusion Matrix

The confusion matrix consists of four basic characteristics (numbers) that are used to define the measurement metrics of the classifier. These four numbers are:

1. *TP (True Positive):* When the expected value matches the actual value, we have a true positive case. This indicates that the model predicted a positive result, and the actual result is positive.
2. *TN (True Negative):* When the expected value and the real value are the same, a truly negative situation arises. This time, however, the model predicted a negative value, and the actual value is also negative.
3. *FP (False Positive):* When the expected value matches with an incorrect value, it known as a false positive. The model anticipated a favorable outcome even though the actual value was negative. This is called the type I prediction error.
4. *FN (False Negative):* When the expected value matches with an incorrect value, this is referred to as a false negative case. Even though the actual result was positive, the model predicted a negative result. This is the type II prediction error.

These four parameters are used to obtain other performance metrics which can be used to better judge the model.

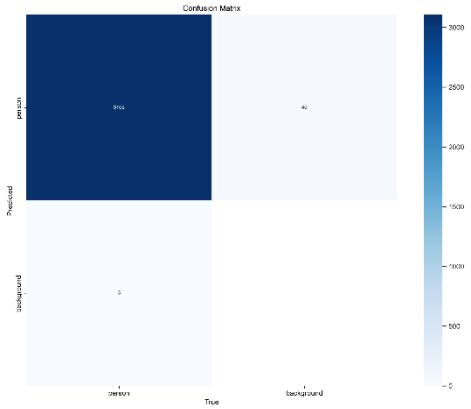

Figure 3: Confusion matrix for depth images

Some additional performance metrics of an algorithm are accuracy, precision, recall, and F1 score, which are calculated on the basis of the above-stated TP, TN, FP, and FN[7].

A number of significant metrics, including the following ones, can be computed using the confusion matrix to assess how well the classification algorithm performed:

Accuracy: The ratio of correctly predicted samples to all samples in the dataset is known as accuracy. It calculates the frequency with which the classifier predicts the right thing. The formula for accuracy is:

$$\text{Accuracy} = \frac{(TP+TN)}{(TP+FP+FN+TN)} \qquad \ldots(2)$$

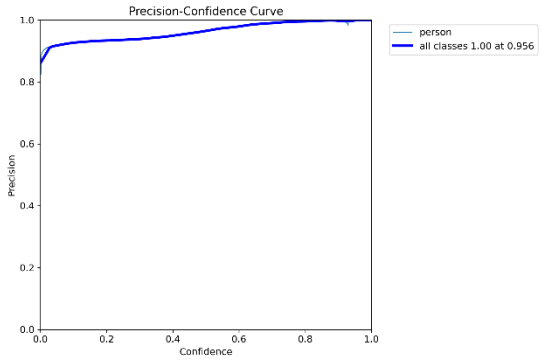

Figure 4: Precision curve

Precision: The ratio of accurately predicted positive samples to the total number of positive samples the model projected is known as precision. It counts the proportion of expectedly positive samples that really are positive. The precision formula is:

$$\text{Precision} = \frac{TP}{TP+FP} \qquad \ldots(3)$$

Recall (Sensitivity): The ratio of accurately predicted positive samples to all positive samples in the dataset is known as recall. It calculates the proportion of real positive samples that the model accurately predicted. The recall formula is as follows:

$$\text{Recall} = \frac{TP}{TP+FN} \qquad \ldots.(4)$$

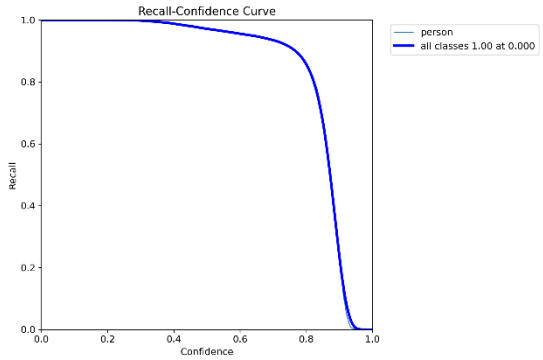

Figure 5: Recall curve

F1 Score: The harmonic mean of recall and precision is the F1 score [1]. When recall and precision are both significant, it is a useful metric. The F1 score formula is:

$$\text{F1Score} = \frac{2*precision*recall}{precision+recall} \qquad \ldots.(5)$$

Recognizing the widespread emphasis on accuracy when assessing performance is crucial before delving into confusion matrices, the foundation of model evaluation. Though it has its uses, precision by itself is not always reliable. Confusion matrices' applications and the drawbacks of accuracy as a stand-alone statistic will both be discussed in this section.

- *True Positive Rate (TPR):* Also known as recall, this metric represents the proportion of positive cases the model identified correctly (TP / (TP + FN)).

- *False Positive Rate (FPR):* This metric represents the proportion of negative cases the model incorrectly classified as positive (FP / (TN + FP)).
- *True Negative Rate (TNR):* Also known as specificity, this metric represents the proportion of negative cases the model identified correctly (TN / (TN + FP)).
- *False Negative Rate (FNR):* This metric represents the proportion of positive cases the model incorrectly classified as negative (FN / (TP + FN)).

Depending on the issue and application, these rates can be helpful for assessing different parts of a classification model's performance. The model's ability to correctly identify positive cases is measured by TPR, while its inclined to incorrectly classify negative circumstances as positive is measured by FPR. The model's capacity to accurately identify negative scenarios is measured by TNR, while its inclined to incorrectly classify positive events as negative is measured by FNR. To create a high-performing classification model, we generally wish to maximize TPR and TNR while reducing FPR and FNR [5].

D. Model training

Machine learning, a cornerstone of artificial intelligence, delves into creating algorithms and models that equip computers with the ability to learn from data. This enables them to make predictions or judgments without relying on explicit programming. In essence, machine learning empowers computers to learn from examples and experience, shifting away from dependence on pre-defined rules or constant human intervention.

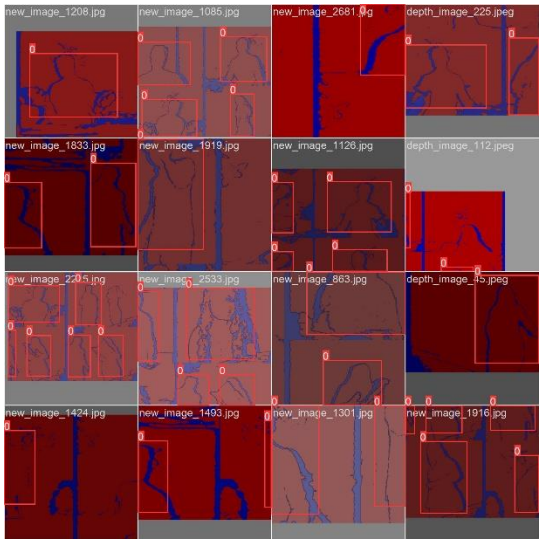

Figure 6: A batch of training data

The process of providing data to an ML algorithm in order to assist in identifying and teaching it suitable values for all related attributes is known as model training in machine language. While there are many other kinds of machine learning models, supervised and unsupervised learning are the most widely used.

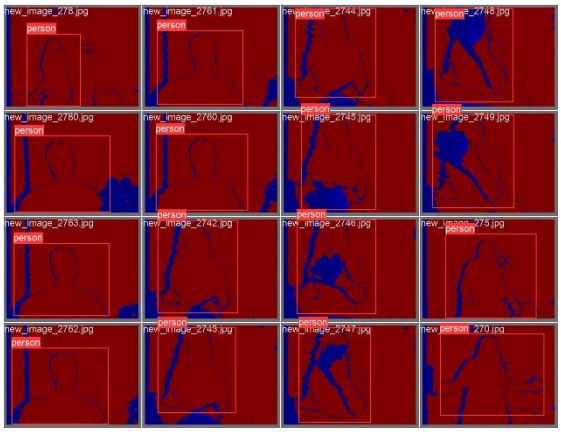

Figure 7: A batch of training label

When both the input and output values are included in the training data, supervised learning can take place. A supervisory signal is any piece of data that has both the inputs and the anticipated result. When inputs are given into the model, the training process is dependent on how much the processed result deviates from the documented outcome [6].

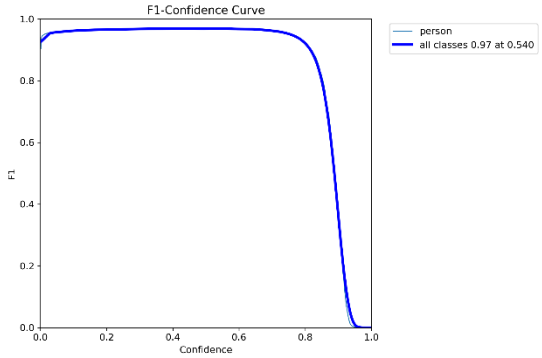

Figure 8: F1 curve

We are using cvat tool for annotating the raw data to mark the object under training. Then the data is exported as per the yolo format necessary for the training.

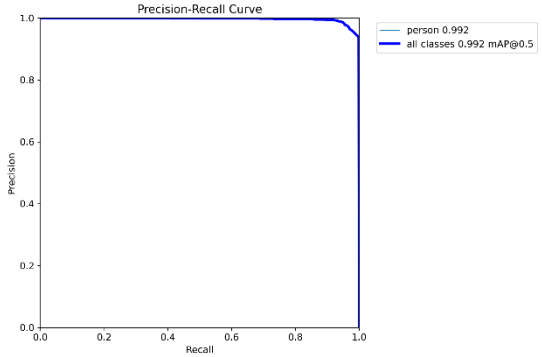

Figure 9: Precision recall curve

The data is trained for 100 epochs using the ultralytics packages for YOLOv8 [7]. 100 epochs is a good balance for object detection while not over fitting the data. Yolov8n is the standard parameters settings which is optimal for object detection tasks.

III. IMPLEMENTATION

A. Experimental setup and sensor placement

The sensor was placed at 100 centimetres from floor and the objects are moved continuously from 10 centimetres to 300 centimetres. Different profiles were also considered for making the algorithm be more robust.

For making the algorithm be aware of various background objects, algorithm was trained with images from different locations.

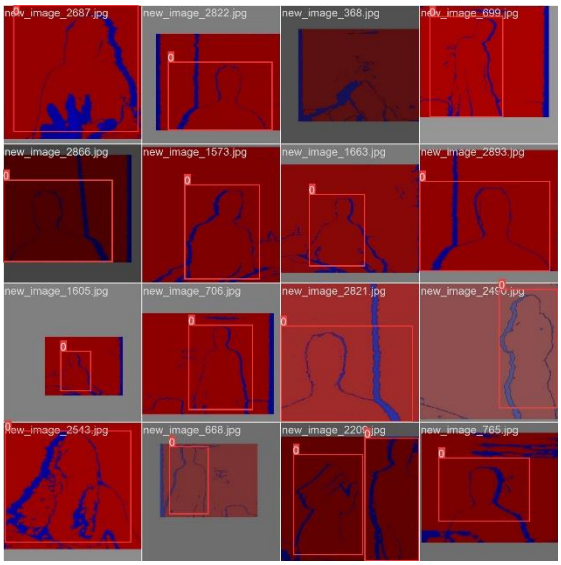

Figure 10: Some examples of the images used for training

B. Data format and collection

Images captured are of jpeg format with a resolution of 640x480. The format of data is chosen to be z16, this allows for good bitrate for data capture which helps in retaining most amount of detail.

The image is converted using absolute scale with no scaling. Then equalize the histogram to retain more information.

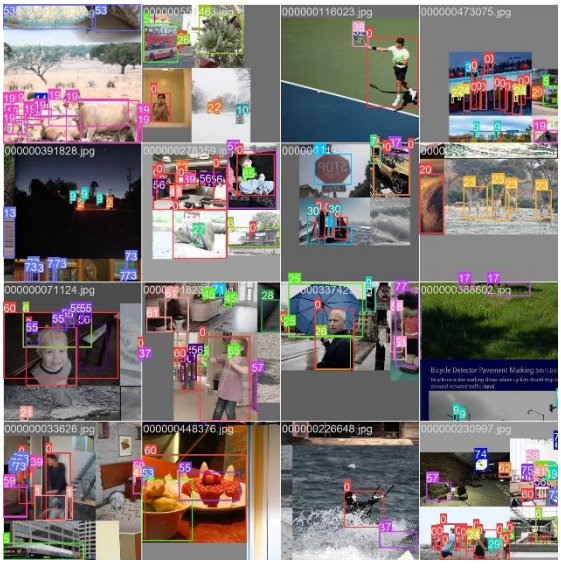

Figure 11: COCO dataset with annotation

IV. YOLO algorithm

The YOLO (You Only Look Once) algorithm is a state-of-the-art, real-time object detection system that has gained popularity due to its speed and accuracy. It was first introduced by Joseph Redmon et al. in a paper published in 2016 [2]. Unlike traditional object detection methods that apply a classifier to various regions of an image, YOLO frames object detection as a regression problem and performs detection in a single pass through the neural network, hence the name "You Only Look Once."

YOLO divides the input image into a grid, and each grid cell predicts a certain number of bounding boxes, confidence scores for those boxes, and class probabilities. The confidence score reflects how certain the model is that a box contains an object and also how accurate it thinks the box is that it predicts [8]. Class probabilities are conditioned on the grid cell containing an object. The predictions from all grid cells are then combined to create the final detection results, with non-maximum suppression being used to prune overlapping boxes.

Through its various versions, YOLO has seen significant improvements. Starting from YOLOv1, the algorithm has been iteratively improved upon with YOLOv2 (also known as YOLO9000), YOLOv3, and subsequent versions like YOLOv4 and YOLOv5, each introducing various enhancements in speed, accuracy, and the ability to detect objects at different scales [3].

YOLOv8 represents the latest iteration in the YOLO series, building upon the success of its predecessors by offering improvements in speed, accuracy, and generalizability. YOLOv8 is designed to operate with a high degree of efficiency, making it suitable for deployment in systems where computational resources are limited, and real-time processing is essential.

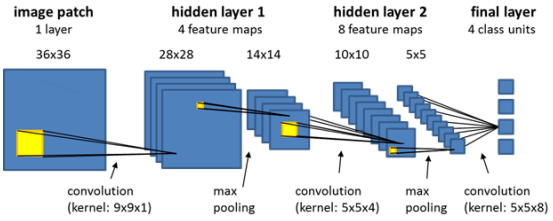

Figure 12: Convolutional Neural Network

YOLOv8, like its predecessors, uses a deep Convolutional Neural Network (CNN) to extract features from images and then applies a series of convolutions to predict bounding boxes and class probabilities. The architecture is typically composed of a backbone for feature extraction, a neck for feature aggregation and fusion, and a head for detection. The backbone is responsible for processing the input image and extracting a rich set of features, while the neck combines these features in a meaningful way to improve the detection of objects at various scales. Finally, the head of the network is responsible for making the final predictions [9].

One of the key aspects of YOLOv8 is its ability to handle a wide range of object sizes within an image, which is achieved through innovations in network design and training strategies. The algorithm has been trained on diverse datasets to ensure that it generalizes well to different environments and object types.

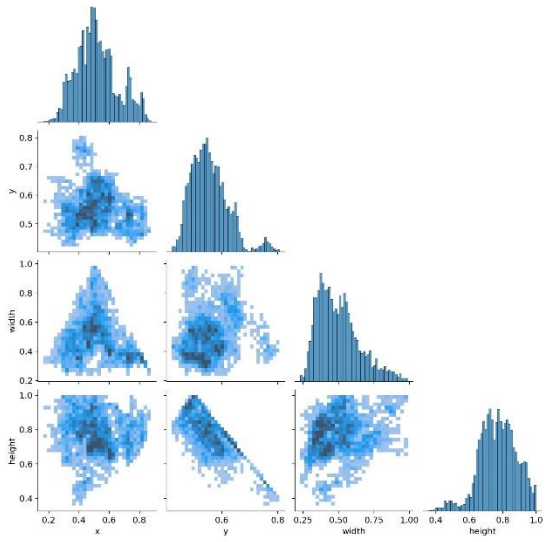

Figure 13: Label correlogram

In addition to the advancements in detection, object tracking also benefits from deep learning methodologies. Object tracking algorithms typically work by establishing the initial state of an object and then predicting its location in subsequent frames. Deep learning-based trackers use features learned by CNNs to maintain robust tracking even when objects undergo changes in appearance, scale, or occlusion.

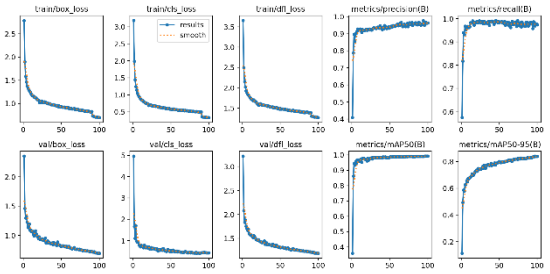

Figure 14: metrics from training for 100 epochs

Incorporating the YOLOv8 algorithm into a tracking framework allows for the seamless detection and re-identification of objects over time. This is particularly useful in complex scenes where objects may interact with one another or become temporarily obscured.

The integration of YOLOv8 with object tracking presents a powerful toolset for computer vision applications, enabling real-time performance without compromising on accuracy or robustness. This introduction sets the stage for a detailed exploration of the YOLOv8 algorithm's architecture, its implementation in object detection, and its role in enhancing object tracking methodologies.

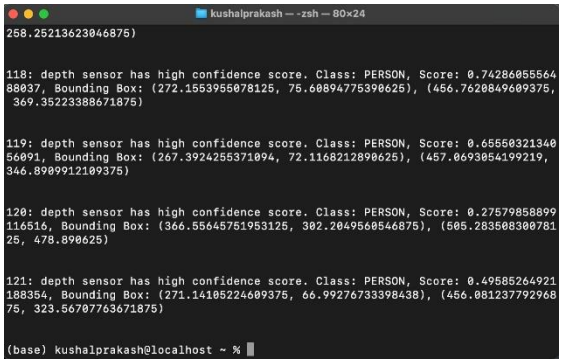

Figure 15: Result of our algorithm choosing the best sensor for each frame of object detection along with other crucial information

V. LIMITATIONS

The intel RealSense D435 sensor is currently not maintained by the manufacturer. Hence drivers are not available for latest versions of operating systems on all platforms such as Linux, Mac and Windows.

As we are running the algorithm on an ARM architecture processor, there is lack of tools and software support for this architecture, hence we can not run both RGB camera and depth camera together. So same video is recorded separately and then fed for the predict function.

When two similar or dissimilar objects are overlapping the sensor frame in close proximity or next to each other (single silhouette) then depth camera will fail to detect the object due to its limited resolution.

Farther the distance of object from sensor, lesser is the resolution of the object depth information captured.

Due to this limitation, we were not able to combine both the data and have an individual function and have a combined result during runtime. Despite this, we were able to see the algorithm predicting the result and we using it to make the best decision possible.

VI. RESULTS

After training for 100 epochs, some of the metrics are as follows. The box loss was reduced to 0.6965, class loss was reduced to 0.333, dfl loss was 1.27, precision is 96.38% and recall is 0.974.

All the current projects surrounding YOLO algorithm is for the data from RGB sensor. But from the experiments conducted, we can conclude that combining RGB sensor with depth camera will help in detecting objects under difficult lighting circumstances. While RGB sensor was able to detect persons under normal lighting conditions better than depth sensor, but when the light is low, then depth sensor has the edge in finding the object also since there is no colour component in the depth images, model does not need large data to detect an object. Such a system is very capable

in detection and tracking objects in autonomous vehicles.

VII. FUTURE IMPROVEMENTS

This project and the report can be used to further extend the algorithm to detect other objects using depth camera, which can make the algorithm more robust and useful for wider applications.

Newer and more advanced sensor can be used where both the RGB and depth sensors are supported in single platform, which can be used to run combined thresholding for detecting an object in runtime or on live video.

VIII. ACKNOWLEDGEMENT

We would like to express our gratitude to Prof. Dr. Peter Nauth and Sudeep Sharan for giving us the chance to work on this topic under their supervision and for guiding us in the right direction to finish the required project and report. We also want to express our gratitude for Prof. Julian Umansky important contribution in providing all the tools necessary for the task to be completed successfully. The topic " Object Detection and Tracking in Computer Vision using Deep Learning" were very beneficial in providing the background knowledge and motivation for learning more on machine learning topics and sparking interest in this field.

REFERENCES

[1] J. a. D. S. a. G. R. a. F. A. Redmon, "You Only Look Once: Unified, Real-Time Object Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[2] A. O. I. K. E. B. Z. K. P. O. Vladimir Tadic, "Application of Intel RealSense Cameras for Depth Image Generation in Robotics," *WSEAS TRANSACTIONS on COMPUTERS,* vol. 18, pp. 107-112, 2019.

[3] M. Hussain, "YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection," *MDPI,* vol. 11, no. 7, 2023.

[4] J. J. Sondi, "scaler," InterviewBit Technologies Pvt. Ltd, 29 11 2023. [Online]. Available: https://www.scaler.com/topics/r-logistic-regression/. [Accessed 12 05 202].

[5] Guo, P.; Shi, H.; Wang, S.; Tang, L.; Wang, Z., "An ROS Architecture for Autonomous Mobile Robots with UCAR Platforms in Smart Restaurants. Machines," Machines, 2022. [Online]. Available: https://doi.org/10.3390/machines10100844.

[6] N. H. L. Y. L. W. Z. a. M. D. Wang, "Segmentation and Phenotype Calculation of Rapeseed Pods Based on YOLO v8 and Mask R-Convolution Neural Networks," *MDPI,* vol. 12, no. 18, p. 3328, 2023.

[7] G. J. a. A. C. a. J. Qiu, "Ultralytics YOLOv8," AGPL-3.0, 2023.

[8] R. G. C. L. G. a. U. J. N. Pereira, "Sort and Deep-SORT Based Multi-Object Tracking for Mobile Robotics: Evaluation with New Data Association Metrics," *MDPI,* vol. 12, no. 3, p. 1319, 2022.

[9] D. E. F. L. Y. C. B. M. Peiyuan Jiang, "A Review of Yolo Algorithm Developments," *Procedia Computer Science,* vol. 199, pp. 1066-1073, 2022.