Radboud University Nijmegen

Faculty of Science

Object detection, classification and depth estimation approach deployed on excavators in simulated environments

How does an autonomous excavator detect traffic signs and estimate the depth to them in a simulated environment?

Bachelor's Thesis Computing Science

Author:
Ioan-Radu Bocu

First assessor:
dr. Yuliya Shapovalova

Second assessor:
dr. ir. Tom Classen

Daily supervisor:
Andreas Stavis

June 2025

Contents

Abstract

This thesis presents a method that is able to recognize traffic signs and estimate how far away they are from the excavator using ROS2, an open source framework that allows robotic components to collaborate with each other. The goal was to create a system that can both identify the traffic signs as well as estimate the depth to them using two cameras aligned at the same height. YOLOv8l which is a large object detection model, plays a crucial role in achieving a reliable traffic sign recognition system. Meanwhile, the depth estimation method based on computer vision concepts such as triangulation, disparity computation and camera calibration makes it possible for the system to tell how far away the traffic signs are.

First, this setup captures synchronized pairs of frames and rectifies them so that corresponding pixels lie on the same horizontal line. After this, traffic sign detection is applied using YOLOv8l and then for each detected traffic sign depth estimation is computed in real time. All these steps are conducted within the ROS2 framework. This system was then tested on a wide range of distances, from 31 cm to 150 cm. The results obtained suggest that our approach is reliable on short range, while the depth estimation errors get significantly larger on long ranges. This lack of precision on long distances is mainly caused by the low camera resolution and imprecise bounding box placement.

Despite all of these limitations, the final product shows good performance on short ranges which is highly-demanded in restricted environments where the distances between traffic signs is really small. The use of open source tools like ROS2 and the lightweight nature of the method make it attractive for autonomous driving systems, such as cars and excavators.

1 Introduction

1.1 Introduction

Automation is reshaping today's society and job market by changing the way tasks are carried out across different sectors[30]. This reduces the need for direct human involvement in both physical and cognitive processes. In the construction sector, 3D printing is used to speed up the building processes and also to reduce the need for manual labor [47, 29]. On the other hand, robots driven by AI methods are assisting in surgeries. This helps doctors conduct procedures with a greater accuracy and improve the patient recovery times [14, 13]. These are just a few examples showing that intelligent systems are starting to tackle tasks in environments that are often unpredictable. Another area where intelligent systems are feeling their presence is in autonomous vehicles, machines that make decisions on their own. Whether driving on roads(day to day cars) or on construction roads(excavators), these vehicles start handling more and more challenges in the real world.

Nowadays, self-driving cars are the first instance that comes to one's mind when it comes to autonomous vehicles. Despite this, they are far from the only example. Excavators, large and heavy machines used in construction, are more and more explored to be automated. Recent studies show growing interest in developing autonomous excavators that can perform operations such as digging, loading and navigation in the construction site without human involvement [17, 16, 46]. In spite of the fact that excavators operate in different environments than road vehicles, they face many of the same challenges. For instance, both instances need to be able to detect and interpret traffic signs, avoid obstacles and react to other visual stimulus. For example, traffic signs located on construction sites can indicate hazards, restricted areas or mandatory commands such as "Stop", exactly like on the public roads. So, given these similarities, many of the AI-driven methods enabled on self-driving cars, especially concerning object detection and depth estimation, are also applicable and useful for autonomous excavators. However, even though cars and excavators share a wide range of driving-related tasks, excavators often require slightly different techniques to accomplish them. This is mainly due to the rough and not well organized terrain they operate in, where dust, uneven terrain and less clearly marked signage can influence the visual perception in a negative way. Still these overlaps allow to build on top on existing research around autonomous vehicles in order to make excavators move and operate safely.

In order to overcome these visual perception challenges mentioned earlier, whether we talk about spotting a traffic sign or estimating how far away this traffic sign is, autonomous vehicles rely on depth estimation and object detection. Object detection helps the machine to localize and recognize traffic signs such as "stop" and "pedestrian crossing", while depth estimation tells how far those traffic signs are. The combined effort of these capabilities allows vehicles to make appropriate decisions and navigate safely through their environment. Behind the scenes, all these tasks are powered by deep learning, more specifically by neural networks that are able to recognize visual patterns like humans do. Thanks to the availability of the large datasets and hardware like GPUs, models have become really fast and precise enough to be responsive in real time in demanding circumstances like autonomous navigation [20]. So, these technologies are needful to make the excavator understand its surroundings and react accordingly.

All in all, this thesis will explore how an autonomous excavator can make use of computer vision for depth estimation and machine learning for object detection in order

to react to the surroundings, just like the systems deployed on autonomous vehicles. The focus in this thesis is centered on two fundamental tasks : detecting the traffic signs and estimating how far they are. These are necessary in places like construction sites where signs can mean restricted areas or required actions and, being able to estimate the distance to them enables making informed decisions(being able to in front of the "Stop" traffic sign at the right distance).[41]. To make this possible, we developed the system and tested it in a simulated environment in order to see how good the excavator is at detecting traffic signs and estimating the distance to them. A key part of this thesis was also the evaluation of multiple object detection models to find one that is both accurate and fast enough computationally. By combining the chosen model for object detection with the depth estimation technique, we aim to turn into reality the autonomous navigation in the construction domain. We will take into account research in autonomous driving, while adapting to the specific requirements of the excavators.

1.2 Related work

In recent years there has been a lot of progress in developing systems that make vehicles understand and respond to their surroundings, especially using traffic sign detection. This is one of the main pillars of autonomous driving which is directly related to the goals of this thesis which further focus on making excavators able to recognize signs and tell how far away they are.

A first important study by Alawaji [1] introduced a traffic sign recognition system which is based on multi-task learning. Their approach was to combine several deep learning models which achieved a high accuracy in real-world tests. The model they trained was able to detect both traffic signs and light and performed quite well even in challenging outdoor conditions. However, their focus was mainly on road vehicles like cars and they did not take into account the distance to the traffic signs.

Another relevant work was conducted by Marques et al. [25] who used YOLOv3 to detect traffic signs and also lights in both real world and simulated scenarios. Their solution made use of a module that allows the robot to take certain actions in terms of the traffic sign detected. While their method worked pretty well in real time and had solid performance, it was not designed to be used in the rough conditions of a construction site and did not include any form of depth estimation.

Kumar et al. [21] took a different approach by testing the capsule networks, a type of neural network designed to better handle changes in the position and orientation of objects. Their model performed well in terms of classifying traffic signs and was even more robust than traditional CNNs. However, their work focused only on classification and did not include traffic sign detection or some kind of testing on an autonomous system.

Another study by Toshniwal et al. [42] worked with the German Traffic Sign Recognition Benchmark dataset and used preprocessing methods like histogram equalization in order to improve accuracy. They ended up achieving a high precision and was fast enough to be used in real time. However, the study focused on standardized datasets and did not explore how the system would perform in more complex environments. They also did not estimate the distance to the traffic signs.

A study by Saleh et al. shows a traffic sign detection system that managed to achieve a 97% accuracy in real world settings [19]. Their depth estimation relied on a monocular(single) camera setup. The idea was to capture more frames of the same traffic sign and compute the distance to it based on the area of the bounding box provided by the classifier. This approach works mainly when the area of traffic signs is constant and the navigation surface is flat. This makes their approach less reliable in less organized environments like construction sites, where elevation level can vary, objects are placed at different angles and lighting fluctuates. Monocular systems cannot perceive depth, they can only provide rough estimates of the distance to the traffic signs. In contrast, stereo vision is able to capture depth by mimicking the human binocular sight [32]. This alternative is more suitable for excavators operating in unstructured environments where knowing the distance to obstacles allows making intelligent decisions.

To sum up, while many studies have developed quite impressive systems for detecting and recognizing traffic signs, most of them still focus on regular road environments and perform depth estimation using monocular vision, which is not as accurate as binocular stereo vision.

1.3 Motivation

Even though robotics and autonomous navigation have evolved significantly over the last years, there is still a gap when it comes to bringing multiple technologies together into one operational system. Topics like object detection and stereo vision are usually studied separately, but combining them is what rises a real challenge, especially for machines that operate in messy, unpredictable environments. For instance, within a construction site, traffic signs may not be placed clearly like on regular roads, and, often, people walk freely in the same areas where vehicles operate. This makes it hard for autonomous systems to make safe, reliable decisions. In contrast to autonomous cars that can rely on GPS, autonomous excavators can function in places where GPS signals are weak or inexistent and there are no ready-made maps to follow. This means that the system has to rely entirely on its own sensors in order to make decisions in real time.

With my thesis, I aim to bring my own contribution by developing a real-time system for an autonomous excavator using ROS2, stereo camera depth estimation and YOLO-based object detection. The final product will not just be able to recognize traffic signs, but also compute the distance to the detected objects which allows making informed decisions in time. By fusing these components into a single integrated system I hope to demonstrate that autonomous intelligent navigation is achievable even in rough terrains like construction sites. This project is a step toward obtaining autonomous navigation on more specialized vehicles like excavators. In order to complete it successfully, we need a deep understanding of some fundamental AI-related and stereo vision concepts.

2 Preliminaries

2.1 Knowledge flow

In order to guide the reader through the technical background for this thesis, Figure 1 was created. This presents a "studying map" that shows the key concepts that represent the foundation of the methods used in this thesis and ,also how they relate to each other. I have organized the diagram in such a way that it reflects the logical flow of knowledge necessary to understand the techniques used in this project.

The map begins with Supervised Learning, which introduces the basic idea of training models on labeled data. From there, we move to the Artificial Neural Networks(ANN) which is the fundamental architecture used in many supervised learning tasks. Then, ANNs represent the prerequisite for Convolutional Neural Networks(CNN), which were specifically created for processing image data. The family of models "You only look once" (YOLO) evaluated in this project is based on CNN. CNNs are required in solving visual tasks like Object Detection, which is one of the goals in this project. In parallel, it is also fundamental to understand how to evaluate your model via metrics like precision, recall and F1-score. That is why, "Evaluation metrics" node is also linked to the "ANN" node which can also be seen in the diagram.

At the same level as the "Object detection" node, we can also see a separate concept which is stereo vision, subcategory of Computer Vision. This field provides many techniques to extract meaningful information from images, unrelated to machine learning. Stereo vision is the base for depth estimation, as it included methods like epipolar geometry and triangulation. These are indispensable when it comes to estimating how far objects are.

Finally, object detection and computer vision point to the final goal of this thesis : enabling an autonomous excavator to recognize traffic signs and compute the distance in real time to them. These combined methods allow the system to perceive its surroundings and make informed decisions without the need to rely on pre-built maps or GPS. First thing that will be studied is Supervised Learning, most fundamental concept of the studying map.

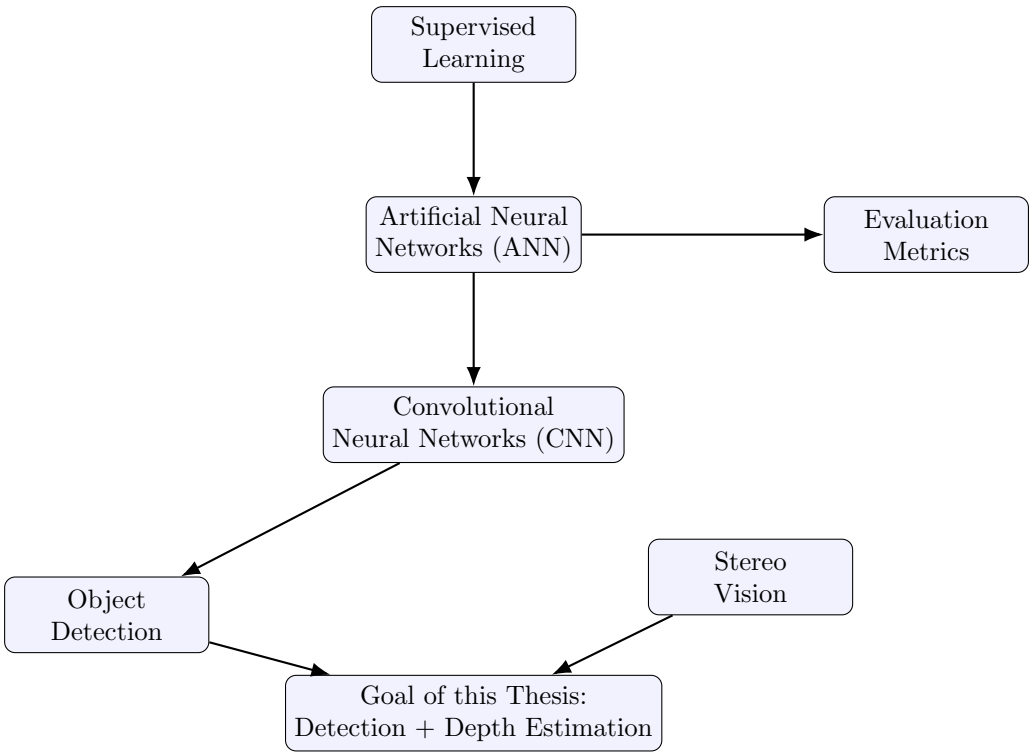


Figure 1: Concept map showing prerequisite knowledge and relationships between key topics. The most fundamental concept that stands on the top of the tree is "Supervised Learning", while at the end of the tree we see the goal of the thesis which can be achieved by understanding all concepts above it.

2.2 Supervised Learning

Supervised learning is one of the main types of machine learning and the fundamental concept sitting at the root of Figure 1. It implies training a model on a labeled dataset. A labeled dataset is a data set that assigns every combination of inputs a corresponding output [26]. For instance, our dataset is annotated as it will be later discussed, which makes it suitable for a supervised learning task like ours to train and evaluate multiple models. The objective of this type of learning is that the model learns the strategic patterns during training phase and then can make good predictions on unseen data. This property is called generalization, the ability of a model to behave well on unseen data. In the end, the model should understand general patterns within training set in order to be able to assign proper predictions to unseen data [38]. The loss function is used to evealute the generalization capability and it computes the difference between predictions and ground truth labels. Understanding supervised learning is needful before diving into more complex model architectures like neural networks.

2.3 Artificial Neural Networks

Artificial neural networks (ANN) are computational models used in supervised learning which output a prediction, based on multiple input features. Their structure is inspired by the human brain. In the same way, our brains consist of neurons, ANNs are made of multiple interconnected layers containing nodes (called neurons). The information is processed mathematically and is forwarded from one layer to the next one.

Each connection between neurons carries a weight which gives the importance of the input. During the learning process, these weights are adjusted such that the prediction is matched with the ground truth. This is done in order to improve future predictions. The network learns to associate inputs with outputs through repeated exposure to data, gradually improving its performance.

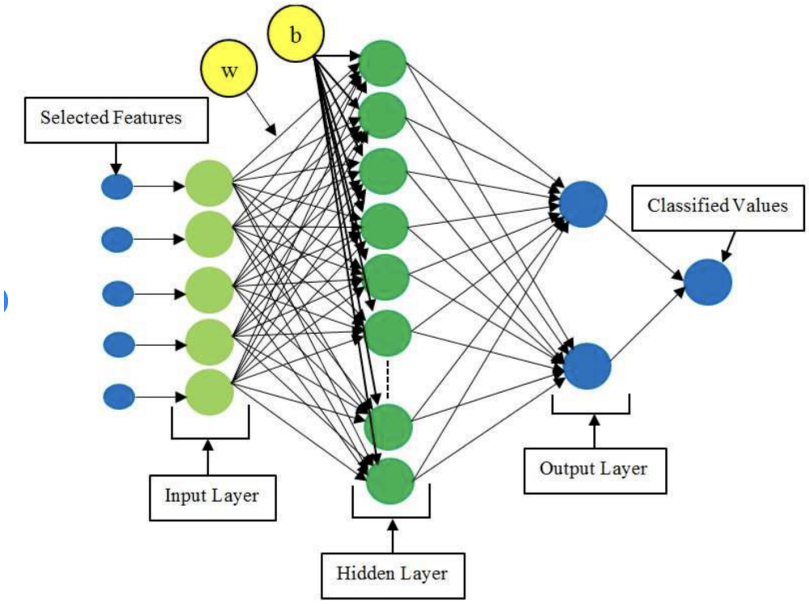


Figure 2: Architecture of a feedforward artificial neural network(ANN), containing an input layer, a hidden layer and an output layer. Each circle in the figure is a neuron and the arrows are weighted connections between layers. This architecture is usually used for classification to which the selected features contribute to the prediction of classes. Adapted from [34].

ANNs prove to be effective for tasks where the association between inputs and outputs cannot be explained by simple and strict rules as in the case of image classification, speech recognition, and object detection. One review by Schmidhuber (2015) presents a deep overview and comprehensive history of deep neural networks and how they evolved to become one of the pillars in modern AI applications [39]. Another relevant study by LeCun, Bengio, and Hinton (2015) discusses the success of deep learning, a subset of neural networks, in solving real-world perception problems [22].

Since the object detection models we will evaluate later are based on ANN architecture, it is crucial to understand how ANNs work, how information flows through neurons and how weights are adjusted and how predictions are made. This serves as the conceptual foundation for understanding more advanced models like convolutional neural networks.

2.4 Convolutional Neural Networks

While traditional ANNs work well with structured data, they struggle with images due to the high dimensionality issue, as each particular pixel needs to be provided as a feature to the model input layer. Convolutional neural networks(CNN) represent a specific category of neural networks and address the high dimensionality issue as they are

able to process grid like data in a more structured manner. Their architecture preserves spacial relationships through local connectivity (by making use of the fact that pixels relate to each other). This makes them effective in tasks involving visual recognition, such as ours. The models we will look into later are based on this fundamental CNN structure. First, we start looking into the internals of a mainstream CNN.

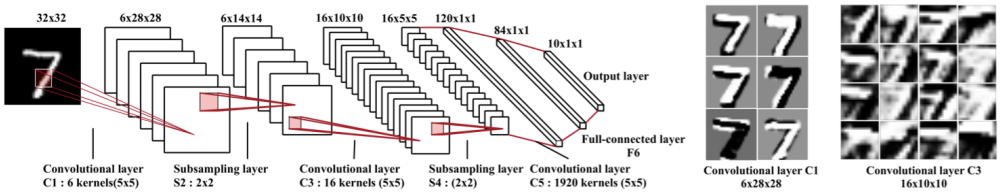


Figure 3: The architecture of a convolution neural network(CNN), used for image classification task. The input is of size 32×32, gray image, followed by 7 convolutional layers (subsampling) and fully connected layers. Each convolutional extracts more and more complex features using multiple learned kernels. Subsampling implies reducing spatial dimensions, while retaining information. The last fully connected layers are mapping extracted featured with the corresponding classes. The right side of the figure highlights how the network learns progressively. Adapted from Gu et al. (2015) [10].

A regular CNN consists of three main type of layers : convolutional, pooling and fully connected. The convolutional layers uses a set of trainable filters(kernels) that slide over each individual region in a picture in order to extract patterns and features. These layers exploit spatial locality by connecting only a subset of neurons in each layer to the neurons in the previous layer. The learned filters activate once specific patterns are detected like edges or textures. Non-linear activation functions such as ReLU are often applied after convolution in order to introduce non-linearity and improve training efficiency [9]. Pooling layers are applied after the convolution phase and are used to reduce spatial dimensionality of the feature maps. This is good, because it reduces the number of parameters which leads to a lower computation time and prevents overfitting [2]. Finally, fully connected layers, the last part of the architecture, take the high-level features and perform classification. Some networks, such as YOLOv3, detach themselves from the regular paradigm, by deploying a fully convolutional design that makes predictions possible at multiple scales. This makes the model present a higher performance when dealing with object detection tasks in images with varying resolutions [36]. This is also required in our case as traffic signs can have different sizes across the world. The concept of receptive fields is critical in CNNs : each neuron in a convolutional layer is connected to a specific region of the input. Deeper layers stack information over increasingly larger potions of the input space. The size of the output feature map is influenced is influenced by hyperparameters such as kernel size, padding , stride, dilatation and follows the equation (1) :

$$\text{Output size} = \left\lfloor \frac{W - K + 2P}{S} \right\rfloor + 1 \tag{1}$$

where W is the input width, K is the kernel size, P is the padding, and S is the stride. These architectural principles allow CNNs to learn powerful hierarchical representations and perform with high accuracy across a broad range of computer vision tasks.

All in all, properties mentioned above make CNNs suitable for analyzing images which involve understanding patterns like shapes and edges. Because of this, CNNs are the fundamental brick of the most modern object detection systems, the topic we will explore next.

2.5 Object Detection

Object detection is one of the supporting pillar in computer vision and the main way that will help our excavator recognize traffic signs. It implies identifying and locating precisely types of objects within images. For instance, it allows a system to find and recognize objects by drawing bounding boxes around them. In the case of traffic sign detection , this means that the model could be able to draw boxes around "stop", "pedestrian crossing", "give way" and so on.

Early object detection relied on the following techniques:

- **Scale-invariant feature transform (SIFT)** [23]: Identifies key points in an image that are independent from scale and rotation.
- **Viola-Jones detector (2001)** [43]: Real-time face detection using a set of simple classifiers.
- **Histogram of oriented gradients (HOG)** [6]: Captures edge information which is then used for performing object detection.
- **Deformable part models (DPM)** [37]: Models objects as flexible parts and was state-of-the-art before deep learning.

Figure 4: Example of traffic sign recognition. Bounding boxes and corresponding positive confidence scores are placed over the detected traffic signs. For instance, the system identifies a pedestrian crossing with a confidence score of 0.94, maximum being 1. Also a roundabout traffic sign is identified with a probability of 0.82, and other 3 traffic signs with different probabilities.

A pivotal movement in object detection happened when deep learning mechanisms were introduced. One such approach is Region-based Convolutional Neural Networks (R-CNN) that appeared in year 2014 [8]. These models learn visual patterns automatically from large datasets, outperforming its ancestors.

Deep Learning-based detectors are divided into 2 categories:

- **Two-stage detectors** : This includes R-CNN, Mask R-CNN, Fast R-CNN. In these models, regions of interest are first proposed then classified
- **One stage detectors**, like YOLO(You Only look Once), SSD(Single Shot Detector) which perform detection in one go. This makes them faster [35] and, thus,

suitable for real-world applications such as ours.

Overall, object detection in combination with deep learning is widely spread in real-world systems, including autonomous cars, surveillance and robotics.

Deep learning models like YOLO, which will be investigated later in this project, still generate multiple overlapping boxes per object. That's why they rely on non-maximum suppression algorithm(NMS) in order to keep only the best detection of each object [44]. Understanding how NMS works is indicated before diving into models like YOLO.

2.6 Non maximum suppression

Most of the object detection algorithms usually don't manage to select the most suitable bounding box for one object in the first attempt. Instead, they generate multiple bounding boxes for one object (in this case for every traffic sign detected). The best bounding box representation is selected by applying the non maximum suppression algorithm for every object [4].

The algorithm works as follows : it takes as input a set of proposed bounding boxes, their corresponding confidence scores and a threshold. The proposal with the highest confidence score is selected and added, then, to the final list of detections. Then, this selected box is compared with the remaining boxes in order to see if they make a reference to the same object. If another box overlaps too much with the already selected one, it is considered redundant and removed immediately from the list. This process continues, picking the next highest-confidence box and removing overlapping ones, until no boxes are left to check [12].

While techniques such as non maximum suppression try to improve model outputs during object detection, they still don't tell us how well the model performs overall. In order to determine how good these predictions are, especially in the case of autonomous navigation, we need proper evaluation metrics, which we will investigate next.

2.7 Evaluation Metrics

2.7.1 Introduction

Once a neural network has been trained then it is crucial to know how well it performs, especially when working with applications like traffic sign recognition that needs to be very safe to use. That's where evaluation metrics like precision, recall and F1-score come in. These metrics help us measure whether model's predictions are accurate, but also complete, bridging in this way the gap between the model architecture (as discussed in the ANN section) and its real-world usefulness. The metrics that will be discussed in this section are : confusion matrix, precision, recall, F1 score and average precision.

2.7.2 Confusion Matrix

A confusion matrix is characterised by 4 parameters:

- **True Positives**: when both predicted and true labels are positive
- **False Positive**: when predicted label is positive , but actual label is negative
- **False Negative** : when predicted label is negative, but true label is positive
- **True Negative** : when both predicted and actual label are negative

The confusion matrix corresponding to a model represents a helpful visualization tool. It can tell you where your model fails, which classes are better identified by your model, which classes may be confused by others by looking at the number of false positives. An example of a confusion matrix can be seen in Table 1. The y-axis represents the ground truth labels(actual) , while the x-axis gives the predicted labels. Ideally, the values belonging to the main diagonal of the matrix should be as high as possible, while the other values should be close to zero.

For instance, in the confusion matrix given as example, it can be seen that 4 "Yield" instance are classified as "Stop"., while 3 "Stop" instances are labeled as "Yield" by the model.

Actual / Predicted	**Stop**	**Yield**	**Speed Limit**
Stop	45	3	2
Yield	4	40	6
Speed Limit	1	5	44

Table 1: Confusion matrix showing predictions among three traffic sign classes: "Stop", "Yield" and "Speed Limit". Each row gives the actual class while each column indicates the predicted classes. This means that all elements on the main diagonal are correct predictions whereas all other elements represent misclassifications. For instance, out of 50 total traffic signs, 45 were correctly classified, 3 were incorrectly classified as "Yield", and 2 were misclassified as "Speed Limit".

2.7.3 Precision and Recall

Precision is the ratio between the correctly classified positive objects and all objects classified as positive. The precision indicates how accurate your model is , when it comes to predictions and it is calculated using the formula (2). For example, if a model detects 6 traffic signs within an image and 3 of which are correctly identified then the precision is $\frac{1}{2}$.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2}$$

Recall represents the ratio between the correctly classified positive samples and all objects whose actual label is positive. The recall shows how many positive objects your model missed in the classification process and it is calculated using the formula (3). For instance, if within an image, there are 6 correctly identified objects and others 3 positive objects that your model missed/misclassified, then the recall is $\frac{2}{3}$.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3}$$

2.7.4 F1 Score

F1-score is a combination between precision and recall into a single metric by computing their harmonic mean as it can be seen in formula (4). A high F1-score means that both precision and recall are high , while a low F1-score indicates a low or imbalanced precision and recall (or both metrics are quite low).

$$\text{F1} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{4}$$

2.7.5 Average Precision

Average Precision(AP) is a way to measure how good an object dector behaves. It looks at how precision and recall evolve when we adjust the confidence level of the predictions. A good object detector should keep both precision and recall high, even if the confidence threshold gets modified. In the ideal scenario, as the model finds more objects (maintains a high recall), the accuracy of those detections(precision) should not decrease too much. Instead of using integrals to calculate the area under the precision-recall curve, we usually use a simpler method called VOC Interpolation Unit. This method checks the precision at 11 points: 0.1, 0.2, 0.3, ..., all the way up to 1.0 recall. Then, it calculates the average of these values in order to obtain the average precision (see equation (5)).

$$\text{AP} = \sum_{r=\{0,0.1,\ldots,1\}} P_{\text{interpolated}}(r) \tag{5}$$

To get an overall score for the entire model, we need to determine the Mean Average Precision, which is the average of all APs belonging to all classes. Having all these metrics discussed above in mind, we now completely understand how realiable the predictions of our model are.

Now that we have covered the machine learning part, it is time to move to the next important part : stere vision. This will enable us to determine how far away the detected traffic signs are, which is important for an autonomous excavator.

2.8 Stereo Vision

The goal of the stereo vision is to compute depth, that is the distance from the cameras to the target point, which in our case is given by the traffic signs. This is the second building rock that contributes to the goal of the thesis. In order to be able to perform stereo vision we need a two stereo cameras setup aligned at the same height. Having these, we can start focusing on calibrating the cameras.

2.8.1 Camera Calibration

The first thing that needs to be done before proceeding with the task is to calibrate the cameras. By calibration, we mean that a number of intrinsic and extrinsic parameters needs to be determined. Intrinsic parameters are parameters related to the interior configuration of the cameras. Such parameters are : focal length on x direction (f_x), focal length on y direction (f_y), and the coordinates of the principal point (u_c, v_c). This principal point is usually the centre of the 2D frame, ($\frac{w}{2}, \frac{h}{2}$) (the picture itself). All these parameters are wrapped inside the calibration matrix, K. Calibration camera looks like this (see matrix (6)) :

$$\text{K} = \begin{bmatrix} f_x & 0 & u_c \\ 0 & f_y & v_c \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

The calibration matrix is a camera specific trait. So in order to perform stereo vision, the calibration matrixes for both left and right camera, K_{left} and K_{right}, need to be computed.

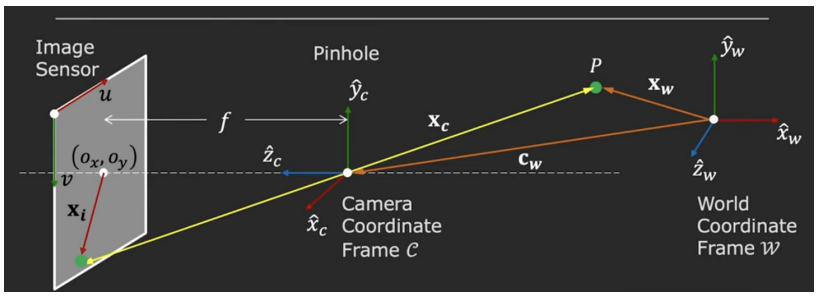


Figure 5: All three coordinate frames in stereo vision visualized. These coordinate frames are : image plane, camera coordinate frame and world frame. A 3D point P is taken from the world frame, projected in the camera frame and then onto the image plane, making use of the focal length f. The transformations represent the fundamental pillars of stereo vision. Adapted from [7].

In stereo vision we work with three coordinate systems(Figure 5) : the world coordinate frame(3D), the camera coordinate frame (3D) and the image plane (2D). In order to project a 3D point from to world onto the image plane, we apply two transformations:

- **Extrinsic parameters** : used to convert world coordinates into the camera coordinate frame. These parameters encode the camera's position and orientation in the physical space. They are given by a rotation matrix and a translation vector wrapped inside M_{ext}.
- **Intrinsic parameters**: transform 3D camera coordinates into 2D image coordinates. These depend from camera to camera and define its internal configurations, like focal length and principal point. Also, they are found in the M_{int}.

Together, the intrinsic and extrinsic parameters form the full projection matrix, as it can be seen in equation (7)**:**

$$\tilde{\mathbf{v}} = P \cdot \tilde{\mathbf{X}}_w, \quad \text{where} \quad P = M_{\text{int}} \cdot M_{\text{ext}} \tag{7}$$

$\tilde{\mathbf{X}}_w$ represents the homogeneous world coordinates, while $\tilde{\mathbf{v}}$ denotes the corresponding homogeneous pixel coordinates.

This equation lets us project any 3D point from the world directly onto the image plane .The projection matrix combines all necessary parameters in order to do the full transformation (world to image plane) in one go.

The next parameter to be determined is the fundamental matrix F. To compute it, we require two images taken from the same height which show the same scene. From these images, at least eight pairs of corresponding points , one from the left image and one from the right ,are selected. By "corresponding" we mean pixels that express the same information in both images. These point correspondences are then used in the following constraint equation (8):

$$\mathbf{x}_L^\top F \mathbf{x}_R = 0 \tag{8}$$

where $\mathbf{x}_L$ and $\mathbf{x}_R$ are the homogeneous coordinates of the matching pixels in the left and right images, respectively.

After inserting all eight point pairs into the equation above, we obtain a system of linear equations. Solving this system allows us to determine all the elements of the fundamental matrix F. This matrix plays a key role in finding correspondences between pixels in the left and right images.

Given a pixel in the left image and the fundamental matrix F, we can compute a line in the right image, called the epipolar line, on which the matching pixel must lie. This line has an equation of the form $ax + by = 0$, where x and y are coordinates in the right image. As a result, the search for a corresponding pixel is reduced from a two-dimensional problem to a one-dimensional search along this line.

Knowing the fundamental matrix, the essential matrix can be computed. The essential matrix is used similarly as the fundamental matrix, but it relates left camera coordinates with right camera coordinates according to the following equation (9) :

$$\mathbf{X}_L^\top E \mathbf{X}_R = 0 \tag{9}$$

In the end, having E, the translation T and rotation vector R can be immediately found using Singular Value Decomposition. This marks the end of the calibration process as we obtained all necessary parameters. These will further help us estimating the distance to a detected traffic sign using methods such as triangulation.

2.8.2 Triangulation and depth computation

Having all parameters determined from the previous step, the images can be rectified. This means that both images are cropped and rearranged such that corresponding pixels lie on the same epipolar line.

With this, finding 2 corresponding pixel points is straightforward since the search is limited to the same epipolar line. After finding the points at positions(x_l, y_l) and (x_r, y_r), the disparity can be computed like this: $|x_l - x_r|$. A higher disparity means that a point is closer to the camera, while a lower one indicates that target point is further away.

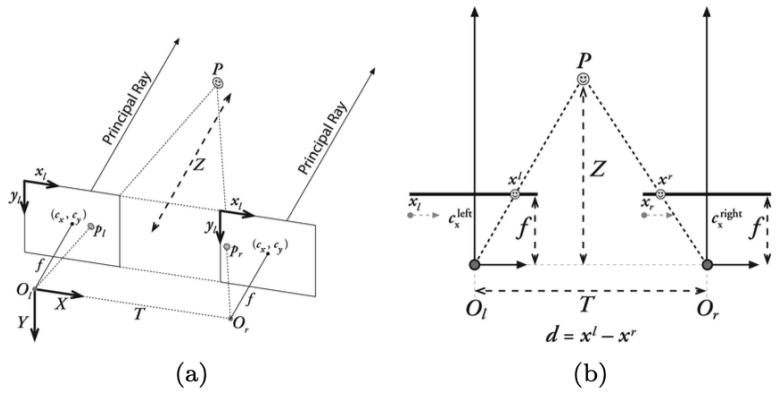


Figure 6: Visualization of epipolar geometry. Depth Z to target point P is computed using the disparity d, the shift of corresponding pixels on x-coordinate, baseline b (distance between cameras) and focal length f. Adapted from [32].

Finally, once the disparity is known, the distance from the camera to a target point can be computed using the following formula (10):

$$\mathrm{Z} = \frac{f \cdot d}{b} \tag{10}$$

where Z is the depth (distance to the point), f is the focal length of the camera, d is the disparity, and b is the baseline — the distance between the two cameras.

With the stereo vision part explained, we now complete the last major building block required fo this thesis. We now have both the machine learning component completed and the ability to estimate depth from images; the two foundational pillars that enable an autonomous system to detect traffic signs and compute how close or far they are. In the next section, we will move our attention to ROS2 which acts as a bridge between the theoretical concepts discussed so far and the practical application of making the excavator understand its surroundings.

2.9 ROS2

ROS2 short for "Robot Operating System" is an open source framework that allows different components of the robot to communicate with each other efficiently. It is not an operating system in the traditional meaning of the concept, but rather a man-in-the-middle that allows programs, called nodes, to talk to each other in a structured mode [24].

In this case, ROS2 play a major role in achieving our goal of traffic sign recognition and depth estimation. These tasks require a continous flow of input from the sensors (such as cameras) to be transmitted to a more powerful computing unit where it will be processed. This transmission of data is facilitated by ROS2 by making it clear how components can receive or send data.

ROS2 uses a subscriber-publisher way of communication based on topics. A topic is simply a communication channel on which data is exchanged. Nodes can publish data on a topic or subscribe to one in order to receive data published on that topic. For instance, the stereo camera node publishes frames on two different topics, one for the left camera and one for the right camera, while another node subscribes to these two topics to receive these images for processing. This is called separation of concerns and it is quite useful when it comes to debugging.

Each node in ROS2 is literally a program responsible for a specific task such as capturing images, performing traffic sign recognition or computing depth using stereo vision. By using topics, nodes can exchange messages independently but also collaborate in the same time. This is what makes ROS very powerful tools in robotics where multiple sensors and components interact with each other.

Overall, ROS2 provides all communication and collaboration capabilities for the prototype excavator used in this project. It ensures that the visual input captured by the cameras can reach to a computation unit where an object detection model can be , for instance, applied. This concludes the last prerequisite required for conducting this thesis. Now we will move our attention to finding a dataset that will further support our experiments.

3 Data

3.1 Choice of the dataset

The dataset used for this project is the Traffic Signs Detection Europe dataset[28] which can be found on Roboflow. This dataset was chosen because it contains a wide range of common, real-world traffic signs. This would make it suitable for training an object detection model like YOLOv8.

Other datasets publicly available online present only a limited number of traffic signs (e.g., 10-15 classes), which would make them less capable for a comprehensive detection [40]. In contrast, this dataset is already fully annotated, which saves significant time and effort. Many datasets that can be found online, are not annotated [27]. Therefore, they cannot be directly used for object detection tasks without additional preprocessing.

Moreover, this data was collected from real-world traffic scenes. This means that a model which would be trained on this dataset, will be exposed to very realistic scenarios, improving in this way its ability to make reliable predictions, an essential requirement for autonomous systems such as an excavator.

So after selecting the dataset that best fits the project's demands, it becomes necessary to understand its structure in details, such as number of classes, how annotations are formatted and what kind of traffic signs are represented.

3.2 Dataset Description

There are 54 classes in the dataset, each representing a different type of traffic sign. Examples of such classes are : "forbidden ahead", "forbidden left", "give way", "pedestrian crossing", "forbidden overtake", "info parking" and "bike lane". There are 4381 images with a resolution of 640x640, together with their corresponding labels. A label is a .txt file that enumerates all traffic signs appearing in the corresponding image on a different line. Each row encodes information in the following order: class of the detected object, x-coordinate of the center, y-coordinate of the center point, the width and the height to the border from the central point.

(a) Example image from the dataset containing four traffic signs: two “forb_ahead”, one “mand_ahead”, and one “prio_road”.

```
36 0.21953125 0.57890625 0.0921875 0.0765625
39 0.2359375 0.4796875 0.13125 0.12578125
0 0.1640625 0.51875 0.05234375 0.0765625
0 0.84375 0.6984375 0.02578125 0.0328125
```

(b) The corresponding label of the image above which is a .txt file containing separated on a new row the encoding of all traffic signs in the image above. Every traffic sign is encoded in the following way :`<class> <x_center> <y_center> <width> <height>`. For instance on the first row we can see a ”prio_road” (class 36) with its center at coordinates $(0.21, 0.57)$.

Figure 7: An example of image-label pair from the traffic sign dataset. Each label file describes the position and size of every sign detected in the image using normalized coordinates.

The imperative traffic signs to be recognized are the ones with mandatory priority. These are the round-shaped traffic signs such as ”forbidden ahead”, ”mandatory forward”, ”mandatory right” and so on.

In figures 7a and 7b, an example image from the dataset is shown together with its corresponding label file. The image contains four traffic signs : two instances of ”forbidden ahead”, one ”mandatory forward” and one ”priority road”. These signs are found in the label file, where each line corresponds to one detected object. Specifically, the first line represents the ”priority road” sign, the second line corresponds to ”mandatory forward”, and the third and fourth lines indicate the ”forbidden ahead” signs.

After understanding how the dataset looks and how structured it is, it is also essential to see how the data is distributed. The next section will present which signs appear more often than others and how the imbalance might influence model performance.

3.3 Data statistics

This dataset was collected from the real world, meaning that certain traffic signs have a higher frequency than others.

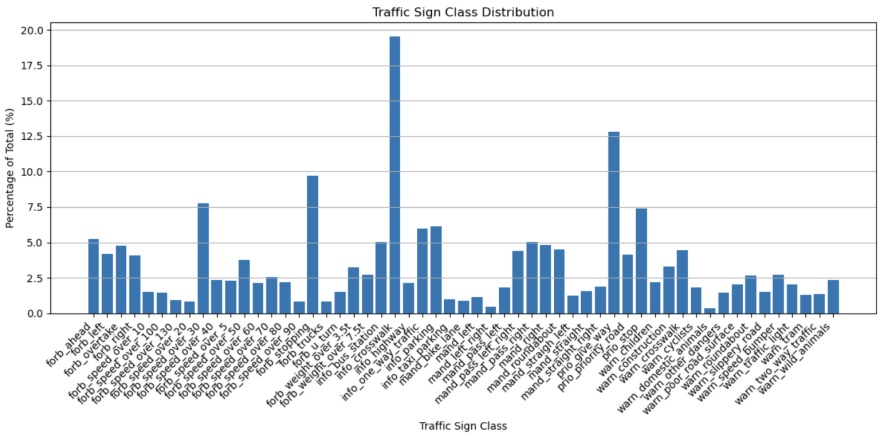


Figure 8: Bar chart that shows the distribution of classes within the dataset. Each bar represents the proportion of images relative to the total number of images of a certain traffic sign, with the sign "info_crosswalk" being the most common class.

From figure 8, it can be seen that the most frequent traffic sign is "info_crosswalk" being captured 19% of the time, while the least frequent traffic sign is "warn_domestic_animals" recorded less than 1% of the time. This means that "info_crosswalk" will have a high influence in the inference task, while "warn_domestic_animals" will have a significantly lower one. The mean number of occurances is 140 which represents 3,37% out of all 4381 train images.

Another metrics that are to be determined are the mean and standard deviation with respect to image pixel intensities. These are essential as they will indicate how balanced the red, green , blue(RGB) channels are distributed across images.

Metric	Red	Green	Blue
Mean	128.57207	128.67885	120.14463
Standard Deviation	59.646942	58.886738	66.398735

Table 2: Mean and standard deviation computed for each of the channels : red, green, and blue channel. For instance, the mean of the red channel is 128.57, whereas the standard deviation of the blue channel is 66.39.

In figure 2, we can see three values for mean, and also three for standard deviation. The scale for pixel intensities lies between 0 and 255. These values correspond to each channel, red, green and blue from left to right. The mean values vary between 120 and 130. This suggests that the pallette is balanced around the images without any color anomalies.

The standard deviation centers around the value of 60. This denotes that each color channel has moderate variability. This also shows that images across the dataset contain variations and contrasts, which is useful in object detection.

Having balanced color values in the dataset is absolutely needed because real-world scenes don't have strong color spikes. If the dataset were to present unusual color tones, the model may then learn patterns that can't be found in real life. This would lead to poor performance. Since our dataset exposes natural colors, then the model is likely to generalize well. Having this solid base, we can now move on preparing the data for training by handling class imbalance through splitting and augmentation.

3.4 Data Splitting

First thing that needs to be done is determining the frequency of every class regarding the total amount of photos. This is measured in percentages. Then, out of these frequencies, the classes whose frequencies appear less than 1% of the time would be selected for augmentation. This is done as they are represented by a relative low amount of instances which produces class imbalance. Despite the fact that YOLO employs weighting loss to punish the model when it classifies incorrectly an underrepresented class, by assigning to these classes higher weights in the classification loss [31], the model still has too few images and might lead to overfitting. So, a target to reach has been defined which is 1% of the total images of the dataset. The underrepresented classes would be augmented by an amount of images equal to the difference between the target and current number of instances for the class. This step is done in order for the model to avoid overfitting the underrepresented classes and to offer them a higher influence in the training phase, maintaining in the same time real world frequencies. The real word proportions of the dataset would have been affected if the classes had been overaugmented, making the model better at predicting the rare classes.

Before applying the splitting, the dataset was organized such that each class had its own folder in my file system containing only instances of that class. This structure allows a more controlled and reliable data split, ensuring that all classes would be properly represented across the training, validation, and test sets without the risk of underrepresentation. The dataset is then divided as follows: 70% of the data was allocated to the training set, 15% to the validation set, and the remaining 15% to the test set. This is a commonly used approach [33], as it provides a sufficient amount of data for training, hyperparameter tuning, and final evaluation, leading to well-founded conclusions about the model's performance. Importantly, this split is performed independently within each class folder, so that 70%, 15%, and 15% of the images from every class are consistently assigned to the training, validation, and test sets, respectively.

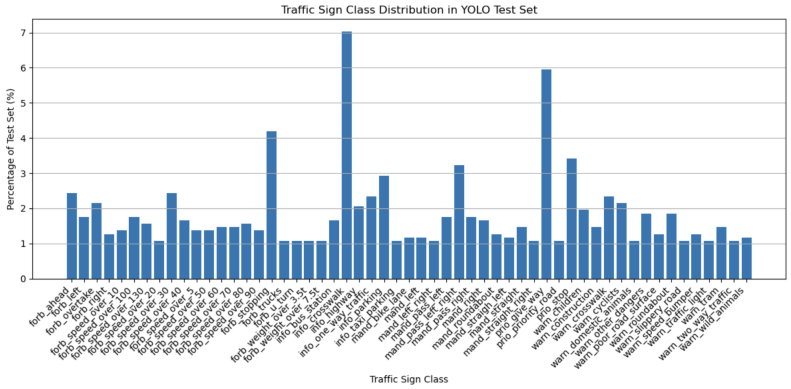


Figure 9: Bar chart that shows the distribution of classes within the test dataset. Each bar represents the proportion of images relative to the total number of images (within the test set) of a certain traffic sign, with the sign "info_crosswalk" being the most common class.

After this initial split, the test set was examined and found to be highly imbalanced, reflecting the original class distribution of the dataset. This can be seen in Figure 9, where the number of instances varies from class to class. So a balanced dataset needs to be created in order to achieve fair performance evaluation. A balanced evaluation set is crucial for ensuring that precision and recall are comparable across classes. Otherwise, it becomes difficult to determine whether one class performs worse than another, simply because it contains more or fewer instances. To achieve this balance, each class was limited to 11 images in the test set, the smallest number of images found across all classes, making sure in this way that each class contributes equally to the test set. This adjustment helps prevent evaluation bias, especially for metrics like precision and recall, which can be affected inverse proportionally by small class sizes. For example, if class A has fewer test instances than class B, then a single misclassification in class A would have a much larger negative impact on its recall or precision score than it would for class B. By assigning equal weight to each class in the test set, the evaluation becomes more meaningful and comparable. This approach ensures that the dataset is fully split without requiring further modifications. As a result of this process, the final dataset consists of 3,361 images in the training set, 915 images in the validation set, and 542 images in the balanced test set.

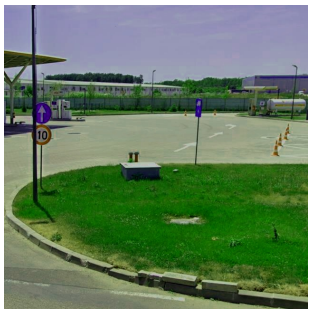
Now that the dataset is properly organized and also split across training, validation and testing sets, the next thing to be done is to give underrepresented classes a better chance during the training. In order to better represent the rare classes, augmentation will be applied. This is a process that increases variation(more diversity regarding image quality), while preserving real-world proportions. It will help the model generalize better without introducing any bias.

3.5 Data Augmentation

The augmentation is applied over the rare classes in order to provide them a higher importance in the training phase , without disrupting the real world proportions of the original dataset. It proved to be a successful solution in the study published by GE et al [11] which produced a 2% improvement in accuracy. Augmentation techniques that do not require the modification of the corresponding annotation file will be applied. The augmentation techniques that are to be applied are the following :

- **Contrast/brightness adjustment**
- **Gamma changes (light/dark)**
- **Slight motion blur**
- **Shift in hues**
- **Extra shadows added**
- **Slight blur**

Extra shadows added



Shift in hues

Slight motion blur

Slight blur

Gamma changes (light/-dark)

Brightness adjusted

We select the augmentation methods from above because they simulate real-world conditions that an autonomous system might encounter and also to give influence to rare classes. Adding extra shadows helps the model detect traffic signs when multiple shadows overlap and "hide" them, a common issue in outdoor sunny places. Adjustments in brightness and gamma are used for simulating night conditions which is crucial for excavators that operate during the night. Shifting hues introduces color variation that may be caused by different camera channels/sensors or overcast skies. Slight motion blur mimics the camera movements,an unstable camera or simply vehicle movement, whereas slight blur reduces visibility of the camera, just like during a rain when drops are on the camera. These augmentation will improve model's robustness without altering annotation files or affect real-world traffic sign proportions captured in the dataset.

With the dataset balanced, augmented and properly split, we conducted all necessary steps to ensure that our model is ready to be fed into an object detection model. The next section focuses on selecting a suitable model to be trained on this data, starting with the YOLO family.

4 YOLO family

4.1 Introducing YOLO family for object detection

To start with, an autonomous excavator needs to be able to quickly understand its surroundings, so recognizing traffic signs in real time is crucial. That's where YOLO, short for "You Only Look Once", comes in. YOLO is a family of object detection models known for being fast and efficient [35]. Instead of analyzing certain regions of images one by one like older model, YOLO is able to process an entire image in a single pass and instantly predicts the position of the detected objects. This makes it suitable for tasks where decisions need to be determined very fast, like, for example detecting a traffic sign on a busy construction site before it is too late.

One of the main reasons YOLO became so popular is because it is fast. Even earlier versions could run in real time, which is perfect for machines like excavators which have to make decisions on the spot. Unfortunately, that speed came with a big cost, accuracy. First versions had difficulties in identifying overlapping objects [36]. For example, YOLOv3 added features like residual connections and multi-scale detection. This made the detections of smaller traffic signs much better. So now, we manage to have them both, high precision and speed simultaneously [36].

Over the time, the research kept improving over YOLO; newer versions like YOLOv4 and YOLOv5 pushed things even further. Smarter techniques, like better feature extraction, starting to be integrated into these models, without slowing things down [3]. What is great for us is that YOLOv5 became very popular as it is easy to use and works well even with small datasets and is flexible enough to do well in different environments [18]. That makes it valuable for our case, as it will help our excavator detect traffic signs reliably even in unpredictable rough surroundings.

One of the main benefits of YOLO models is their versatility. They can be trained on both small datasets and large-scale datasets like COCO or Pascal VOC, while still being accurate [15]. Our traffic sign dataset has a significant size as mentioned earlier. This makes YOLO a great choice for our study case as it allows efficient training without requiring enormous computational resources, while still being able to deliver high performance in detecting traffic signs. Its ability to generalize well across varied conditions also helps our model perform reliably in realistic scenarios, such as those encountered by autonomous excavators in outdoor environments.

Another great model released in the market is YOLOv8. A great thing about is that it does not need to rely on predefined anchor boxes anymore. This may sound too technical, but here is what it means : older versions had to guess out of a set of fixed box shapes and sizes to detect objects [44]. This made training way more complex and less accurate, especially in the case when your dataset has objects like traffic signs that come in many different sizes and shapes. By removing anchors, YOLOv8 makes the training process simpler. It can better adapt to unusual objects such as traffic signs, which is perfect for our traffic sign datatset as some signs can look very different from others ("pedestrian crossing" looks completely different from "give way").

The YOLO family enjoys a lot of popularity among industry and research for its strong balance between speed, accuracy and easy of use. These qualities make it appropriate for our task of detecting traffic signs on autonomous excavators. YOLOv8, an anchor free architecture, is powerful in detecting signs of different shapes and sizes which makes it a

great fit for this project. Also, this will make the excavator more robust in construction sites, where traffic signs are more likely to be tilted or bent. Meanwhile, YOLOv5 stands out for its flexibility and easiness of training on datasets like ours. To compare their capabilities, we selected YOLOv8l, YOLOv8s and YOLOv5. The next section will explore each of this model in more detail.

4.2 YOLOv8

The architecture of YOLOv8 shown in Figure 10 is a good compromise between performance and efficiency, making it well suited for real-world applications such as autonomous excavators. As shown in figure 10, the backbone extracts features from the input image, while the head uses multi scale detection approach. This is convenient as it allows the model to detect both small and large traffic signs. Especially because the size of the traffic signs is also determined by the distance to the traffic signs. Then, the use of C2f modules and simplified path allow fast inference time, crucial for quick reactions. Meanwhile, multi-resolution feature fusion helps the model recognize overlapping or similar looking signs. All in all, this architecture supports robust and fast detection even under varying shapes or sizes encountered on unstructured terrain. This contributes to our goal of reliable traffic sign recognition. We opted to work with the "large" variant of YOLOv8 for its more sophisticated architecture that can understand subtle patterns out of our complex data set.

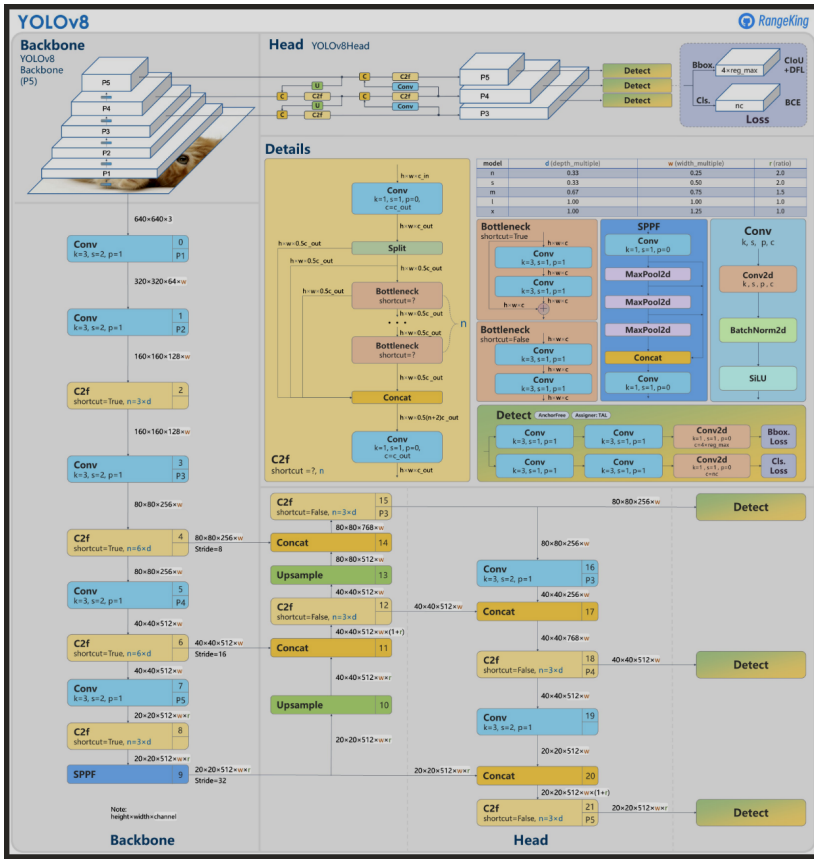


Figure 10: Internal architecture of YOLOv8. This includes backbone, neck and the detection head. The backbone is responsible for extracting multi-scale features using convolutional and C2f layers. The Neck (located between backbone and head) aggregates features via upsampling among the different resolutions (from large to small). Head will in the end perform object detection on the three feature levels (P3,P4, P5). Adapted from [45].

Yolov8l was chosen as mentioned earlier because it is one of the largest and most powerful variants of YOLOv8 family. This status is given by its sophisticated architecture. Since our dataset is quite complex, containing over 50 classes and thousands of images, using a larger model helps capturing the subtle visual differences between the traffic signs. YOLOv8 is capable of handling such complexity given its large size, fact that makes it the most promising candidate of achieving the highest accuracy among the models we train. On the other hand, YOLOv8s, the second model, chosen for this task stands on the other extreme, being one the lightest members of YOLOv8 branch. This makes it an ideal candidate to obtaining a low inference time.

So YOLOv8s, in contrast, was selected as it represents the lightweight member of YOLOv8 family. It is designed to run quickly, which makes it ideal for real-time use, especially when devices have a low computing power. The computing power of the excavator's processing unit, which is a MacBook Pro with M1 chip, is not that high. This makes it a possible model to consider for traffic sign detection. Even though, it is small, it still uses a smart architecture like the larger models, which makes it still perform well without being that heavy.

By comparing directly YOLOv8l and YOLOv8s, we are putting two very different models head to head: the biggest versus the lightest. This is more than just a simple technical comparison, it is a way to understand what really matters more for our goal. Do we want the excavator to be super accurate and detect every possible sign, or is it more beneficial that it reacts quickly? Analyzing both contradictory ends of the spectrum will help us find out which feature is more important for the thesis goal.

The next section will talk more about the third canditate, YOLOv5, which remarks itself as being easy to use and able to process variable size datasets, while maintaining high performance. This is important in our tasks as our dataset is quite complex, so YOLOv5 flexibility may be able to deal with this aspect.

4.3 YOLOv5

YOLOv5 is one of the most popular object detection models today. known for its high speed, reliability and easy to train capability. Its simple setup and flexibility make it a great choice for real world tasks like traffic sign detection on autonomous excavators.

YOLOv5 structure is shown in Figure 11. Architecturally, it consists of three main parts: the backbone, neck and head. The backbone is responsible for feature extraction from the input images. The neck refines these features across different scales using upsampling and merging. The localization and classification is performed by the head at three different resolution levels. The multi resolution feature is valuable for our scenario as traffic signs can be of multiple sizes and appear at different distances.

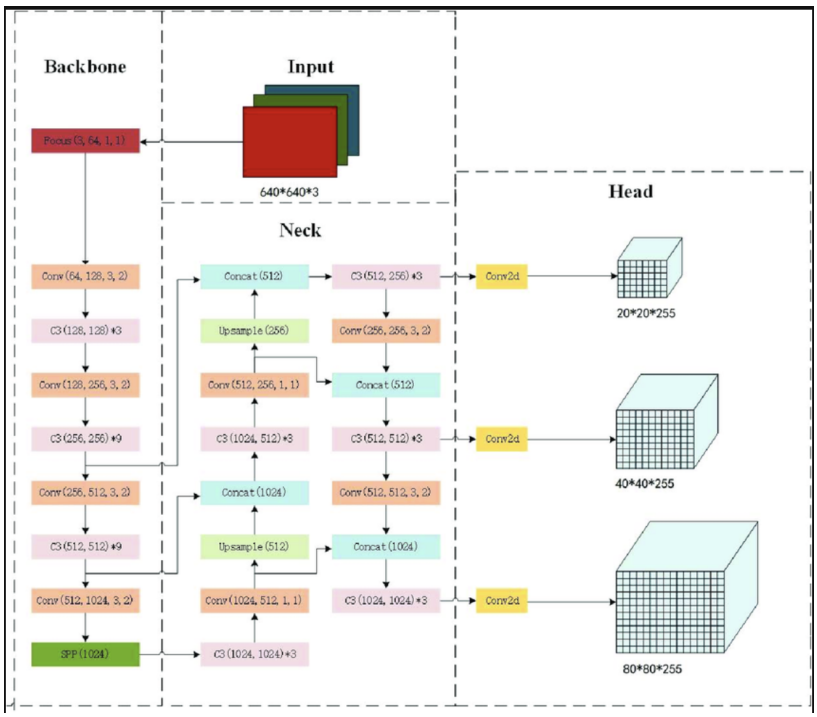


Figure 11: Architecture of YOLOv5 internal architecture. The image that is processed is of size $640 \times 640 \times 3$ is processed by three main components : backbone, neck and head. The backbone is used for feature extraction while the neck is used for aggregating multi-scale features using concatenation. In the end, the head will be used for object detection at three different resolutions (large, medium and small). Finally, the output will be the collection of bounding boxes together with their corresponding probabilities. Adapted from [18]

What also makes YOLOv5 attractive for this thesis is its flexibility. It can handle medium to large datasets like ours that contain over 4000 images and over 50 classes, without needing much fine-tuning or computing. At the same time, it would deliver stable results, which is critical when working with real-time detection machines like autonomous excavators.

In short, YOLOv5 represents a balanced, suitable choice for this thesis. YOLOv8l aims to achieve high accuracy, YOLOv8s pushes for a low inference time and in this spectrum, YOLOv5 sits in their middle. Including it in our comparison will help us better understand which trade-off, speed, complexity or simplicity represents the most sensible choice. It will act as a tiebreaker for YOLOv8l and YOLOv8s.

This concludes the review of all models that will be evaluated in this research. We have so far discussed about their advantages at a really high level, preparing the ground for the next section where we will compare their performance in a specific situation, by evaluating the performance on our chosen dataset.

5 Model Evaluation

Model Evaluation represents a major part of this project as it allows to judge the performance of our selected models and, then choose the safest to deploy on our excavator. It is critical to pick a safe model as a model that typically misclassifies traffic signs is very likely to behave unpredictably and make decisions that would damage the construction site. First, we need to mention that each model was trained on 50 epochs(amount of times the data is passed through the model) and on a batch size of 16(the amount of data model can process at once). The high number of epochs allows capturing vast patterns of our complex dataset, whereas the low batch size will introduce noise in the training process that would prevent overfitting the data. Now, we will go through the performance of each model individually and then compare each other's performance together.

5.1 YOLOv8l

This is the most complex model out of all three, and it was chosen for its large that could handle the complexity of our dataset. Its performance can be seen below :

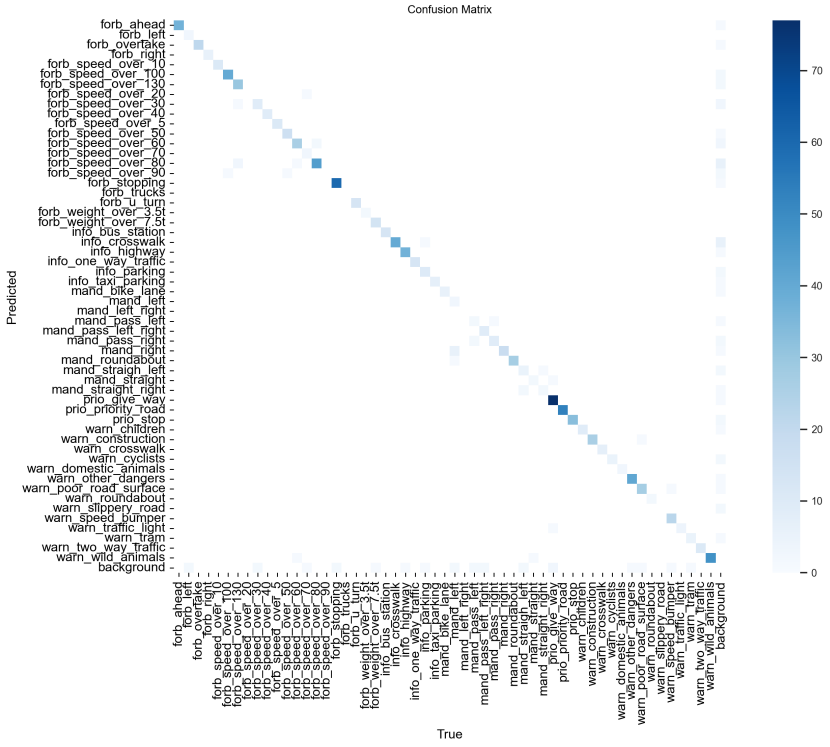


Figure 12: The confusion matrix of the YOLOv8l model regarding traffic sign classification. Each cell indicates how often true a class (vertical axis) was classified as a wrong one (horizontal axis). The elements on the main diagonal show the correct classification performed by the model.

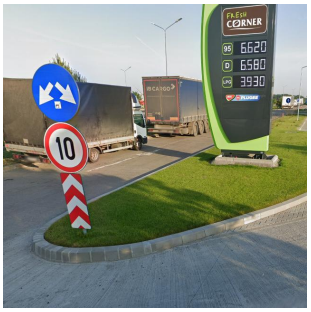
Class	Images	Instances	Box(P)	R	mAP50	mAP50-95
all	417	1002	**0.935**	**0.858**	**0.912**	0.817
forb_ahead	24	24	**1.000**	**0.954**	**0.992**	**0.865**
forb_overtake	16	17	**0.944**	**0.991**	**0.992**	**0.920**
prio_priority_road	14	14	**0.986**	**1.000**	**0.995**	**0.922**
forb_u_turn	12	12	**0.989**	0.833	**0.983**	**0.912**
mand_straight_right	13	13	0.667	0.692	0.793	0.755
mand_straigh_left	12	12	0.713	0.650	0.673	0.598
mand_pass_left	10	10	**0.966**	0.462	0.568	0.506
prio_stop	42	42	**0.910**	0.618	**0.931**	0.837
warn_slippery_road	11	11	0.627	**1.000**	0.714	0.676
info_crosswalk	61	69	**0.971**	**0.961**	**0.978**	0.841
warn_children	16	16	**0.981**	**0.938**	**0.988**	**0.960**
info_parking	20	26	**0.952**	0.808	**0.880**	0.712
mand_right	22	23	0.788	**1.000**	**0.892**	0.796
forb_speed_over_5	12	12	**1.000**	**0.995**	**0.995**	**0.863**
info_highway	17	17	**1.000**	0.657	0.725	0.610
warn_traffic_light	13	13	**0.955**	**1.000**	**0.995**	**0.907**
warn_other_dangers	21	21	**0.939**	0.735	0.827	0.773

Table 3: Detection sample showing the metrics obtained for a subset of traffic sign classes. The metrics are precision, recall, mean average precision and number of instance. For instance, for class "forb_ahead", the model managed to score 100% precision and 95% recall.

From the table 3 it can be seen that YOLOv8l managed to obtain the following overall performance : 93% precision, 85% recall and a mean average precision of 91%. These are quite high statistics show that our model has a good generalization capability. This contributes significantly to the goal of obtaining reliable traffic sign detection. We will start our analysis by having a look over the classes that achieve high scores.

When looking more in depth to the results, we can see that YOLOv8l performs pretty well on some of the most fundamental traffic signs for the excavator. For instance, it is able to detect signs with mandatory character like "forbidden ahead", "forbidden overtake", "priority road", "forbidden U turn" with very high precision and recall, often above 0.85. These signs are decisive for safe navigation, as they directly influence decisions like when to stop, whether or not to turn around or simply avoid certain directions. The fact that YOLOv8l handles these traffic signs well gives us confidence that the model understands and correctly identifies signs that matter most for our use case. Despite this, our model still registers some weaker performance among other traffic signs.

As mentioned earlier, our model still presents weaknesses among a certain group of classes. For instance, YOLOv8l struggles a bit with some mandatory direction signs like "mand_straight_right", "mand_straight_left " and "mand_pass_left" which can be seen in figure 13. These signs are similar looking, the only main difference being the direction of the arrow inside the blue circle. When the signs are captured from tricky angles the model can get confused. This is understandable, but still it remains a challenge, since these signs are also vital in guiding the movement of the excavator. So, improving the detection on these particular subtle differences could help the model make even smarter decisions. Though, not all classes are critical for traffic sign detection, as they are very unlikely to be encountered by the excavator.



(a) mand_pass_left_right and forb_speed_over_10

(b) forb_speed_over_20

(c) forb_speed_over_100

(d) mand_left

(e) mand_left_right

(f) mand_pass_left

(g) mand_straight_left

(h) mand_straight_right

(i) mand_pass_right

Figure 13: Examples of similar-looking mandatory and forbidden traffic signs that often cause confusion. First row shows the speed limit traffic signs(forb_speed_over_100,forb_speed_over_20, forb_speed_over_10) while the other 2 rows represent directional traffic signs with mandatory character.

One such example is the class "warn_domestic_animals" which had a recall 0 which means the model didn't detect any instances of this class correctly. However, this is not a major problem for our application. Signs like these are not usually found in construction areas as these are well separated from the farmland. While it is technically a significant miss, it will not affect the model's capability in terms of autonomous guidance, as it is very unlikely for an excavator to see such signs on-site.

Overall, YOLOv8l presents a strong performance in terms of detecting the most relevant traffic signs for autonomous navigation, especially the ones related to mandatory actions and related. Of course, there are still weaknesses with similar-looking traffic

signs, rare classes or less relevant ones, but the results are still promising. Now we will move our attention to an older model, but still very effective, YOLOv5.

5.2 YOLOv5

YOLOv5 was chosen for this task for its ease of use and its flexibility on processing different size datasets. The dataset used for this task as mentioned earlier is quite large, so YOLOv5 would be a suitable model, given its capabilities, to be trained on it. From table 4 it can be seen that YOLOv5 performs well, managing to score a precision of 83%, a recall of 85% and a mean average precision of 86%. The per class metrics, precision and recall are also quite high which suggests, as the case of YOLOv8l, a solid generalization capability. This is vital for the goal of the thesis to deliver reliable object detection. We will start our model performance analysis by looking at both its strengths and weaknesses.

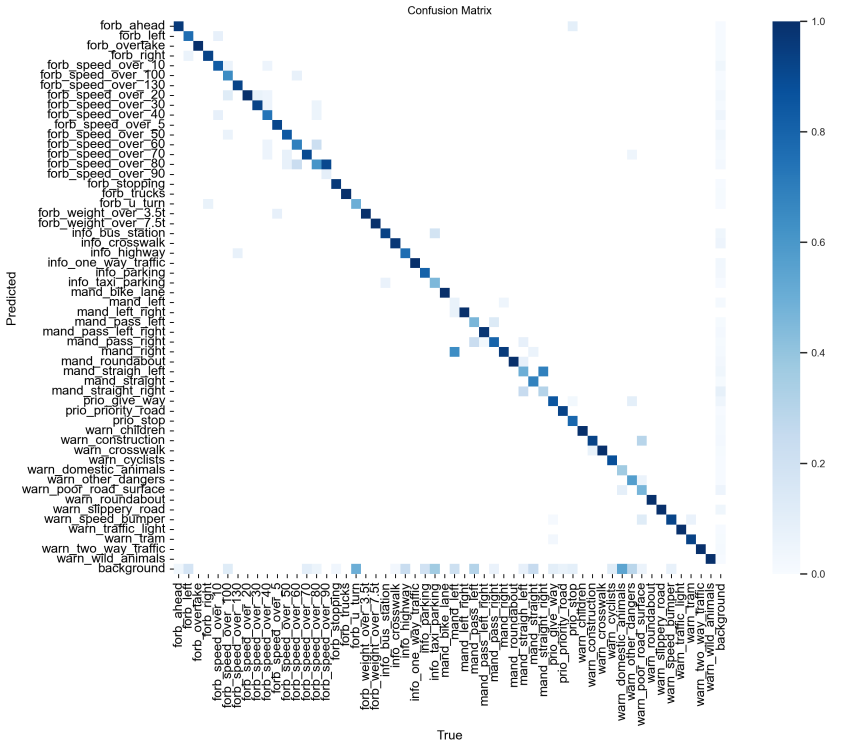


Figure 14: The confusion matrix of the YOLOv5 model regarding the traffic sign classification. Each cell indicates how often true a class (vertical axis) was classified as a wrong one (horizontal axis). The elements on the main diagonal show the correct classification performed by the model.

Class	Images	Instances	Box(P)	R	mAP50	mAP50-95
all	417	1002	0.83	**0.852**	**0.862**	0.752
forb_speed_over_10	12	12	0.607	**0.917**	**0.927**	0.836
forb_speed_over_100	17	17	**0.863**	0.647	0.731	0.643
forb_speed_over_130	13	13	**0.94**	**1**	**0.995**	**0.867**
forb_speed_over_20	11	11	0.616	**1**	**0.965**	**0.932**
forb_speed_over_30	27	27	**0.874**	**0.926**	**0.934**	0.807
forb_speed_over_40	19	19	0.752	**0.895**	**0.9**	0.811
forb_speed_over_90	11	11	**0.986**	0.0909	0.658	0.527
prio_stop	42	42	**0.977**	0.833	**0.915**	0.784
prio_priority_road	14	14	**1.000**	**0.916**	**0.931**	0.779
prio_give_way	68	93	**0.933**	**0.853**	**0.892**	0.751
forb_ahead	24	24	0.830	0.817	**0.958**	**0.882**
forb_right	14	14	**0.865**	**0.929**	**0.969**	0.785
mand_pass_left	13	13	0.687	0.462	0.651	0.545
mand_left_right	39	39	**0.924**	**1.000**	**0.993**	**0.864**
mand_straight	16	16	**0.900**	0.750	0.782	0.562
warn_domestic_animals	11	11	**1.000**	0.339	0.656	0.554
warn_other_dangers	21	21	**0.941**	0.571	0.584	0.529
warn_poor_road_surface	17	17	**0.880**	**0.865**	**0.891**	0.814
warn_traffic_light	13	13	**0.945**	**1.000**	**0.995**	**0.899**
warn_two_way_traffic	11	11	**0.854**	**1.000**	**0.972**	**0.883**
info_crosswalk	61	69	**0.917**	**0.961**	**0.981**	**0.827**
forb_trucks	11	11	**0.937**	**1.000**	**0.995**	**0.890**
warn_roundabout	14	14	**0.959**	**1**	**0.995**	0.834
mand_right	22	23	0.689	**1.000**	**0.939**	0.838
info_parking	20	26	**0.856**	0.808	0.821	0.579

Table 4: Detection sample showing the metrics obtained for a subset of traffic sign classes in the YOLOv5 model. The metrics are precision, recall, mean average precision, and number of instances. For instance, "info_parking" was detected at a precision of 85% and a recall of 80%.

Looking more closely to the results of the test phase (see table 4), YOLOv5 performs well on several critical traffic signs which are vital for autonomous navigation. For example, our YOLOv5 is able to identify "prio_stop", "prio_priority_road", "prio_give_way", "forb_ahead" and "forb_right" with high precision and recall. As mentioned earlier, these signs are indispensable for a safe excavator operation. This makes the excavator practical and good at decision making in a construction site.

However, this model has weaknesses when dealing with visually similar signs such as "forb_speed_over_x" group (see figure 13), like YOLOv8l. In contrast to YOLOv8l, YOLOv5 shows a lower performance among the speed limit group. This can be seen in the top left area of the YOLOv5 confusion matrix which presents hues of blue around the main diagonal. In case of the speed limit group, these signs differ only by the number displayed inside the red circle, making them hard to distinguish unless the image is sharp and clear. So slight blurs or different angle orientations can easily lead to misclassification. Directional mandatory signs such as "mand_pass_left", "mand_left_right" and "mand_straight" also cause problems. This happens likely because they share the same background, only the small arrow inside being able to make the difference between them.

Another challenge is given by the warning traffic signs such as "warn_domestic_animals", "warn_other_dangers" and "warn_poor_road_surface". These could be relevant in more complex, and structured construction sites. Not all these traffic signs are critical for an excavator but some may still appear in semi-organized areas or construction sites that are found on the public roads. Their lower scores indicate the need for better representation of this category.

That said, YOLOv5 shows encouraging results even for rare classes. For instance, "warn_traffic_light" and "warn_two_way_traffic" are detected with significant perfor-

mance, despite having relatively few instances in the training set. This shows a good class separability which contributes to the high generalization ability of the model. Overall, YOLOv5 provides a flexible solution especially when easy integration is a priority. In the next section , we will focus on the most lightweight architecture of this project, YOLOv8s, a model known for its high inference speed.

5.3 YOLOv8s

This model was chosen for its small size which allows it to be deployed on machines with limited computing power and also for its quick detection time when it comes to traffic sign detection. It is also our case as the processing unit that applies the model for inference is a Macbook Pro with M1 cpu. From table 5 it can be seen that the model achieves a precision of 78% , a recall of 80% and a mean average precision of 84%. These results are lower than the ones achieved by the two previous models. This is understandable for a lightweight model like YOLOv8s trained on a complex dataset like ours.

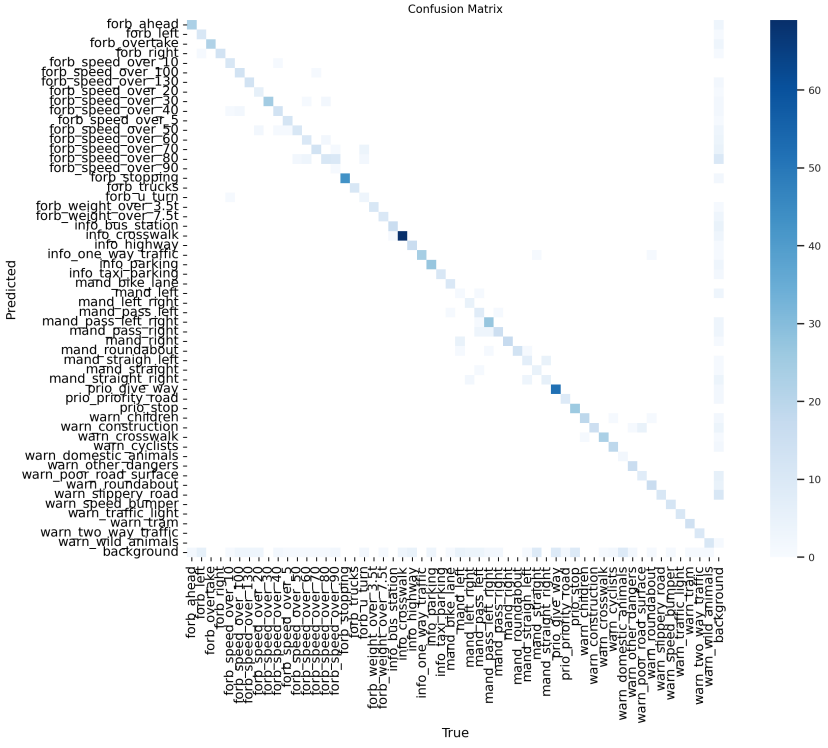


Figure 15: The confusion matrix of the YOLOv8s model regarding traffic sign classification. Each cell indicates how often true a class (vertical axis) was classified as a wrong one (horizontal axis). The elements on the main diagonal shows the correct classification performed by the model.

Class	Images	Instances	Box(P)	R	mAP50	mAP50-95
all	417	1002	0.789	0.807	0.846	0.745
forb_ahead	24	24	0.830	**0.958**	**0.969**	0.835
forb_left	17	17	**0.882**	0.706	0.786	0.755
forb_overtake	16	17	**0.860**	**1**	**0.940**	**0.880**
forb_speed_over_10	11	12	0.717	0.750	**0.851**	0.787
mand_pass_left	10	13	0.727	0.412	0.578	0.491
mand_left_right	11	11	0.727	0.492	**0.873**	0.736
mand_pass_right	14	13	0.657	**0.864**	0.711	0.648
mand_straigh_left	12	12	0.421	0.366	0.519	0.469
prio_stop	42	42	**0.947**	**0.857**	**0.952**	0.836
warn_tram	14	15	**0.894**	**1.000**	**0.995**	**0.902**
warn_two_way_traffic	11	11	**0.943**	**1.000**	**0.982**	**0.893**
warn_speed_bumper	13	13	0.845	**0.923**	**0.926**	**0.884**

Table 5: Detection sample showing the metrics obtained for each traffic sign class. The metrics are precision, recall, mean average precision and number of instances and provide insights about the performance of YOLOv8s.

To begin with, this model performs poorer than YOLOv5 on mandatory actions classes. For instance, it manages to score a precision of 71% and recall of 75% on "forb_speed_over_10". This is not encouraging as the excavators can operate in rough, dangerous terrains where speed needs to be kept at minimum in order to prevent accidents. Similar to the above models, YOLOv8s also struggles with certain traffic signs that look very similar to each other such as "mand_pass_left", "mand_left_right", "mand_pass_right" and "mand_straight_left" (see figure 13). This is given by the low precision scores that span in the range [0.65, 0.77] .But this is not entirely surprising because YOLOv8s was designed for speed, so it does not have the capacity to understand small visual differences, like the orientation of an arrow inside a circle in the case of direction traffic signs as the ones mentioned earlier. These subtle differences can be hard to learn even for stronger models.

Despite its failures, this model has decent performance when it comes to detecting mandatory traffic signs such as "forb_ahead", "forb_left" and "forb_overtake" and "prio_stop". This is beneficial for enabling safe navigation in the construction site. For instance, it manages to score 83% precision and 95% accuracy for the "forb_ahead" class. It even manages to obtain good performance like YOLOv5 on less populated and less relevant classes for an autonomous excavator. For example, it scores high on classes like "warn_tram", "warn_two_way_traffic" and "warn_speed_bumber", the precision being usually over 0.82. As mentioned in previous subsection, this shows solid class separability which is critical for safe navigation.

In short, YOLOv8s shows us the benefits and the limitations of using a lightweight model. It is efficient and manages to detect many traffic signs quite well as it can be even seen in the blue-based main diagonal of the confusion matrix. However, its lower accuracy on similar looking traffic signs indicates that subtle patterns are sacrificed in favor of the speed, which is the main strength of this model. In order to understand which model is the best fit for our project, we will have to compare the overall performance of all three models side by side.

5.4 Model selection for deployment

One observation that can be seen clearly when comparing all the models side by side is the trade-off between performance and speed. So far, YOLOv8l delivers the best performance with a precision of 93%, recall of 85% and mean average precision of 91%. YOLOv5 follows with also solid results (83% precision, 85% recall and 86% mean

average precision), whereas YOLOv8s obtains the lowest scores out of all 3 models (78% precision, 80% recall and 84% mean average precision). As far as inference speed is concerned, YOLOv8s is the fastest at just 365 ms per image, YOLOv5 takes 938 ms per image and YOLOv8l takes the longest time for detection , 1143 ms per image.

Despite this, YOLOv8l is chosen as the most suitable model for our thesis. Its capability to accurately detect the most vital traffic signs makes it a great choice for reliable sign detection, which is the goal of this thesis. Even though, this is slower than the other two models, the compromise is acceptable for the reason that excavators move at very low speeds (around 3-6 km/h) [5] which means they don't require instant response time like fast-moving cars on a highway. In that kind of setting, a few milliseconds of delay can matter a lot. But, in the case of an excavator, achieving reliable and accurate sign detection is more valuable than reacting few hundred milliseconds faster. Therefore, the extra time required by YOLOv8l is a fair compromise for the higher precision it provides, and thus safety. In the next section, we will look into the excavator prototype.

6 Excavator Setup

6.1 Mechanical Setup

In order to achieve the goal of the thesis, a prototype robotic excavator is used. It is built as a conventional excavator consisting of : two arms, cup, cabin and wheels. It consists of four motors: two motors for both sets of wheels, one motor that operates the cabin and one motor that operates the arms. In terms of dimensions, the robot is 40 cm long, including the span of the arm, 15 cm wide and 31 cm tall. The LiDAR sits at the top of this height and it is used for scanning the surroundings, as it will be seen in the next subsection.

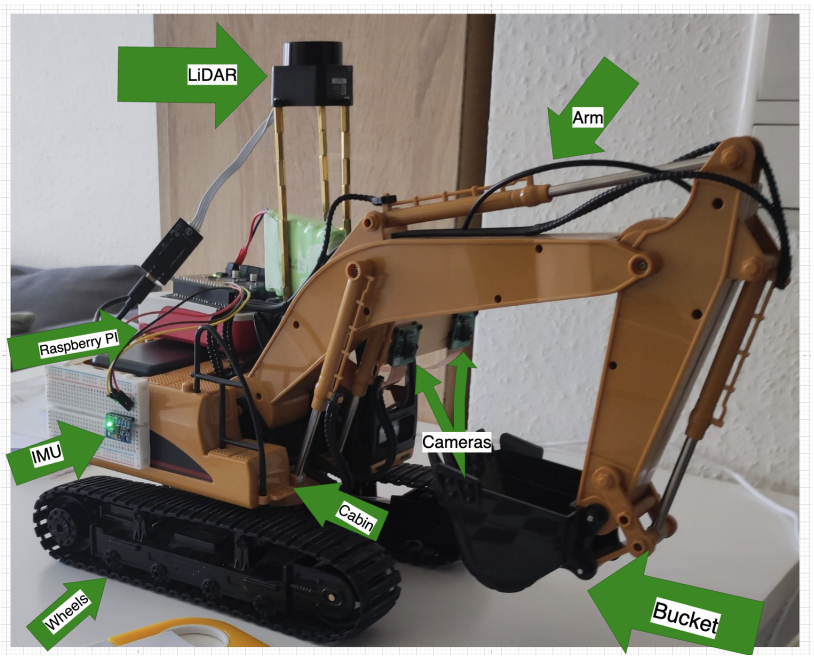


Figure 16: Excavator prototype including components such as the ones marked : bucket, IMU, LiDAR, Raspberry Pi, Arm , wheels and so on. The traffic sign recognition and depth estimation pipeline will be deployed on this machine.

6.2 Sensor suite

In order to process information about the environment, the robot makes use of sensors. The sensors and input devices installed on the robot are :

- **Stereo camera system**: Two Raspberry Pi cameras aligned at the same height which capture images for depth estimation by computing the disparity between corresponding pixels
- **LiDAR**: The LD19 sensor scans the surroundings with lasers to build a 2D map and detect obstacles.
- **IMU**: Sensor that tracks robot's movement and orientation in real time

These sensors provide the raw input needed for depth estimation, obstacle detection, and localization. In the next section, we explain how this information is processed and used to create an autonomous excavator.

6.3 Computation and Data Processing

The two cameras are aligned at the same height. This enables us to perform stereo vision for our depth estimation goal. The cameras will be used to capture frames from the environment, which then would be processed by the YOLOv8l model in order to perform traffic sign detection. The LiDAR is used to scan the environment by sending laser scans in all directions. This helps the model understand its surroundings by creating a point-based map which is useful for applications such as obstacle avoidance. In the end, the IMU (Inertia Momentum Unit) is a device used for real time location tracking. The fusion node that combines data from both IMU and LiDAR ,allowing navigation and localization, is fundamental for autonomous operation. All the input information captured by the sensors needs to be processed by a computing power, which acts like the "brain" of the excavator. This processing unit role is ensured by a Raspberry PI 2 with an Ubuntu 24.04 as operating system. This PI will be responsible for distributing the data from the sensors to my MacBook Pro with M1 cpu for further analysis and processing. We will use ROS2 Jazzy for the data flow between components and the processing unit. In the next section, we will focus on the system that allows the information to be transmitted from sensors to the device where it will be further used.

7 Methodology

7.1 Overview

The main goal of this methodology is to create a system capable of recognizing traffic signs and determining how far away they are. In order to do this, we use two Raspberry Pi cameras that capture images in the same time. Then, YOLOv8l model is applied over both images in order to extract the traffic signs. Then for every traffic sign detected in both frames we calculate how far away it is using stereo vision. We will look at the same traffic sign from 2 different angles, given by the perspectives of the 2 cameras deployed. So, our approach combines both object detection and depth estimation into a single working unit.

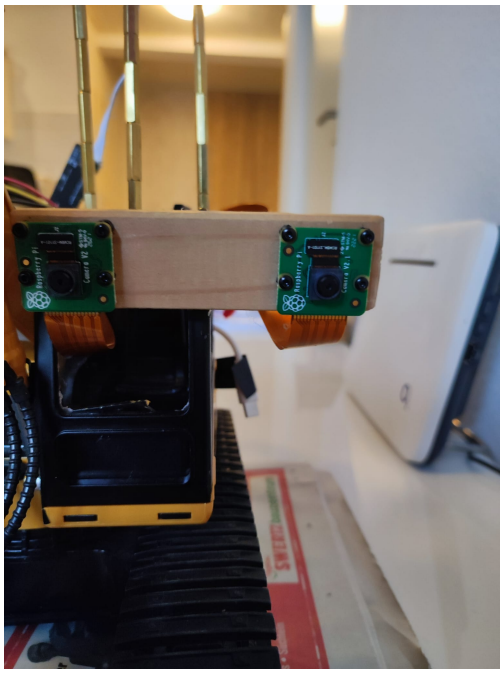


Figure 17: Two Raspberry Pi v2.1 cameras aligned at the same height for stereo vision, used for capturing images in real time. Object detection will be performed on each frame individually.

Figure 18: The 2 cameras, left and right, up and running, capturing images in real time. The top image is input captured by the left frame, while bottom frame is input captured by the right frame.

7.2 Frame capturing node

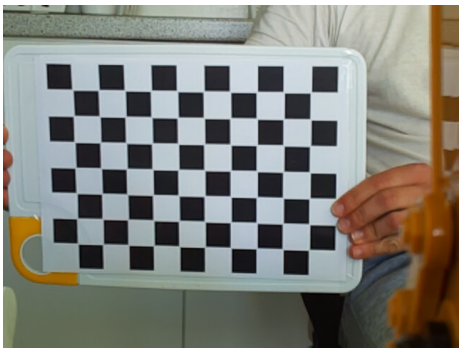
To start capturing images, we will install two Raspberry Pi v2.1 cameras(see Figure 17) at the same height; necessary condition for performing stereo vision. These cameras will play the role of the robot's "eyes", capturing synchronized views of the environment. So, a ROS2 node was developed to start both left and right cameras and record the frames captured by them in a synchronized way. The images are transmitted at a rate of 30 frames per second at a resolution of 640x640 pixels over the network.

The left camera transmits image information on a topic called "left_camera\image_raw", while the right camera publishes images on the topic called "right_camera\image_raw". We use different topics to ensure concern separability such that tasks like recording and applying object detection models can be performed in different nodes. This isolation is effective in debugging and in making the application more scalable. It also provides flexibility as the nodes can subscribe just to the data they need, without affecting other nodes. Now, having the cameras up and running and able to transmit data on their corresponding topics over the network(see figure 17), we can shift our attention to obtaining the required parameters for depth estimation via calibration.

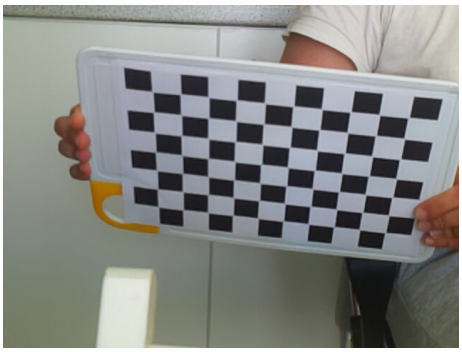
7.3 Camera calibration

In order to compute accurately the distances to the traffic signs, the cameras need to be first calibrated. For this task we need 15 pairs of images(rough estimate for which obtained parameters were precise) capturing a checkerboard in multiple angles and positions. By pair it is meant that the frames are captured in the same time by the left and right camera. The checkerboard is held at different angles, distances and positions within the image in order to cover a wide range of views. This can be seen in Figure 19 showing a small sample of the checkerboard images used for calibration. The diversity in the views is essential because it helps the calibration process find the correct matches between the same corners of the checkerboard in both images.

(a) Left Camera - Image 1



(b) Right Camera - Image 1



(c) Left Camera - Image 2

(d) Right Camera - Image 2

Figure 19: Example of stereo pairs of checkerboard images used during the calibration process. Each row shows a pair of images captured simultaneously by the left and right cameras. These pairs will be provided as input in the calibration script, which provides the intrinsic and extrinsic parameters for depth estimation. Corner correspondencies between pictures are used to compute these parameters.

The system was able to determine the internal parameters of the cameras such as the calibration matrixes corresponding to each camera using OpenCV's built in tools. The calibration matrix contains parameters like focal length and principal point. It also determined how both cameras are physically positioned relative to each other by computing the rotation matrix R and translation vector T, which represent the extrinsic parameters. Also, the calibration process managed to determine the Essential matrix (E) and Fundamental matrix (F), which describe the geometric relationships between the two views. Besides these parameters, distortion maps for each camera were computed

and these check how curved real world straight lines appear in the frame due to the lens used by cameras. Now, having all necessary parameters for depth estimation, it is also imperative to find how accurate they are, as we will see in the next subsection.

7.4 Calibration accuracy

To check how precise the calibration process was, we will use the distance between the cameras, also known as baseline, as evaluation metric. We will compare the baseline measured in real conditions with the one resulted from the calibration process. This is done using the translation vector T of size 3x1. We will use the formula (11) written below :

$$b = \sqrt{t_x^2 + t_y^2 + t_z^2} \tag{11}$$

where t_x, t_y and t_z are the elements of the translation vector.

By applying it, we obtain a computed baseline of 6.9 cm. The physically measured baseline is 7.1 cm, resulting in a slight error of about 2 mm in the calibration process. This small differences confirms that the calibration was very accurate and both cameras position well relative to each other, which is a strict requirement for a precise depth estimation. Having accurate parameters allows us to rectify both frames reliably as it will be seen later.

7.5 Image rectification

Once the cameras are fully calibrated, the next thing we need to do is image rectification. This process adjusts the left and right images so that all corresponding pixels from the left and right frames lie on the same horizontal line, called epipolar line. This alignment is crucial because it simplifies the disparity computation, the shift on the x-coordinate of the matching points, which is directly used in estimating the depth.

In our setup, the rectification is realized inside the ROS2 node called "depth_node". This node subscribes to both "left_camera\image_raw" and "right_camera\image_raw" topics. and it is responsible for image rectification, object detection performance and depth estimation as it will be seen later.

In practice, we do this using the cv2.initUndistortRectifyMap function from OpenCV library. This function is used for removing distortion and assigning each pixel with the position of another pixel in the original frame where to get its value from.

These pixel associations are stored in two maps for each camera, one for the x-direction and the other for the y-direction. These maps play the role of a look-up table and for each pixel in the final image it assigns the value of its corresponding pixel from the original image. Once having these maps for both coordinates, we simply combine them and obtain for each pixel the final image coordinates (i,j). The value in the final image will be taken from the previously computed coordinates of the original distorted image. This gives us two perfectly aligned images where features from both cameras are matched both horizontally and vertically, as it can be seen in the Figure 20.It can be seen that the right image is slightly cropped in the bottom right corner(it does not cover the entire window) and the left image is cropped and stretched to the right such that all corresponding points will lie on the same epipolar line. So, with the rectified images,

we move on to detecting traffic signs using both cameras and computing the distance to them via stereo vision.

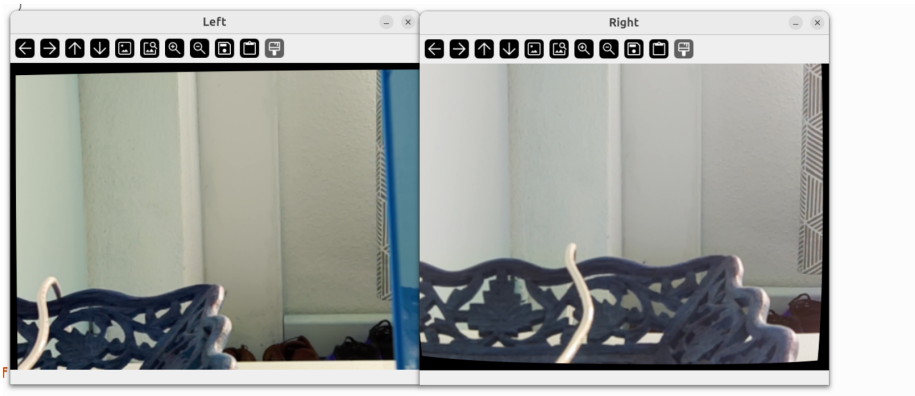


Figure 20: Rectified left and right images. The left and right images have been rectified so that all corresponding pixels lie on the same epipolar line, which is crucial for a correct depth estimation.

7.6 YOLOv8l Traffic sign detection

After rectifying the images from both cameras, we apply the YOLOv8l model individually on both left and right frame. The model will extract the traffic signs in each image and will return the bounding boxes around these identified signs. At this point, we have two sets of traffic signs, one for each camera.

Next, we will determine the common signs that appear in both frames. This is necessary because depth estimation can only be computed if a traffic sign is captured in both cameras. For each common traffic sign, we calculate the center point of the bounding boxes in both left and right images. These center points will serve as target points for estimating the depth.

Since the images are rectified, the y-coordinate of these center points will stay the same, while the x-coordinate will be shifted left or right. So we will focus on the horizontal position, given by the x coordinate and we will calculate the disparity using the formula (12) :

$$d = |x_{left} - x_{right}| \tag{12}$$

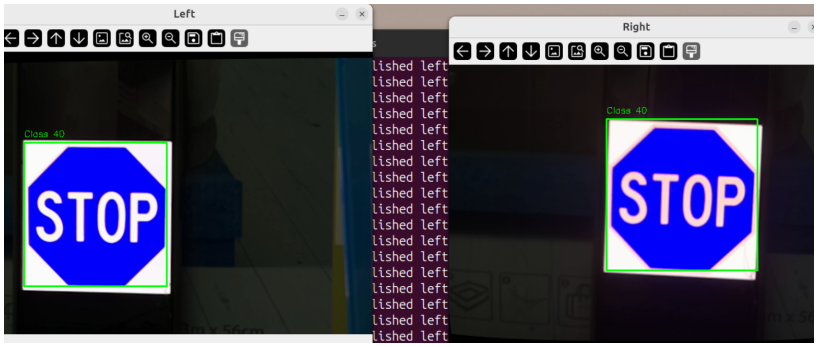
where d is the disparity, x_{left} and x_{right} are the x -coordinates of the center of the detected traffic sign in both frames.

The disparity basically tells us how shifted the target point(the center of the common traffic sign detected) between the two images is. Finally, we use the stereo depth formula (13) :

$$\text{depth} = \frac{f \cdot b}{d} \tag{13}$$

where f is the focal length of our cameras(intrinsic parameter) and b is the baseline, the distance between the two cameras which in our case is 7.1 cm. This formula gives us the distance from the robot to the detected traffic sign, allowing in this way navigation

in unorganized environments such as construction sites. This can be seen in Figure 21 where the two cameras capture a "stop" traffic sign and estimates the depth to 40 cm. It is a demonstration of both stereo vision and object detection working together. This concludes the methodology section. We have managed to build an approach that is both able to recognize traffic signs and compute depth using machine learning and stereo vision concepts. In the next section, we will conduct experiments in order to evaluate the usability of our approach.



(a) Left and right camera detection frames. Both left and right cameras capture and recognize the same traffic sign, "prio_stop" (class 40), that sits in front of the excavator. In both frames, the detection box is placed over the detected traffic sign which means that model was able to recognize it successfully.

```
0: 480x640 1 prio_stop, 1612.2ms
Speed: 0.6ms preprocess, 1612.2ms inference, 1.6ms postprocess per image at shape (1, 3, 480, 640)
[INFO] [1750432906.625605073] [depth_detector]: Left signs: [(40, 167, 276)]
[INFO] [1750432906.625949724] [depth_detector]: Right signs: [(40, 397, 221)]
[INFO] [1750432906.626245038] [depth_detector]: Common signs found: [(40, 167, 397)]
[INFO] [1750432906.626818163] [depth_detector]: [Depth Estimation]: Class 40 at ~0.40 m
[INFO] [1750432925.389106941] [depth_detector]: Stereo callback is triggered
```

(b) Object detection performed and depth estimation computed. In the image we can see the traffic signs captured in left and right frames individually. A detection is formulated in the following format : { <class> <horizontal coordinate> <vertical coordinate> }. There is a set of traffic signs captured by the right camera, one such set for the left camera and a set that is the result of the union of the first two sets mentioned.For instance, the "stop" traffic sign identified in both left and right images(with small shifts in the position coordinates) is estimated to be 40 cm away from the cameras.

Figure 21: Left and right cameras both detect the same traffic sign, "prio_stop", using the YOLOv8l model. The system manages to identify this traffic sign in both frames, matches the detections, and estimates the distance to be around 40 cm. This confirms that both traffic sign detection and depth estimation work correctly.

8 Experiments and interpretation of results

8.1 Running the experiments

In order to check the accuracy of the depth estimation, we will run a series of experiments by placing a traffic sign at multiple distances relative to the camera. To keep the experiment simple, we decided to perform the object detection over a single traffic sign such that the focus falls on estimating how far the traffic sign is, not on traffic sign recognition which has already been evaluated. For this experiment, I will use my phone for displaying the traffic sign and a box as support for it. The traffic sign used for these

tasks is "warn_crosswalk" (see Figure 22), but using other class that model performs well on works just fine.

Figure 22: Side-by-side visualizations in left and right camera of the "warn_crosswalk" traffic sign at different distances. The upper images show the detected traffic sign at a lower distance, while the bottom ones recognize the same traffic sign("warn_crosswalk") at a higher distance.

We start by placing the traffic sign at certain distances relative to the camera, in the following order : 31 cm, 33 cm, 47 cm, 61 cm, 87 cm , 130 cm and 150 cm. This range of distances allow us to check the accuracy of depth estimation on short distances (31-47 cm), medium distances (61 and 87 cm) and long distances (130 cm and 150 cm). This will tell us in which scenario our method performs the best. For instance, on construction sites some signs are sporadic and can be placed at long distances away from each other. So, it is crucial to know whether the system performs better at close or long range.

So, once agreed on how the experiment would be conducted, we launch the ROS2 nodes, the publisher which in our case is the frame capturing node together with the subscriber which is responsible for object detection and depth estimation. After experimenting with every distance mentioned earlier, we obtain the following results :

Computed Distance (cm)	Real Distance (cm)
35	31
39	33
58	47
76	61
130	87
220	130
280	150

Table 6: Table illustrating the comparison between the computed and real distances during the depth estimation. The first column shows the distance computed by the stereo vision system, while the second column represents the corresponding distance measured in real world conditions.

These measurements show clearly how well our system is able to compute distances under multiple conditions. In the next section, we will analyze these results in order to better understand the strengths and weaknesses of our method.

8.2 Interpretation of results

Looking at the obtained results, we can clearly observe that the system performs the best on short distances. For example, in the first row of the table, the real distance is 31 cm while the computed distance is 35 cm, resulting in a 4 cm error, which is acceptable. The other computed short distance from the second row is only 6 cm overestimated relative to its real corresponding distance of 33 cm. However, as the real distance increases, the precision decreases significantly. For instance, at 150 cm distance, the system estimates the depth to 280 cm, which results in an error of 130 cm which is almost as long as the real distance. This clearly suggests that our method is mostly reliable for close-range depth estimation.

To understand why this happens, we need to recall the depth formula (13):

So, as an object moves farther and farther away, the disparity values become smaller and smaller. In fact, if the disparity tends to 0, then the computed depth approaches infinity as it can be seen in the following limit (14) :

$$\lim_{\mathrm{d}\to 0} \frac{f \cdot b}{\mathrm{d}} = \infty \tag{14}$$

Disparity is based on the shifts on the horizontal coordinate which is directly influence by the resolution of the camera. Our system uses 640x480 resolution cameras. This is a low resolution compared to nowadays standards, like 1080p or higher. Having fewer pixels leads to a lower precision in computing small disparities, especially at long distances. So, a small error in estimating the disparity can cause huge errors in the depth formula.

Another factor which influences the depth estimation is given by the accuracy of the parameters obtained from the calibration process. Fortunately, our calibration process was highly accurate, with a slight error in the computed baseline of about 2 mm. However, our calibration is not perfect and even small imperfections in the obtained parameters can still have a negative affect, especially at long ranges.

Finally, an influence in the depth estimation is also given by placement of the bounding boxes by the object detection model. While our model performs well, there are still moments when bounding boxes are placed very inaccurately over the traffic signs. So, if the center of the detected object is misplaced, the disparity and therefore the depth are imprecise. Despite the fact that this was not a major problem in our experiments, it is still something that should be considered in future reproductions of this experiment. This concludes our research, at the end of which we managed to create an approach based on machine learning and stereo vision that allows an excavator to recognize traffic signs and compute the distance to them. This is required in order to understand and navigate the construction sites where it operates.

9 Conclusion

9.1 Summary

This thesis had in mind the possibility of combining stereo vision and machine learning into one single functional system that would be capable of recognizing the traffic signs and estimating the distance to them using two Raspberry Pi cameras. The motivation comes from real world robotic applications like autonomous excavators in which recognizing traffic signs and computing depth are necessary for a safe navigation. By combining an object detection model and stereo vision concepts, we managed to create a working product that can be run in real time and deployed on lightweight hardware.

First of all, a suitable object detection model for this task had to be chosen. So, three versions of the YOLO object detection family were analyzed : YOLOv8s, YOLOv5 and YOLOv8l. YOLOv8s was chosen for its high inference speed and lightweight architecture while still being able to achieve good performance, whereas YOLOv8l was chosen for its large size capable of learning even the subtle patterns of our complex 54 classes dataset. In this comparison, YOLOv5 acts like a middle ground in the size-speed trade-off maintained by both YOLOv8l and YOLOv5. Out of these selected models, YOLOv8l has shown the highest performance, followed by YOLOv5 and then YOLOv8s. The performance differences span in the range (0.78, 0.93) for precision and (0.80, 0.85) for recall. These differences in metrics are mainly caused by the different sizes of the models considered. It can be seen clearly that the larger models, like YOLOv8l, present better metrics than lightweight models, like YOLOv8s. The difference produced by YOLOv8l with respect to YOLOv8s is 0.15 which is quite significant and attributed mostly to the more complex architecture of YOLOv8l. Even though, all the models have obtained impressive performance, they still exhibited lower evaluation metrics for certain categories of traffic signs like the speed limit ones given by "forb_speed_over_x" classes and also on similarly-looking classes such as "mand_right", "mand_pass_left", "mand_pass_left_right". This can be a significant issue on very complex construction sites where redirection using traffic signs like "mand_pass_left", "mand_pass_left_right" is required for safe navigation. Another factor that influenced the choice of the model for this task is the inference speed. In this case, YOLOv8l had the highest inference speed while YOLOv5 and YOLOv8s came next. In our case, as excavators move at a low speed, they don't require almost instant reaction time like in the case of a car traveling on the motorway at high speed. That's why, for our application, we considered the model performance more valuable than inference time as any wrong decision on a construction site can lead to material damage. So, the model chosen for this object detection and depth estimation task was YOLOv8l. This is the first pillar for obtaining a functional pipeline able to recognize traffic signs and calculate how far away they are.

Further, in order to construct this, a stereo vision and machine learning based method was created and integrated in the ROS2 framework. The system captured synchronized pairs of images using two Raspberry Pi cameras, calibrated them using a checkerboard and OpenCV functions, and, then rectified them in order to prepare them for the depth estimation step. YOLOv8l was applied on both frames separately to detect the traffic signs. For the traffic signs that were captured in both frames, the corresponding center point of the bounding box was determined in both images. This pair of center coordinates was then used to determine the disparity, by computing the shift along the horizontal direction. Disparity was used together with the baseline and the focal length (determined from the calibration process) to estimate the depth to the detected object.

This system was tested across a wide range of distances ranging between 31 cm and 150 cm. The same traffic signs were used within the experiments for consistency. The results of the conducted experiments show that the system is mostly accurate on short distance, even though there are still errors within a few centimeters. For example, at a physically measured distance of 31 cm , the computed depth is 35 cm resulting in an acceptable error of 4 cm. Unfortunately, the accuracy decreases a lot as the real distance increases. An example showing this would be that at 150 cm of distance, the generated error was 130 cm, which is almost equal to the real distance itself. This indicates that our setup struggles significantly on long range estimations.

The main reason behind the errors mentioned before lies in the computation of the disparity. The farther away you move from the camera, the smaller the disparity gets. From the depth formula $depth = \frac{f \cdot b}{disparity}$, we can see that when disparity tends to 0, the depth tends towards infinity. This indicates that even small errors in disparity can generate extremely inaccurate depth estimations. This situation is also facilitated by the low resolution cameras that have been used in this project. Low resolutions limit the precision in pixel operations, so pixel level information on long distances is lost. The disparity computation can also be negatively influenced by the quality of the calibration process and the model bounding box precision. These were precisely computed in our thesis, but still not perfect. For experiment reproduction, the quality of these parameters should be strongly considered.

Despite all of these constraints, the system shows that our approach which combines stereo vision with object detection performs pretty well on short range distances. This is promising in environments where traffic signs appear close to each other. In addition, the lightweight hardware and open source tools make our approach to be easily deployed on excavators in the real world.

9.2 Future work

For future work, several improvements can be investigated. For instance, higher resolution cameras would provide a higher precision when performing pixel based operation such as disparity computation. It could also be investigated whether a more complex model like YOLOv8x would perform even better on the chosen dataset, and thus, place the bounding boxes more precisely over the detected traffic signs. This would also make the disparity more precise and, therefore the depth estimation as well. Moreover, this approach could be improved even further by combining it with information captured by the LiDAR and IMU in order to bring it closer to a fully autonomous system.

In conclusion, this thesis manages to create successfully a solution that is able to recognize traffic signs, estimate depth and could be easily deployed on heavy machines like excavators due to its lightweight nature. While it performs the best at short ranges, it still represents a solid solution on top of which more advanced and scalable robotic systems could be built.

10 Appendix

This section is meant for the readers who want to reproduce the experiment conducted in this thesis. It also complements the arguments and ideas already presented in the thesis and it includes the specific tools and setup used in this project in details.

10.1 Dataset classes

The dataset used in this thesis contains 55 classes, each corresponding to a specific traffic sign. All the classes and their own id can be seen below in Table 7.

Class	Name	Class	Name
0	forb_ahead	28	mand_left
1	forb_left	29	mand_left_right
2	forb_overtake	30	mand_pass_left
3	forb_right	31	mand_pass_left_right
4	forb_speed_over_10	32	mand_pass_right
5	forb_speed_over_100	33	mand_right
6	forb_speed_over_130	34	mand_roundabout
7	forb_speed_over_20	35	mand_straight_left
8	forb_speed_over_30	36	mand_straight
9	forb_speed_over_40	37	mand_straight_right
10	forb_speed_over_5	38	prio_give_way
11	forb_speed_over_50	39	prio_priority_road
12	forb_speed_over_60	40	prio_stop
13	forb_speed_over_70	41	warn_children
14	forb_speed_over_80	42	warn_construction
15	forb_speed_over_90	43	warn_crosswalk
16	forb_stopping	44	warn_cyclists
17	forb_trucks	45	warn_domestic_animals
18	forb_u_turn	46	warn_other_dangers
19	forb_weight_over_3.5t	47	warn_poor_road_surface
20	forb_weight_over_7.5t	48	warn_roundabout
21	info_bus_station	49	warn_slippery_road
22	info_crosswalk	50	warn_speed_bumper
23	info_highway	51	warn_traffic_light
24	info_one_way_traffic	52	warn_tram
25	info_parking	53	warn_two_way_traffic
26	info_taxi_parking	54	warn_wild_animals
27	mand_bike_lane		

Table 7: Table showing all the classes our dataset contains and their corresponding id.

10.2 Tools and libraries

The entire project was realized using ROS2 Jazzy and Python 3.11. The main python libraries used in this project are :

- **OpenCV**: library used for computer vision tasks like stereo vision
- **Ultralytics** : library used for object detection related tasks like inference and bounding box extraction

- **Node** : library that enables ROS2 and allows nodes to be created and also node communication over the topics.

OpenCV was used for starting the cameras and capturing frames. It was also used to determine both the intrinsic and extrinsic parameters such as Rotation matrix(R), Fundamental matrix(F), Essential matrix(E), Translation vector(T) and so on. Ultralytics was used to perform inference on the frames captured, to extract the bounding boxes corresponding to the detections in the frames and also determine the center of these bounding boxes that would be used then as target point for depth estimation. Node library is the main component of the ROS2 and it is used to create nodes, run nodes and communicate with other nodes over the network by sending messages over the topics. One thing that needs to be taken into account is that the version of ROS2 needs to be compatible with the Linux version being used. In this project, ROS2 Jazzy was used together with the Linux version Ubuntu 24.04 in order to ensure compatibility.

```
1
2  retL, mtxL, distL, _, _ = cv2.calibrateCamera(
3      objpoints, imgpoints_left, grayL.shape[::-1], None, None)
4  retR, mtxR, distR, _, _ = cv2.calibrateCamera(
5      objpoints, imgpoints_right, grayR.shape[::-1], None, None)
6
7  criteria = (cv2.TERM_CRITERIA_MAX_ITER + cv2.TERM_CRITERIA_EPS, 100, 1e-5)
8  flags = cv2.CALIB_FIX_INTRINSIC
9
10 retStereo, _, _, _, _, R, T, E, F = cv2.stereoCalibrate(
11     objpoints,
12     imgpoints_left,
13     imgpoints_right,
14     mtxL, distL,
15     mtxR, distR,
16     grayL.shape[::-1],
17     criteria=criteria,
18     flags=flags
19 )
20
21 np.savez("stereo_calibration_data.npz",
22          K1=mtxL, D1=distL,
23          K2=mtxR, D2=distR,
24          R=R, T=T, E=E, F=F)
```

Figure 23: This code first calibrates left and right camera individually, then performs stereo calibration(on both cameras simultaneously) in order to compute rotation(R), translation(T), and essential/fundamental(E,F) in order to determine relations between left and right frames. These parameters are then saved in '.npz' format for later use or other stereo vision tasks.

10.3 More detailed statistics

This subsection provides more insights about the performance of the models, YOLOv5 and YOLOv8l. The corresponding P-curve and F1-confidence curve for both models are be presented below.

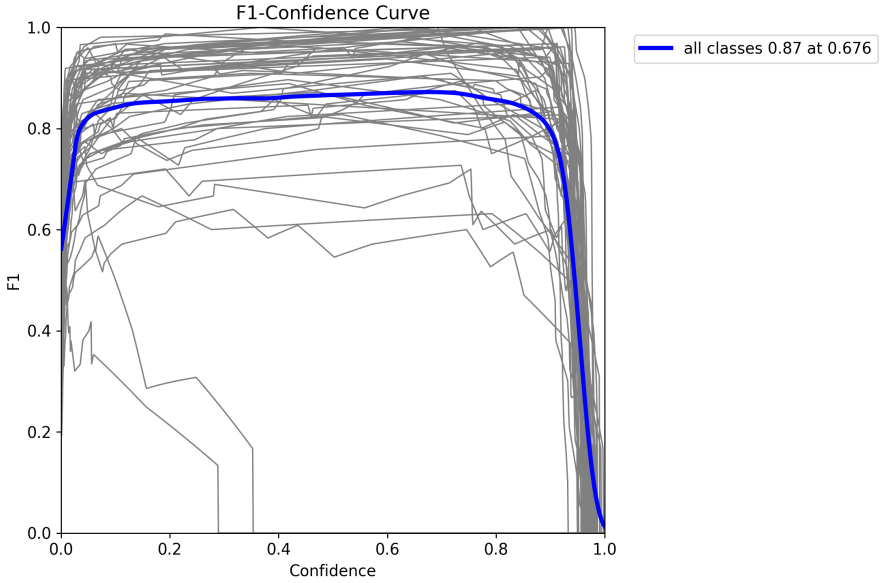


Figure 24: This plot illustrates how the F1 score changes as the confidence threshold varies for YOLOv8l model. Each gray line represents the F1 score for a specific class, while the blue line shows the average F1 score across all classes. The peak(maximum) of the blue curve reaches around 0.90 at a confidence threshold of 0.5, meaning this is the optimal balance point between precision and recall for the model. Setting the confidence threshold around 0.5 is ideal, as it filters out uncertain detections while still preserving accurate ones. Overall, the high and stable F1 scores indicate that the model performs well across most traffic sign classes.

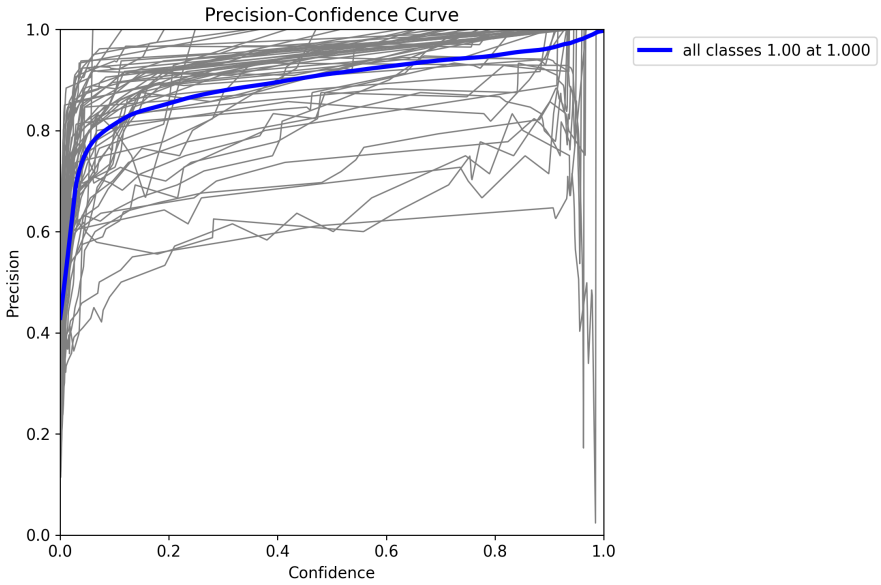


Figure 25: This curve shows how precision varies under different thresholds for YOLOv8l model. Each gray line represents the precision curve of a specific traffic sign class, while the blue line indicates the average precision across all classes. The peak of the average precision is 1.0 and it is achieved at a threshold of 0.999. This means that the model is almost completely confident in its predictions, making no false positives. The blue curve keeps itself really high, close to the top of chart, meaning that all classes are confident in making decisions. Overall, these results suggest that the model is really precise when making confident predictions.

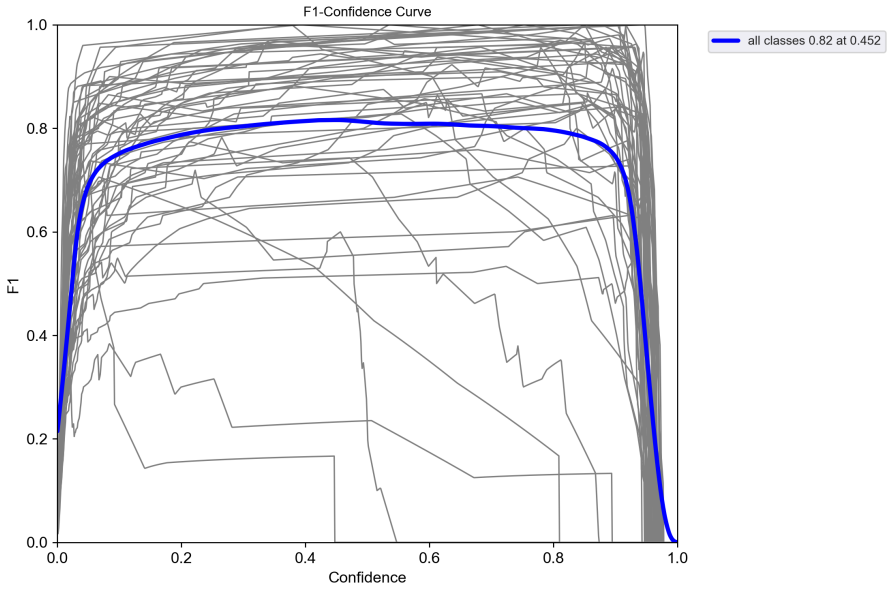


Figure 26: This plot illustrates how the F1 score changes as the confidence threshold varies for YOLOv5 model. Each gray line represents the F1 score for a specific class, while the blue line shows the average F1 score across all classes. The peak(maximum) of the blue curve reaches around 0.81 at a confidence threshold of 0.46, meaning this is the optimal compromise point between precision and recall for the model. Adjusting the confidence threshold around 0.45 is ideal, as it filters out uncertain detections while still preserving accurate ones. Overall, the high and stable F1 scores indicate that the model performs well across most traffic sign classes.

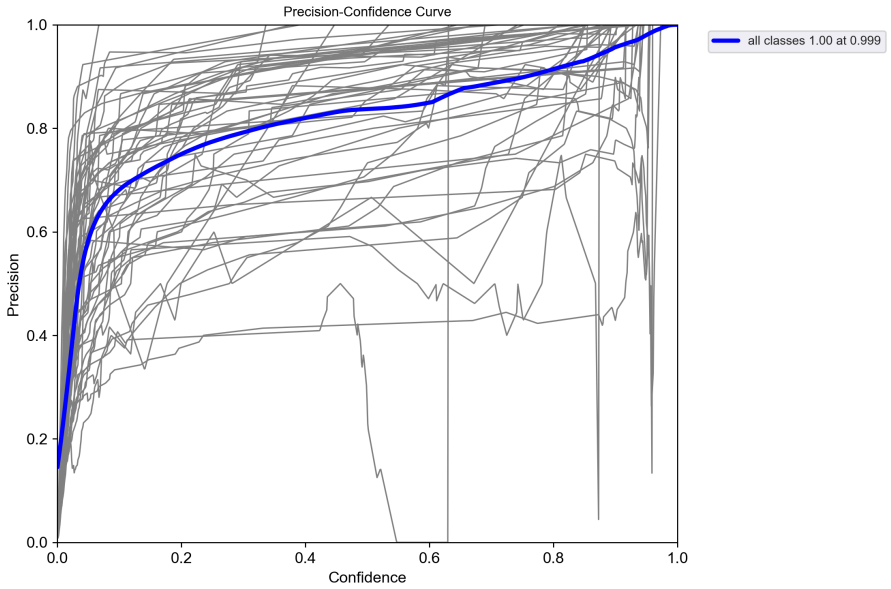


Figure 27: This curve shows how precision varies under different thresholds for YOLOv5 model. Each gray line represents the precision curve of a specific traffic sign class, while the blue line indicates the average precision across all classes. The peak of the average precision is 1.00 and it is achieved at a threshold of 0.999. This means that the model is almost completely confident in its predictions, making no false positives. These results suggest that the model is really precise when making confident predictions.

10.4 ROS2 architecture

The architecture that enables depth estimation consists of 2 nodes that communicate over two topics. The first node is responsible for capturing and publishing the frames over the topics "left_camera\image_raw" and "right_camera\image_raw". The second node, "depth_node", subscribes to the two before mentioned topics, performs object detection over the captured frames and then depth estimation. The code used by these two nodes will be presented in this subsection in Figure 28 and Figure 29.

```
self.model = YOLO('/path/to/best.pt')
self.K1 = np.load("K1.npy")
self.D1 = np.load("D1.npy")
self.K2 = np.load("K2.npy")
self.D2 = np.load("D2.npy")
self.R = np.load("R.npy")
self.T = np.load("T.npy")

self.image_size = (640, 480)
self.R1, self.R2, self.P1, self.P2, self.Q, _, _ = cv2.stereoRectify(
    self.K1, self.D1, self.K2, self.D2, self.image_size, self.R, self.T
)
self.map1x, self.map1y = cv2.initUndistortRectifyMap(
    self.K1, self.D1, self.R1, self.P1, self.image_size, cv2.CV_32FC1)
self.map2x, self.map2y = cv2.initUndistortRectifyMap(
    self.K2, self.D2, self.R2, self.P2, self.image_size, cv2.CV_32FC1)

disparity = abs(cxL - cxR)
if disparity != 0:
    depth = (self.fx * self.baseline) / disparity
```

Figure 28: ROS2 "depth_node" for traffic sign recognition and depth estimation. This Python code loads the trained YOLOv8l model and calibration parameters then performs image rectification and depth estimation. Disparity computation is used in order to estimate how far away traffic signs are from the excavator. Disparity is the shift on the horizontal direction of corresponding pixels. So, this node allows the system to detect traffic signs and calculate the distance to them.

```
1
2  gst_left = (
3      'libcamerasrc camera-name="/.../i2c@80000/imx219@10" ! '
4      'video/x-raw,width=640,height=480,format=NV12,framerate=30/1 ! '
5      'videoconvert ! appsink'
6  )
7  gst_right = (
8      'libcamerasrc camera-name="/.../i2c@88000/imx219@10" ! '
9      'video/x-raw,width=640,height=480,format=NV12,framerate=30/1 ! '
10     'videoconvert ! appsink'
11 )
12
13 self.cap_left = cv2.VideoCapture(gst_left, cv2.CAP_GSTREAMER)
14 self.cap_right = cv2.VideoCapture(gst_right, cv2.CAP_GSTREAMER)
15
16 retL, frameL = self.cap_left.read()
17 retR, frameR = self.cap_right.read()
18 msgL = self.br.cv2_to_imgmsg(frameL, encoding='rgb8')
19 msgR = self.br.cv2_to_imgmsg(frameR, encoding='rgb8')
20 self.pub_left.publish(msgL)
21 self.pub_right.publish(msgR)
```

Figure 29: ROS2 frame publishing node. This code starts by defining the pipelines corresponding to both left and right cameras. The resolution of the frames being published is 640×480. Then the camera capturing nodes are started using 'cv2.Videocapture()' function. Then corresponding left and right frames are captured and published over the topics as messages(msgL, msgR).

10.5 Conclusion(Appendix)

This appendix gives all the necessary tools, dataset details, and code snippets needed in order to reproduce this project. It also shows how YOLO models were evaluated, and how the ROS2-based system was built for performing real-time traffic sign detection and depth estimation. Overall, it complements our thesis well by offering a clear and practical look at the technical implementation behind the results achieved.

References

[1] Alawaji, Khaldaa, Ramdane Hedjar, and Mansour Zuair: *Traffic sign recognition using multi-task deep learning for self-driving vehicles.* Sensors, 24(11):3282, 2024. https://www.mdpi.com/1424-8220/24/11/3282.

[2] Albawi, Saad, Tareq Abed Mohammed, and Saad Al-Zawi: *Understanding of a convolutional neural network.* In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. IEEE, 2017.

[3] Bochkovskiy, Alexey, Chien Yao Wang, and Hong Yuan Mark Liao: *Yolov4: Optimal speed and accuracy of object detection.* arXiv preprint arXiv:2004.10934, 2020.

[4] Bodla, Navaneeth, Bharat Singh, Rama Chellappa, and Larry S. Davis: *Soft-nms — improving object detection with one line of code.* In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[5] Boleo US: *Mini excavator travel speed: How fast can they go?*, 2023. https://www.boleous.com/blog/mini-excavator-travel-speed/, Accessed: 2025-06-07.

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

[7] First Principles of Computer Vision: *First principles of computer vision - youtube channel.* https://www.youtube.com/@firstprinciplesofcomputerv3258, 2024. Accessed: 2025-06-07.

[8] Girshick, Ross, Jeff Donahue, Trevor Darrell, and Jitendra Malik: *Rich feature hierarchies for accurate object detection and semantic segmentation.* Proceedings of the IEEE conference on computer vision and pattern recognition, pages 580–587, 2014.

[9] Gu, Jiuxiang, Zhenhao Wang, Jason Kuen, Junhui Ma, Amir Shahroudy, Bing Shuai, Ting Liu, and Gang Wang: *Recent advances in convolutional neural networks.* Pattern Recognition, 77:354–377, 2018.

[10] Gu, Jiuxiang, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Li Wang, Jianfei Cai, and Tsuhan Chen: *Recent advances in convolutional neural networks.* arXiv preprint arXiv:1512.07108, 2015.

[11] Guo, Yu, Gang Xiong, Zhen Li, Junzheng Shi, Mingxin Cui, and Gaopeng Gou: *Tagan: Gan based traffic augmentation for imbalanced network traffic classification.* In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 3597–3601. IEEE, 2021.

[12] Hosang, Jan, Rodrigo Benenson, and Bernt Schiele: *Learning non-maximum suppression.* In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[13] Iftikhar, Muhammad, Muhammad Saqib, Muhammad Zareen, and Hassan Mumtaz: *Artificial intelligence: Revolutionizing robotic surgery: Review.* Annals of Medicine & Surgery, 86(9):5401–5409, September 2024.

[14] IV, Thomas F. Byrd and Christopher J. Tignanelli: *Artificial intelligence in surgery—a narrative review.* Journal of Medical Artificial Intelligence, 6:20, 2023. https://jmai.amegroups.org/article/view/9200.

[15] Ji, Bin, Jiafeng Xu, Yang Liu, Pengxiang Fan, and Mengli Wang: *Improved yolov8 for small traffic sign detection under complex environmental conditions.* Franklin Open, 8:100167, 2024.

[16] Jin, Shiyu, Liuwang Kang, Yuan Chih Peng, Zhenpeng He, Ruitao Song, Lingfeng Qian, and Liangjun Zhang: *Autonomous excavator system for construction earth moving.* In *ICRA 2023 Future of Construction Workshop Papers*, pages 24–27. IAARC, 2023. `https://www.iaarc.org/publications/icra_2023_future_of_construction_workshop_papers/autonomous_excavator_system_for_construction_earth_moving.html`.

[17] Jud, Dominic, Simon Kerscher, Martin Wermelinger, Edo Jelavic, Pascal Egli, Philipp Leemann, Gabriel Hottiger, and Marco Hutter: *Heap – the autonomous walking excavator.* Automation in Construction, 129:103827, 2021. `https://www.sciencedirect.com/science/article/pii/S092658052100234X`.

[18] Khanam, Rahima and Muhammad Hussain: *What is yolov5: A deep look into the internal features of the popular object detector*, 2024. `https://arxiv.org/abs/2407.20892`.

[19] Khwandah, Shadi Saleh *et al.*: *Traffic signs recognition and distance estimation using a monocular camera.* In *The 6th International Conference Actual Problems of System and Software Engineering (APSSE)*. CEUR Workshop Proceedings, 2019. 97% real-time accuracy using YOLO and active learning.

[20] Kom, Larry: *The role of gpus in training models*, 2025. `https://corescientific.com/resources/blog/unlocking-ai-potential-with-hpc-role-of-gpus-in-training-models/`, Accessed: 2025-05-26.

[21] Kumar, Amara Dinesh, R. Karthika, and Latha Parameswaran: *Novel deep learning model for traffic sign detection using capsule networks.* International Journal of Recent Technology and Engineering (IJRTE), 10(6), 2022. Presented at Amrita School of Engineering, Amrita Vishwa Vidyapeetham, India.

[22] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton: *Deep learning.* Nature, 521(7553):436–444, 2015.

[23] Lowe, David G: *Distinctive image features from scale-invariant keypoints.* International journal of computer vision, 60(2):91–110, 2004.

[24] Macenski, Steve, Alberto Soragna, Michael Carroll, and Zhenpeng Ge: *Impact of ros 2 node composition in robotic systems.* In *Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021. Available at: `https://arxiv.org/pdf/2305.09933` or official IROS proceedings.

[25] Marques, Rafael, Tiago Ribeiro, Gil Lopes, and A. Fernando Ribeiro: *YOLOv3: Traffic Signs & Lights Detection and Recognition for Autonomous Driving.* In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence (ICAART)*, pages 818–826, 2022.

[26] Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar: *Foundations of machine learning.* 2012.

[27] Neuhold, Gerhard, Tobias Ollmann, Samuel Rota Bulo, and Peter Kontschieder: *The mapillary vistas dataset for semantic understanding of street scenes.* In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, page 4990–4999, 2017.

[28] Oprea, Radu: *Traffic signs detection europe*. https://universe.roboflow.com/radu-oprea-r4xnm/traffic-signs-detection-europe, 2023. Accessed: 2025-05-29.

[29] Pan, Yifan, Zhang Yulu, Dakang Zhang, and Yuying Song: *3d printing in construction: state of the art and applications*. The International Journal of Advanced Manufacturing Technology, 115, July 2021.

[30] Peng, Gang and Rahul Bhaskar: *Artificial intelligence and machine learning for job automation: A review and integration*. Journal of Database Management, 2023. https://www.researchgate.net/publication/368800845_Artificial_Intelligence_and_Machine_Learning_for_Job_Automation_A_Review_and_Integration.

[31] Phan, Trong Huy and Kazuma Yamamoto: *Resolving class imbalance in object detection with weighted cross entropy losses*. arXiv preprint arXiv:2006.01413, 2020.

[32] Pire, Taihú, Pablo de Cristóforis, Matías Nitsche, and Julio Jacobo Berlles: *Stereo vision obstacle avoidance using depth and elevation maps*. In *Proceedings of the Argentine Symposium on Artificial Intelligence (ASAI)*, pages 1–10. FCEyN - University of Buenos Aires, 2012. https://example.com/pire2012.pdf.

[33] Quiñonez, Y., M. Ramirez, C. Lizarraga, I. Tostado, and J. Bekios: *Autonomous robot navigation based on pattern recognition techniques and artificial neural networks*, n.d. Unpublished or non-standard publication.

[34] Rahman, Akizur and Ravie Chandren Muniyandi: *Feature selection from colon cancer dataset for cancer classification using artificial neural network*. International Journal of Advanced Computer Science and Applications, 9(9):1–6, 2018.

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

[36] Redmon, Joseph and Ali Farhadi: *Yolov3: An incremental improvement*. arXiv preprint arXiv:1804.02767, 2018.

[37] Ruan, Zhiwei, Xinggang Lin, Guijin Wang, and Jing Hao Xue: *Deformable part-based model transfer for object detection*. IEICE Transactions on Information and Systems, E97.D(5):1394–1397, 2014.

[38] RudderStack: *What is generalization in machine learning?* https://www.rudderstack.com/learn/machine-learning/generalization-in-machine-learning/, 2023. Online article.

[39] Schmidhuber, Jürgen: *Deep learning in neural networks: An overview*. Neural networks, 61:85–117, 2015.

[40] Stallkamp, Johannes, Marc Schlipsing, Jan Salmen, and Christian Igel: *Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition*. Neural Networks, 32:323–332, 2012.

[41] Theiss, LuAnn, Laura Higgins, and Gerald L. Ullman: *Studies to improve temporary guide signs in work zones*. Texas A&M Transportation Institute, 2021. https://rosap.ntl.bts.gov/view/dot/59250.

[42] Toshniwal, Dhruv, Saurabh Loya, Anuj Khot, and Yash Marda: *Optimized detection and classification on gtrsb: Advancing traffic sign recognition with convolutional neural networks*, 2024. https://arxiv.org/abs/2403.08283.

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

[44] Yang, Shouheng, Yiming Liu, Ling Liu, Zhenzhen Liu, and Yonghong Tian: *From yolov1 to yolov8: A review of object detection algorithms*. arXiv preprint arXiv:2304.00501, 2023. https://arxiv.org/abs/2304.00501.

[45] Yaseen, Muhammad: *What is yolov8: An in-depth exploration of the internal features of the next-generation object detector*, 2024. https://arxiv.org/html/2408.15857.

[46] Zhang, Liangjun, Xibin Song, Liyang Wang, Lingfeng Qian, Tianrui Guan, Zhenpeng He, Zhixian Ye, Ruitao Song, Haodong Ding, and Dinesh Manocha: *An autonomous excavator system for material loading tasks*. Science Robotics, 5(49):eabc3164, 2020. https://www.science.org/doi/10.1126/scirobotics.abc3164.

[47] Žujović, Maša, Radojko Obradović, Ivana Rakonjac, and Jelena Milošević: *3d printing technologies in architectural design and construction: A systematic literature review*. Buildings, 12(9):1319, 2022. https://www.mdpi.com/2075-5309/12/9/1319.