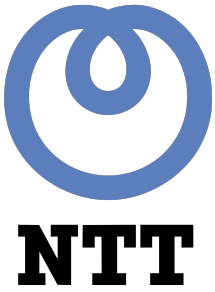
Master thesis : Deep Learning for Automatic Traffic Sign Inventory on an Embedded Device

Auteur : Cabay, Jean-Philippe
Promoteur(s) : Geurts, Pierre
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil en science des données, à finalité spécialisée
Année académique : 2021-2022
URI/URL : http://hdl.handle.net/2268.2/16307



University of Liège - School of Engineering and Computer Science

Deep Learning for Automatic Traffic Sign Inventory on an Embedded Device

Graduation Studies conducted for obtaining the Master's degree in Data Science and Engineering by

Jean-Philippe Cabay

Supervisors

Pierre Geurts
Frédéric Jourdain

Jury

Pierre Geurts
Marc Van Droogenbroeck
Gilles Louppe

Academic Year 2021 - 2022

Acknowledgements

This endeavor would not have been possible without the help of Pr. Pierre Geurts, my academic promoter, as well as M. Frédéric Jourdain, my supervisor who were always present for technical advices, and without whose help I would not have progressed so far in my work.

Thanks should also go to Pascal Mathis for his wise tips.

I had the pleasure of working with the NTT Ltd. team, with whom I have spent an incredible 4 months internship. I learned a lot from them and I will always remember this valuable experience.

Last but not least, I would like to give a special thanks to my family and my partner, who helped me for the annotation of the dataset and, above all, for their moral support.

Abstract

This work focuses on a real-time embedded automatic traffic sign identification solution implemented both in hardware and software. We use a machine vision based traffic sign inventory that consist of several steps : detection, classification and localization of traffic signs.
First, we will explain the main goal of this work, which is improving the driving safety by allowing better road maintenance, and why our team at NTT Ltd. chose to work on this project. We will also state the major problems which we will have to face, and what are the solutions at our disposal.
Second, we will go through the theoretical background needed to implement our solution. This will lead us to our third part which deals with the actual methods used by our embedded system.
We will close this work by presenting the results obtained by our solution, as well as some quick tracks to explore in the future to improve our current method.

Contents

Chapter 1

Introduction

1.1 Motivation

In 2021, 34.640 road traffic accidents took place in Belgium, bringing the total of casualties to 42.566, including 38.952 slightly injured, 3.098 seriously injured and 516 persons who died within 30 days of the accident. [38] This number of accidents is still too high but can be reduced by providing a safer road environment through better maintenance of the current infrastructure. The maintenance of traffic signs represents simple and cost-effective interventions that can have an impressive rate of return in terms of road safety and thus reduced socio-economic consequences.

Traffic signs play an essential role in maintaining road safety by providing drivers with important information about the road and its surroundings. They convey information through their shape, colors and pictograms. It is therefore necessary to ensure the visibility of these features through their placement and regular maintenance.

The road infrastructure should encourage a safe and smooth flow of traffic and reduce the risk of evaluation and driving errors on the part of road users. Traffic signs deserve special attention, as most of the information used during driving is perceived by the eyes. Unfortunately, the maintenance of road infrastructure is often lacking. In Germany for example, out of the 25 million traffic signs present across the country, it is estimated that 33% are non-readable and 25% are more than 15 years old. In practice, this means that many traffic signs fail to perform their basic function, thus creating an increasingly hazardous road environment. [27]

Until now, the assessment of the condition of signs has been mainly done by time-consuming manual measurements. A modern technology solution could help today to develop a local inventory database that public agencies could use to manage their signage assets, which would allow them to obtain the geolocation of the sign (to verify its correct location) and to measure the technical condition (intactness, color, contrast or retro-reflectivity) of their signs. An automatic traffic sign inventory solution can help to carry out sign surveys, assess compliance with national standards, facilitate sign maintenance and ensure timely replacement.

1.2 Presentation of NTT Ltd.

NTT Ltd. is a leading IT infrastructure and services company and part of the global NTT corporation. With revenues of over USD 10 billion, they employ more than 40.000 people in a diverse and dynamic workplace that spans 57 countries, trading in 73 countries and delivering services in over 200 countries and regions, and serve 5.000 clients.

Working with organizations around the world, they achieve business outcomes through data-driven, connected, digital, and secure technology solutions. They provide expertise in connectivity, cloud transformation, cybersecurity, IoT, big data, artificial intelligence and analytics. Their products include Private 5G, Network as a Service, Multicloud as a Service, Edge as a Service and Software-defined Infrastructure Services. [2]

In their Client Innovation Centers, they co-innovate with their clients, from ideation to proof of concept, to solve business challenges together, invent new solutions and create market differentiation through innovation. These center of reference, present only in Belgium and Japan, serve also as a showcase to the new technologies and use cases developed by NTT R&D and all their partners.

1.3 Objectives and Challenges

NTT's Client Innovation Center in Diegem works in a co-innovation partnership with a client having large fleet of vehicles travelling throughout Belgium. This partnership has the end-goal to install an embedded solution monitoring traffic signs on each vehicle in the client's fleet to ensure a large coverage of the country's

traffic signs. The final solution combining edge and cloud to create an automatic inventory system.

This thesis will focus on the development of a Proof of Concept for an embedded traffic sign inventory system. The solution is expected to run in real-time and to detect, classify and estimate the precise geo-coordinates of traffic signs.

Because the end-goal is to deploy the solution on an entire fleet of vehicle, the hardware used should not be too expensive and should not be too power hungry as it is supposed to work the whole day on an external or the vehicle's battery. These conditions limit the hardware and algorithms we can pick.

It is also important for this work to create a solution that can integrate face and number plate anonymisation without too many changes in the future, in order to follow the requirements of the EU General Data Protection Regulation.

The traffic sign detection and classification model will be based on the one-stage YOLOv4 detector. Traffic sign position will be estimated by combining the vehicle's position with the relative position of the traffic sign to the camera obtained through stereo depth.

Chapter 2

Context

2.1 Traffic Signs in Belgium

Belgium is part of the 83 countries that have signed and ratified since 1988 the Vienna Convention on road signs and signals [35]. The Vienna Convention on Road Signs is a multilateral treaty that aims to achieve standardization of traffic signs and road markers among different countries. The convention divides traffic signs into categories based on the type of information they convey and recognizable by their color and shape.
Belgian traffic signs can be divided into seven main categories:

1. **Danger warning signs**
2. **Priority signs**
3. **Prohibitory signs**
4. **Mandatory signs**
5. **Stopping and Parking signs**
6. **Indication signs**
7. **Additional signs**

There are still differences between the traffic signs of the countries that have signed the Vienna Convention. Even inside Belgium, the same traffic sign can have slightly different versions.

Official traffic signs in Belgium are defined in the Royal Decree 1 Dec 1975 on the traffic police and the use of public life. They have to comply with the police regulations and the regulation of each region.

Belgium has 3 regions with their own competences. In concrete terms, this means that depending on whether you live in Flanders, Wallonia or Brussels-Capital, you may be subject to different rules (dimension of traffic signs, speed limit).

2.1.1 Dimension of traffic signs in Belgium

The Ministerial Decree of 11 October 1976, defines the minimum dimensions and special conditions for the placement of traffic signs. This decree has been amended by each region such that installation of traffic signs is part of the competence of each region.

As illustrated in the Figure 2.1, the dimensions depend on the shape of the sign and the speed limitation of the road they are next to (Region Wallonia) or whether the road is in a build-up area or outside and the type of road (Region Brussels-Capital and Flanders).

Brussels-Capital Region and Flemish Region) In built-up areas, road signs having the shapes drawn below have the following minimum dimensions

Normale afmetingen	0,70 m	0,90 m	0,60 m	0,40 m	Dimensions normales
Uitzonderingen wegens plaatselijke omstandigheden.	0,40 m	0,40 m	0,40 m	0,40 m	Exceptions dues à des circonstances locales.

In Brussels-Capital Region and Flemish Region (Outside built-up areas)

Autosnelwegen		1,10 m	—	0,90 m	0,90 m	Autoroutes
Autowegen		1,10 m	0,90 m	0,90 m	0,40 m	Routes pour automobiles
Andere wegen	Minder dan 4 rijstroken	0,90 m	0,90 m	0,70 m	0,40 m	Moins de 4 bandes — Autres routes
	Ten minste 4 rijstroken	1,10 m	0,90 m	0,90 m	0,40 m	Au moins 4 bandes
Uitzonderingen wegens plaatselijke omstandigheden.		0,40 m	0,40 m	0,40 m	0,40 m	Exceptions dues à des circonstances locales.

In Wallonia Region: Article 6.4 (amended by AM 30/06/2020) defines the minimum dimension according to the shape of the road sign for speed limit

Forme du signal / Limitation de vitesse				
Plus de 90 km/h	1,10 m	0,90 m	0,90 m	0,90 m
Supérieure à 50 km/h et inférieure ou égale à 90 km/h	0,90 m	0,90 m	0,70 m	0,40 m 0,90 m pour les signaux B11 et B13
Supérieure à 30 km/h et inférieure ou égale à 50 km/h	0,70 m	0,90 m	0,60 m	0,40 m 0,90 m pour les signaux B11 et B13
Inférieure ou égale à 30 km/h	0,40 m	0,40 m	0,40 m	0,40 m

Figure 2.1: Rules for Traffic Sign Dimensions in Belgium

But exception exist upon the region, for example in Wallonia depending on local circumstances[1].

- smaller signs, corresponding to a lower speed limit, may be used upon taking local circumstances and when repeating on the left size of the road.[2]
- the range of a larger signal may be used at the end of a 30 km/h or less speed limit zone
- and dimensions can even be reduced to a minimum of 0.30 m for road signs with announcements to be observed by cyclists, speed pedelecs and two-wheeled moped drivers

2.1.2 Maintenance of traffic signs in Belgium

Ministerial Decree of 11 October 1976 article 6.3.: "Road signs shall be kept as clean as possible so that they remain identifiable by users." In Belgium, it is the authorities which places the sign which are responsible to maintain traffic signs in a good state. There are no specific directives on when a sign should be replaced other than they should be maintained in state of cleanliness, legibility and retroreflectivity that allows their unambiguous identification at all times.

2.2 Metrics [28]

In this section we will introduce the metrics that we will be using in this work to evaluate the performances of the different elements of our solution.

Confusion matrix is a matrix describing a model's performance for classification problems by comparing the models prediction to the ground truth. For binary classification, as illustrated in Table 2.1, the matrix is composed of the four combination of predicted and actual values:

- **True Positive (TP):** The number of samples where the model's prediction is positive and the ground truth is True.

[1]By local circumstances referred to in Article 6.4.1the dimension of the sign can be adapted for example in the following circumstances: the narrowness of the carriageway; the need to place the signal on a small island

[2]Signs shall be placed in such a way that they can be easily recognized in time by the drivers to whom they are addressed. They shall normally be placed on the side of the road corresponding to the direction of traffic; however, they may be placed or repeated above the carriageway.

- **False Positive (FP):** The number of samples where the model predicted positive but the ground truth is False.
- **False Negative (FN):** The number of samples where the model predicted negative but the ground truth is True.
- **True Negative (TN):** The number of samples where the model's prediction is negative and the ground truth False.

For multiclass cases, each row of the confusion matrix represents a class and it will be calculated based on a one-vs-the-rest approach.

		Actual	
		True	False
Predicted	Positive	TP	FP
	Negative	FN	TN

Table 2.1: Binary confusion matrix

From the confusion matrix, multiple metrics can be obtained:

$$\textbf{Error rate} = \frac{FP + FN}{TP + FP + TN + FN}$$

$$\textbf{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

$$\textbf{Precision} = \frac{TP}{TP + FP}$$

$$\textbf{Recall} \text{ or } \textbf{Sensitivity} = \frac{TP}{TP + FN}$$

$$\textbf{Specificity} = \frac{TN}{TN + FP}$$

$$\textbf{F1-score} = \frac{2.\text{Precision.Recall}}{\text{Precision+Recall}}$$

The F1-score is a useful metric because by finding the confidence threshold that maximizes the F1-score, we will obtain the best balance between precision and recall.

Precision-Recall curve is a graph representing the trade-off between precision and recall for different thresholds.

Average Precision (AP) is a metric for binary classification problems which summarises the precision recall curve into one number by calculating the weighted

mean of precisions for each threshold. Where the weight represents the increase in recall from the previous threshold.

$$AP = \sum_{k=1}^{N} p(k)\Delta r(k)$$

Mean Average Precision (mAP) is the mean of the average precision metric applied to multiclass cases. For each class, the average precision will be calculated based on a one-vs-the-rest approach. Then the mAP can be obtained by taking the mean of the average precision for each class.

$$mAP = \frac{1}{n_C} \sum_{i \in C} AP_i$$

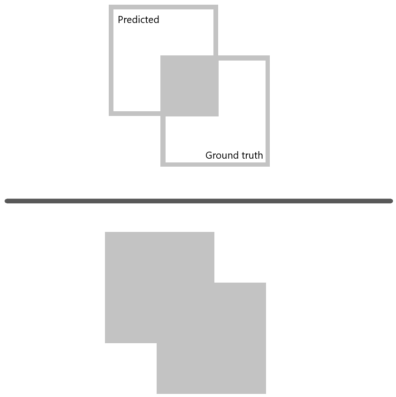


Figure 2.2: Intersection Over Union

Intersection over union (IoU) is a metric measuring the similarity between two sets. In object detection, it is used to evaluate the similarity between a model's predicted bounding box and the ground-truth box as illustrated in Figure 2.2.

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$

2.3 Traffic Sign Detecion

With the rise in research for the development of autonomous vehicles, many methods have been proposed to detect traffic signs in images. Traffic sign detection is composed of two sub-tasks, localization and classification.

2.3.1 Challenges [41] [4]

Traffic sign detection possess many types of challenges.

The first challenge comes from the diversity of sign that exist. As said earlier, the same sign can have its appearance vary from a country to another but even inside of the same country it is not uncommon to find multiple slightly different version of the same sign. This intraclass variability can make detection more challenging. Also, the vast majority of methods only consider static signs, there are very few works where dynamic, electronic or variable message signs are included.

The road environment also presents a challenge for detection systems, due to the presence of many objects that resemble traffic signs in terms of shape and color. For this reason, a high precision is usually harder to attain compared to recall.

Another important element influencing the detection performances is visibility. The visibility of traffic signs can be affected by multiple factors. Traffic signs may be damaged, occluded, have a bad orientation or have their color fading. Because detection system are usually installed inside moving vehicles, the movement can introduce motion artifacts and blur. Lighting condition, the time of day, reflections and weather conditions can also influence the detection capabilities.

It is currently quite difficult to compare traffic sign detection methods, as researchers usually consider different sign sets or use signs from different countries. In addition, the datasets used are often private due to the absence of very large and well annotated public datasets. The most commonly used public dataset is the German Traffic Sign Detection Benchmark (GTSDB) [29] covering 43 classes with only 900 images.

2.3.2 Classical approaches [41] [4]

Classical approaches to traffic sign detection rely primarily on the colors and geometric shapes of the signs. Some of these approaches achieved excellent results.

Because traffic signs need to be noticed quickly by drivers, they have a set of bright and distinctive colors that differentiate them from the background. **Color-based methods** use color information to segment regions of interest inside the image. Popular color-based methods use color thresholding on different color spaces, others use color indexing or region growing to achieve the same goal.

The advantage of color-based methods is that they are generally computationally cheap, making them suitable for high-resolution images. However, these methods are generally sensitive to various factors such as distance from the sign, weather conditions, time of day, reflections and age of the sign. [41]

Just as traffic signs have specific colors, they also have very well-defined shapes that can be searched for. **Shape-based methods** ignore the color in favor of the characteristic shape of signs. These methods usually are based around the detection and processing of edges in grayscale images using Hough transforms. Another popular approach is template matching, where a predefined sign template is searched through the entire image using windows of different size.

Shape-based methods are usually more robust to various lighting condition. However they are also more computationally expensive. In addition, shape-based methods do not handle damaged, partially obscured, faded and blurred traffic signs very well.

Because color-based and shape-based methods have both advantages and disadvantages, **hybrid methods** have been developed to take advantage of both colors and shape features.

The main drawback of the classical methods is that most of them focus on the detection of only one type of sign. Some can detect more types but they can not identify the label of each detection on their own, another classification method is then required.

2.3.3 Machine learning approaches [41] [4]

Since the success of the first application of the Viola-Jones detection model to detect triangular signs [], many researchers have focused their efforts in the development of machine learning based detection methods.

We can use classical or learning-based methods to localize regions of interest within the image, and then these regions can be fed to a classification algorithm that will associate a label with each region. Detection methods that split the work between localization and classification like this are called two-stage detectors.

In [11], detection is performed using maximally stable extremal regions and then a cascade of linear support vector machines classifiers using the HOG feature descriptor, label the regions with one of 43 classes. The method achieved a true positive rate of 83.3% and a false positive rate of 85% on their own private dataset. The method is fast but not sufficient for real-time detection. Others used random forest and HOG features to classify regions in images.

Nowadays, traffic sign detection is usually addressed with convolutional neural networks. They can achieve better results than previous methods and can be made more robust to adverse conditions where visibility is low. Some one-stage architecture can achieve real-time performances on appropriate edge hardware. The disadvantage of using deep learning is that a large amount of annotated data is required for the model to learn a good representation.

When real-time performance is the objective, many authors([17] [7] [10]) get the inspiration for their model from the YOLO architecture. In [7] they used compared the performances of YOLOv3 and YOLOv4 trained on a synthetic dataset generated by a Generative Adversarial Network an they achieved respectively 84.9 and 89.33% accuracy on their private dataset using 4 classes. With their improved YOLOv4-tiny model, in [17] they achieved 64.52% recall and 70.1% precision on the 45 classes on the TT100K dataset [44].

Chapter 3

Theoretical Background

3.1 You Only Look Once

Many two stage detectors achieve excellent performances on various types of detection task by (1) extracting regions inside of the image containing an object and then (2) applying a classification algorithm on each of the proposed regions. Because of the division of the system into different steps, these detectors are usually computationally heavy and hard to optimize for speed.

Joseph Redmon and his colleagues published in 2015 a paper called " *You Only Look Once: Unified, Real-Time Object Detection*", where they proposed a single neural network architecture performing all the different steps, a single-stage detector. At the time of its release, it offered remarkable performance in terms of speed and accuracy compared to alternative detection methods. Since then, YOLO has had several major updates.

For its real-time capabilities and state-of-the-art results, the object detection model we will use in this thesis is based on the YOLO architecture. We also made this choice in order to support the integration constraint of face and license plate anonymisation without increasing our inference time too much. Indeed, we would only need to add two classes to the model and train it on new data sets instead of adding a detector running in parallel. However, we are also aware that adding two classes could impact the quality of the detections made by the model.

The following subsections will present the key improvements in the evolution of the YOLO model.

3.1.1 YOLO [12]

The main idea of YOLO is to divide the image into a S by S grid where each cell of the grid predicts C class probabilities and B bounding boxes coordinates as well as the confidence score for these boxes. An illustration is provided on Figure 3.1. The confidence score, also called objectness score, is defined formally as $Pr(\text{object}).IOU(\text{truth}, \text{object})$ and represents the probability that the centre of an object will fall into the grid cell together with the quality of the bounding box for that object.

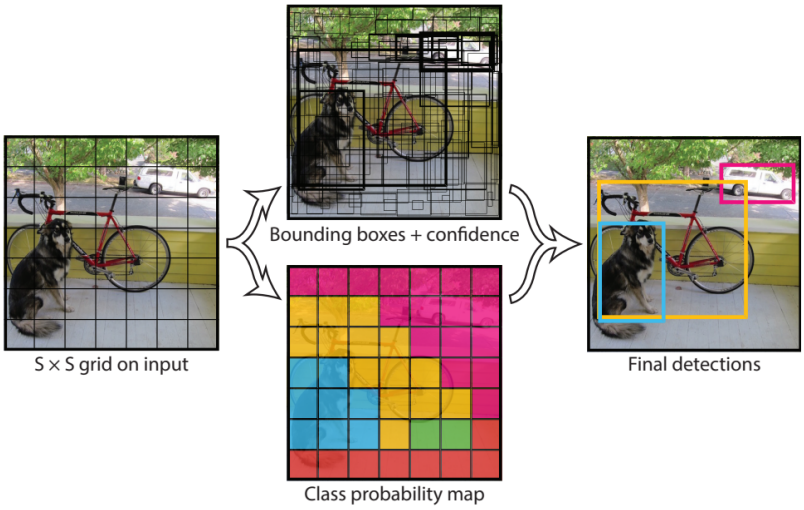


Figure 3.1: YOLO workflow [12]

The YOLO model is based on the Darknet architecture and composed of 24 convolutional layers followed by 2 fully connected layers where all the activation functions are Leaky Rectified Linear Unit (leaky ReLU) except for the final layer using a linear activation function. The output predictions of the last fully connected layer is reshaped into a tensor of shape $S \times S \times (B * 5 + C)$. The original YOLO network is illustrated in Figure 3.2.

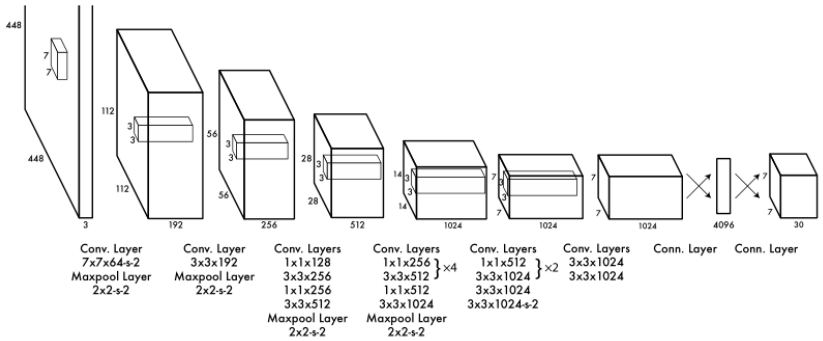


Figure 3.2: Original YOLO architecture [12]

The **loss function** $\mathcal{L}_{\text{YOLO}}$ used to train the network is the combination of a localization loss, a confidence loss and a classification loss.

$$\mathcal{L}_{\text{YOLO}} = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \tag{3.1}$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \tag{3.2}$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \tag{3.3}$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \tag{3.4}$$

$$+ \sum_{i=0}^{S^2} \mathbf{1}_{ij}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \tag{3.5}$$

where λ_{coord} and λ_{noobj} are balancing parameters;
$\mathbf{1}_{ij}^{\text{obj}}$ is a binary variable indicating whether cell i contains an object and whether box j in cell i is responsible for detecting that object (between the B boxes, box j has the highest IoU with that object's box);
$\mathbf{1}_{ij}^{\text{noobj}}$ is a binary variable indicating whether there is no object in cell i or whether there is an object, but box j is not responsible for that object.

The **localization loss** is composed of the two first terms of the full loss.

The first term (3.1) evaluates the deviation from the ground truth bounding box through the sum of square error between the predicted box center (x_i, y_i) and the ground truth box center $(\hat{x}_i, \hat{y}_i)$ for all $B * S^2$ bounding boxes.

The second term (3.2) is the sum of square error between the square root of the predicted box's width and height (w_i, h_i) with the square root of the ground truth box's width and height $(\hat{w}_i, \hat{h}_i)$ for all $B*S^2$ bounding boxes. They used the square root of width and height such that little differences are more penalized in small bounding boxes than in large bounding boxes.

The **confidence loss** is composed of the third and fourth terms of the full loss.

The third term (3.3) uses the sum of the squared errors on the predicted confidence score of the boxes representing an object contained in their cell. In this case, the ground truth is $\hat{C}_i = 1$ and the predicted score $C_i \in [0; 1]$.

The fourth term (3.4) uses the sum of the squared errors on the predicted confidence score of the boxes that do not represent an object. The balancing parameter λ_{noobj} is used to make this loss smaller compared to the third term. In this case, the ground truth is $\hat{C}_i = 0$ and the predicted score $C_i \in [0; 1]$.

The **classification loss** is the last term of the full loss.

The last term (3.5) sums the squared errors of all the classes probabilities in all S^2 cells.

When an object is present in more than one cell, the object may be detected by several cells and when it happens during inference we need to use **non-maximal suppression** (NMS) to keep the best detection. The first step of NMS is to filter out all bounding boxes having a confidence score inferior to a user defined threshold. Then we select the detection with the highest confidence score as a true detection and remove all the detections having an IoU score with it greater than another user defined threshold. When this is done, we select the next detection with the highest confidence score from those remaining and repeat the process until all bounding boxes have an IoU score below the threshold with each other.

The original YOLO model has several **limitations**. The first limitation appears when dealing with close objects because each cell can propose only B bounding boxes and one class for the whole cell. For the same reason, the total number of objects YOLO can detect is limited to $B * S^2$. Comparatively to two-stage models

such as Faster-RCNN, YOLO makes lots of localisation errors, has a lower recall and has more difficulties to detect small objects.

3.1.2 YOLOv2 [13]

A number of iterative improvements have been made to YOLOv2 compared to YOLO, notably the use of batch normalization, classifier training with increased resolution, multi-scale training and the model becoming fully convolutional with anchor boxes. YOLOv2's main focus will be to reduce the localization errors and to allow for multi-object detection in the same cell.

Anchor boxes, also called prior boxes, are a set of N pre-computed bounding boxes designed to represent the typical shapes and sizes of objects in the training data. They are usually computed using k-means clustering on the bounding boxes of training set to create N clusters based on the IoU score between each bounding box.

Unlike YOLO, YOLOv2 infers the bounding box around the detected object not as an arbitrary rectangle, but as an offset from one of the anchor boxes. As illustrated by Figure 3.3, when considering a cell c having an offset of (c_x, c_y) from the top left corner of the image, an anchor box with a width P_w and height P_h, and the 5 network predictions $(t_x, t_y, t_w, t_h, t_o)$, then the predicted bounding box correspond to:

$$b_x = \sigma(t_x) + c_x \tag{3.6}$$

$$b_y = \sigma(t_y) + c_y \tag{3.7}$$

$$b_w = P_w e^{t_w} \tag{3.8}$$

$$b_h = P_h e^{t_h} \tag{3.9}$$

$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o) \tag{3.10}$$

where $\sigma()$ is the sigmoid activation function, (b_x, b_y) are the center coordinates of the bounding box, b_w is the width, b_h is the height associated to the bounding box.

The use of anchor boxes makes it easier for the model to learn and allows the detection of multiple objects in each grid cell. Indeed, each cell will have N anchor boxes and predict the class and confidence score for each box. The fully connected layers of YOLO are removed, making the model fully convolutional with an output of shape $S \times S \times N * (4 + 1 + C)$.

The paper does not state any changes to the loss function.

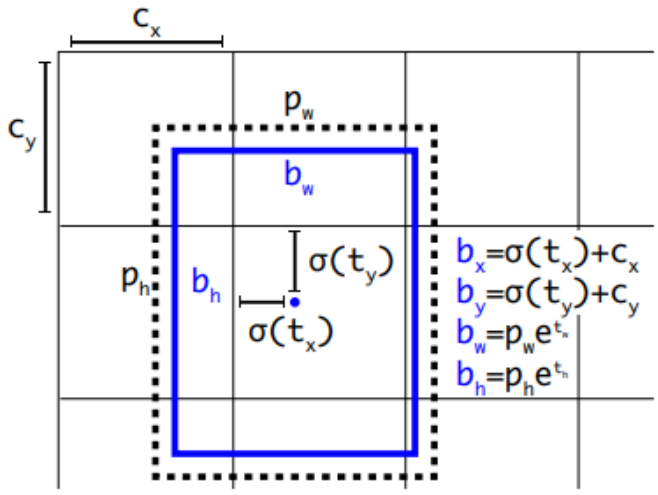


Figure 3.3: Bounding boxes with dimension priors and location prediction [13]

3.1.3 YOLOv3 [14]

YOLOv3 made several big architecture changes by introducing skip connection layers, multi-label classification, an updated loss and a new multi-scale architecture. The main focus of this update is to improve overall performances and small objects detection.

Multi-label classification is now allowed by YOLOv3. In most classifiers, the class labels are assumed to be mutually exclusive and, therefore, previous versions of Yolo used a softmax function to transform the logits into probabilities whose sum adds up to one. In YOLOv3, the softmax activation is replaced by a sigmoid activation that makes each class prediction the output of an independent logistic classifier. This allows an object to be characterized by multiple labels. Replacing the softmax function has also the effect of reducing the computational complexity.

In this paper, more information is given on how the **loss function** has evolved. Now, each ground truth object is associated with a single anchor box, the one that has the greatest overlap with it. This associated anchor box must have a confidence score of 1. If other anchor boxes overlap with the object with an IoU above a predefined threshold, their confidence score is not penalised. Otherwise, for all anchor boxes that are not assigned to a ground truth object, only the confidence loss is applied and not the location and classification losses.

For the **classification loss** and the **confidence loss**, the sum of squared errors are dropped for the binary cross-entropy loss.

The **localization loss** still uses the sum of squared errors directly applied to the network coordinate predictions (t_x, t_y, t_w, t_h) where the ground truth $(\hat{t}_x, \hat{t}_y, \hat{t}_w, \hat{t}_h)$ are obtained by inverting the equations (3.7), (3.8), (3.9) and (3.10) respectively.

The **architecture** of YOLOv3 makes predictions at 3 different scales. To make multiscale predictions, The authors were inspired by the **Feature Pyramid Network** [22] (FPN) design, in which the feature map gradually decreases in spatial dimension before increasing again while being concatenated with previous feature maps of corresponding size. Then the feature maps of different sizes are sent to distinct detection modules and each of them predicts classes and bounding boxes. The FPN design has the advantage of introducing global context to all the output feature maps while preserving spatial information. In the YOLOv3 architecture, the feature map with decreasing spatial dimensions is the backbone, the detection modules are the head and the layers in between are called the neck. The YOLOv3 model architecture is represented in Figure 3.4. The backbone of the model is Darknet-53, a larger version of Darknet-19 used in YOLOv2 updated with skip connections.

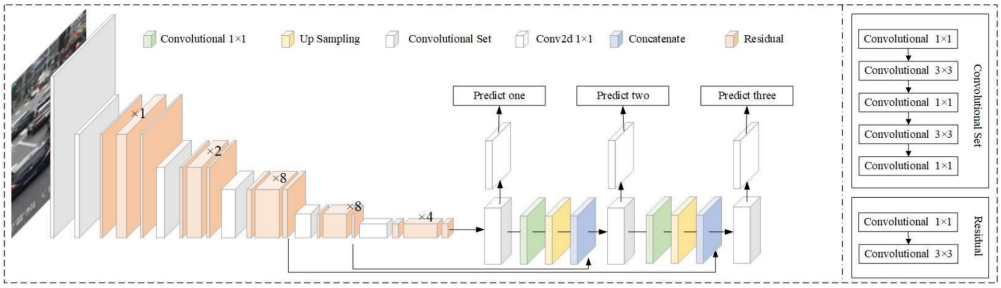


Figure 3.4: The architecture of YOLOv3 using Darknet-53 as a backbone [19]

3.1.4 YOLOv4 [6]

In YOLOv4, the authors tried multiple combinations of different recent technological and architectural advancements known to work well in order to find the most effective. We will focus on the elements that are the most relevant for us.

The backbone

YOLOv4 uses CSPDarknet53, it consist of two blocks: standard convolution block and Cross Stage Partial block.

Cross Stages Partial connection block (CSP) [8] used by YOLOv4 separates the input feature map into two part, one part will go through a dense block and a transition layer and the other part is not processed. The two part are then combined to form the output feature maps. The dense block is a sequence of n convolutions where the output of each convolution is concatenated with its input. This blocks preserves fine-grained features during forwarding and allows for more gradient to flow across the network which alleviates the vanishing gradient problem.

The neck

The Spatial Pyramid Pooling block (SPP) [15] directly follows the backbone to increase the receptive field and separate the most important features by pooling multi-scale versions of its input. The SPP used by YOLOv4 work as follows: First, before entering the SPP block the output feature map of the backbone is reduced from 1024 to 512 by a 1x1 convolution. Then, the input feature maps are duplicated into 3 versions where each version is max-pooled with different sized kernels. Finally, padding is used to keep the size of the 3 output feature maps equal to the input size and the 3 of them are concatenated with the input feature maps.

Path Aggregation Network (PANet) [20] is an improvement over the FPN design improving information flow to low-level feature maps. It does it by adding a bottom up path augmentation to the top-down path used by the FPN as illustrated by Figure 3.5. The PANet used by YOLOv4 is slightly different from the original, when feature maps are merged they use concatenation instead of addition.

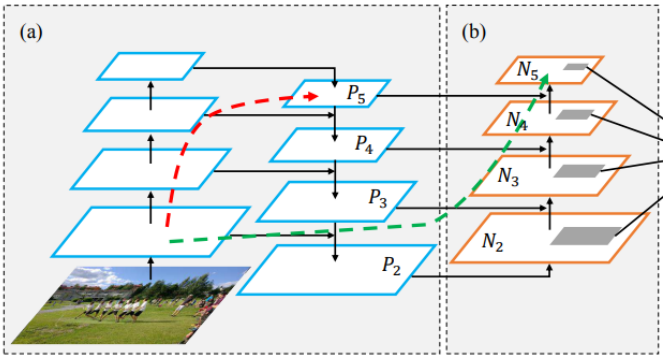


Figure 3.5: PANet [20]

The head

The head of YOLOv4 is composed of three modules each taking a multi-scale output feature maps of the neck. The modules use standard convolutions and their final layer use a linear activation to output a feature map with $N(C+1+4)$ channels.

The loss

The IoU score is a good metric for assessing the similarity between two bounding boxes based on their overlap area. Therefore, it would be an appropriate loss for learning bounding box regression. The **GIoU loss** [36] takes this idea but also includes information about the shape and orientation of the object. Meanwhile, the **DIoU loss** [42] works by incorporating the normalized distance between the central points of the two boxes. An alternative to the GIoU is the **CIoU loss** [43] which adds information about the normalized distance between the central points and the aspect ratios of bounding boxes.

YOLO v4 uses the CIoU loss as a bounding box loss, as it leads to faster convergence and better performance compared to the other loss mentioned.
The equation of the CIoU loss is the following:

$$\mathcal{L}_{\text{CIoU}} = 1 - \text{IoU} + \frac{\rho^2(b, b^{gt})}{c^2} + \alpha v \tag{3.11}$$

where b and b^{gt} represent the central points of our bounding boxes, $\rho(.)$ is the euclidean distance, c is the diagonal length of the smallest enclosing box covering the two boxes, v measures the consistency of aspect ratio and α is a balancing parameter defined as $\alpha = \frac{v}{(1-\text{IoU})+v}$.

Data augmentations

Data augmentation techniques are a powerful tool to make a model more robust by increasing the variability in the training set without collecting more data.

Mosaic data augmentation consists of grouping four training images together. This makes it easier to train the model to find smaller objects and to focus less on the environment near the object.

CutMix data augmentation [21] randomly masks out square regions of the input and replace the removed regions with a patch from another image. The ground truth labels are mixed proportionally to the number of pixels of combined

images. This technique helps the model to learn objects representation from partial views.

Post Processing

DIoU-NMS uses DIoU in non-maximal suppression by taking the IoU and the distance between the center points of two bounding boxes into account during suppression. This approach makes the method more robust for occlusion cases.

3.2 Hardware overview

In this section, we will present the components we use but we will not yet explain why we have chosen a certain type of technology. The technological choices will be motivated later in this chapter.

After having explored the market, we have selected the following equipment for our edge system. It is composed of an Asus Thinker Edge R, to which will be connected a GPS receiver and a Luxonis OAK-D stereo camera.

The GPS receiver we will be used to collect data about the current position of the device. It's reference is G-MOUSE VK-162 and it has a 10Hz receive frequency.

The OAK-D is more than just a stereo camera, we will be using it to capture image data, estimate depth, run the object detection model and measure the orientation and linear acceleration of the device.
It has several components:

- 2 monochrome cameras with a resolution of 1280x800p for depth estimation
- A color camera with a resolution of 4032x3040p
- An Intel movidius Myriad X Vision Processing Unit (VPU) with 16 SHAVE cores (128-bit very long instruction word (VLIW) vector processors). It has 4 TOPS of computing power where 1.4 TOPS is dedicated to run AI models.
- An inertial measurement unit (IMU) composed of an accelerometer, a gyroscope and a magnetometer.

The Asus Thinker Edge R is an ARM based edge computer running on debian 10. It will receive position, orientation, acceleration, depth and detection data from its peripherals, which it will use to track and locate traffic signs. The thinker edge also contains an accelerator for neural networks, but this feature will not be used in our work.

The camera is installed inside the vehicle using a mount attached to the windshield with a suction cup and the GPS receiver is magnetically attached above the camera on the roof of the car.

3.3 Hardware Aware Optimization

Due to the rising interest in edge AI application, a multitude of different efficient neural network architecture are published every year. They are usually evaluated on different type of hardware making the comparison of different models more difficult. Indeed, due to differences in the hardware, a model very fast on a CPU may run poorly on a GPU. Also some inference libraries are more optimized for some types of neural network blocks and do not support others. Therefore it is important to understand well the target hardware and the inference libraries used when building a model.

In the context of this thesis, we want to build a model that runs efficiently on the Intel Movidius Myriad X VPU under the OpenVino runtime. The VPU was built to take advantage of highly parallel workloads. OpenVino supports on VPUs special neural network optimizations in the form of layer fusion, joining and decomposition. Layer fusion consist in merging a convolution layers and an activation function (ReLU or LeakyReLU) into a single layer. Layer joining consist of merging layers located on topologically independent branches that can be executed simultaneously on the same hardware units if resources are available. Layer decomposition can tile convolution and pooling layers. [1]

A common way to adapt YOLO to different type of hardware is by replacing the backbone or the blocks of the model by others. On CPU, usually backbones such as ShuffleNet, MobileNet or SqueezeNet work well, whereas on GPU fast backbones are VGG, ResNet or DenseNet. For VPUs, the authors of the YOLOv4 paper recommend using lighter backbones such as MobileNet, MixNet, GhostNet or EfficientNet-Lite. [6]

3.3.1 RepVGG [24]

We will now present the main building block we will be using in our model.

RepVGG is an architecture designed as a multi-branch model that can be converted to a VGG-like model with successive stacks of 3×3 convolutions and

ReLUs via structural re-parametrization. The model gives identical results during inference but is highly optimised by modern computing libraries. The architecture is illustrated in Figure 3.6.

Many recent state-of-the-art models use multi-branching, where the input passes through different layers before being aggregated. Multi-branching architectures are useful for facilitating learning by reducing the vanishing gradient problem, which leads to higher accuracies. However, many of these architectures use complicated blocks to obtain a low number of Floating Point Operations (FLOPs), but do not take into account the high memory access costs introduced by the multiple branches. Furthermore, these complicated architectures can reduce the ability to parallelize computations, making them less suitable for highly parallel processing units. As a result, a single-stage model such as VGG-16 has 8.4 more FLOPs than the multi-branch EfficientNet-B3 model but runs 1.8 times faster on a 1080ti GPU.
Multi-branch designs also impose constraints that limit their adaptability and restrict the applicability as well as the performance of techniques such as channel pruning and filter pruning.

The motivation of the authors was to create an architecture that takes the training advantages given by multi-branch models but keeps the inference advantage of single-branch models on highly parallel processing units.

The trained **RepVGG block** sends its input to three branches, one performing a 3x3 convolution followed by batch normalisation, a second performing a 1x1 convolution followed by batch normalisation and a final branch which is just an identity operation followed by batch normalisation. The outputs of the three branches are then summed and passed through a ReLU activation function. If the block has a stride different than 1, then the identity branch is dropped.

Structural re-parametrization consists of a sequence of steps to convert a trained block into a single 3×3 convolution layer for inference. The steps are represented in Figure 3.7.

The first step is to incorporate for each branch the batch normalisation into the convolution parameters. The batch normalization of an input M at inference can be represented as (3.12).

$$bn(M, \mu, \sigma, \gamma, \beta)_{:,i,:,:} = (M_{:,i,:,:} - \mu_i)\frac{\gamma_i}{\sigma_i} + \beta_i \tag{3.12}$$

where μ, σ, γ and β are respectively the accumulated mean, standard deviation, learned scaling factor and bias of the batch normalisation layer.

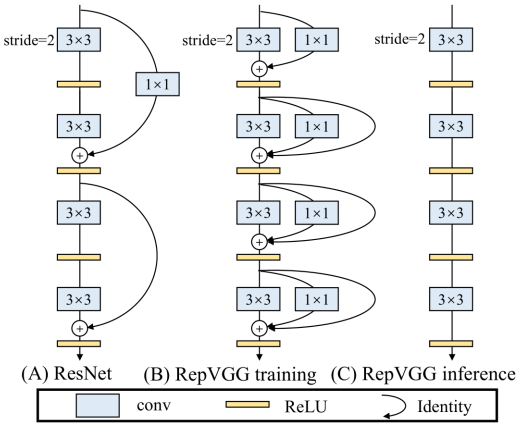


Figure 3.6: The architecture of RepVGG compared to ResNet [24]

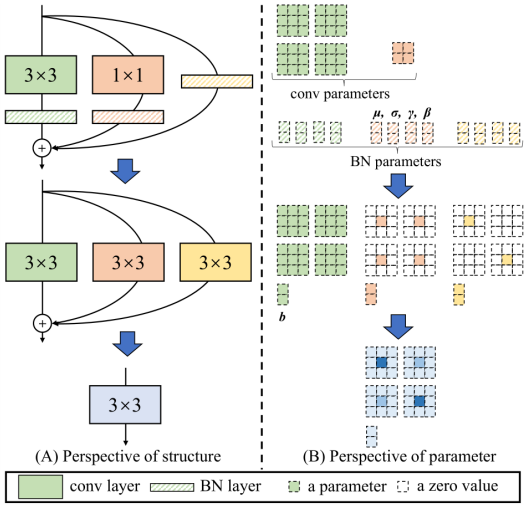


Figure 3.7: Structural re-parametrization [24]

When the batch normalization layer is preceded by a convolution layer with kernel W, they can be converted into a single convolution layer with kernel W' and a bias vector β'. The equation (3.13) describe how this conversion is done.

$$\begin{aligned} bn(M * W, \mu, \sigma, \gamma, \beta)_{:,i,:,:} &= ((M * W)_{:,i,:,:} - \mu_i)\frac{\gamma_i}{\sigma_i} + \beta_i \\ &= (M * W)_{:,i,:,:}\frac{\gamma_i}{\sigma_i} - \frac{\mu_i \gamma_i}{\sigma_i} + \beta_i \\ &= (M * W')_{:,i,:,:} + \beta'_i \end{aligned} \tag{3.13}$$

where $M * W$ is the convolution of input M with the kernel W.

We apply this transformation directly to the branch with the 3x3 convolution and the branch with the 1x1 convolution. For the identity branch, we first represent the identity operation as a 1x1 convolution with a fixed kernel and then apply the transformation to it and the branch's batch normalisation layer.

After merging the batch normalisation operation by transforming the kernels, the kernels are fused into one by first zero-padding the two 1×1 kernels into 3×3 kernels and then by summing all three kernels together. The result is a single 3×3 convolution layer that represents the three branches.

If the stride is not equal to 1, there is no identity branch but we can still apply the re-parametrization to the two branches left.

3.4 Traffic sign localisation

In this section, we introduce how to retrieve the geo-coordinates of traffic signs.

To estimate the coordinates of a detected traffic sign, we need two pieces of information, (1) the coordinates of the camera and (2) the position of the traffic sign relative to the camera.

3.4.1 Estimating device coordinates

A simple solution for estimating the current position of the device is to use the **last received GPS position** directly. However, the accuracy of GPS receivers can be affected by the environment, weather conditions and satellite configuration. In addition, the GPS signal may be lost for some period of time, making the use of raw GPS data unsuitable for our problem.

A reliable method of increasing positioning accuracy is to project the measured GPS coordinates onto the nearest road using a **vector map of the roads**. This method might be suitable if a map of road coordinates already existed but collecting this information for the map is very difficult and resource consuming, so it is not suitable for our purposes. We could use the Google Roads API, but the pay per query service they propose would increase the operational costs per device too much as the solution is supposed to run during a few hours almost every day.

An alternative to road vector maps is to combine GPS data with a model of the dynamics and any additional information, such as that provided by an IMU, using a **Kalman filter**. The Kalman filter has the advantages of increasing the GPS position accuracy, being able to handle short-term loss of the GPS signal and to filter out sudden "jumps" to a point far from the actual position.

In this work, we will estimate the coordinates of the camera by combining the GPS receiver's measurements about the current coordinates and the acceleration given by the IMU using a Kalman Filter. In the following sections, we will introduce how Kalman Filters work.

3.4.2 Estimating the relative position to the device

The relative position of an object relative to its position in the camera There exist multiple approach to measure the relative depth.

The first idea we had was to use the **standard dimensions** of the road signs as a reference to estimate the position. Others have also tried this approach with success [18]. However, in Belgium, the dimensions of traffic signs vary according to various rules having exceptions as mentioned above. Using this trick would require having only one calibrated camera, but it would result in inaccurate measurements in Belgium.

Another option used by [], would be to add a **lidar** next to our camera. Lidars allow 3D projections to be made by targeting an object or surface with a laser and measuring the time it takes for the reflected light to return to the receiver. Lidars are very accurate, but there are several drawbacks to this approach: lidar cameras are generally expensive and the alignment of lidar measurements with the camera image can be quite complex.

Finally, it is possible to know the relative position of an object by triangulating it using **stereoscopic vision**. This solution is an in between, it is more reliable than using reference dimensions and less expensive than lidar cameras. These authors also used the stereoscopic vision approach.

For our system, we will use stereoscopic vision. In the following sections, we will introduce the concept of stereoscopic vision.

3.5 Kalman Filters [32] [26] [25]

The Kalman filter is a recursive filter used to estimate the current state of a dynamic system based on incomplete sequences of noisy sensor measurements and using a linear model of the system dynamics. The state of the system is represented as a multivariate Gaussian distribution whose mean is the most likely state and covariance matrix is the uncertainty. The state transition model and the sensor model are assumed to be linear transformation with additive Gaussian noise.
The Kalman filter is called an optimal estimator because it minimizes the mean square error of the estimated parameters. In this sense, Kalman filter is the best unbiased linear estimator.

The Kalman filter algorithm consists of three stages:

1. **Initialization step:** Set the initial state and initial uncertainty about the system.
2. **Prediction step:** Project the current belief state forward from t to $t + 1$ through the transition model.

3. **Update step:** Update this new state using the new evidence.

After the initialization, we loop between step 2 and 3 every time new sensor measurements are available.

Before going further, it is interesting to define the variables we will encounter:

- State variable $\mathbf{x}_t$: It is a $n \times 1$ vector representing the state estimate of the system at time t.
- State covariance matrix $\mathbf{\Sigma_x}$: It is a $n \times n$ matrix representing the uncertainty about the current state.
- State transition model $\mathbf{F}$: It is a $n \times n$ matrix representing how states propagate with time.
- Evidence variable $\mathbf{e}_t$: It is a $m \times 1$ vector representing the sensor measurements at time t.
- Evidence covariance matrix $\mathbf{\Sigma_e}$: It is a $m \times m$ matrix representing the sensor measurements' noise.
- Sensor model $\mathbf{H}$: It is a $m \times n$ matrix representing the transformation from the state space to the sensor measurements space.
- Control vector $\mathbf{u}_t$: It is a $k \times 1$ vector representing a quantity from the outside world affecting the system at time t.
  Note that the control vector is not mandatory for the Kalman filter to work.
- Control transition matrix $\mathbf{B}$: It is a $n \times k$ matrix representing how the control vector affects the state of the system.
- State transition noise covariance matrix $\mathbf{\Sigma_u}$: It is a $n \times n$ matrix representing the uncertainty that arises from the influence of the outside world during a state transition.
- Kalman gain matrix $\mathbf{K}_{t+1}$: It is a $n \times m$ matrix representing the weight given to the new sensor measurements and current state estimate.

Note that although $\mathbf{\Sigma_e}$ and $\mathbf{\Sigma_u}$ are supposed to reflect the statistics of the noises, in many practical applications the true statistics of the noises is not known or not Gaussian. Therefore, $\mathbf{\Sigma_e}$ and $\mathbf{\Sigma_u}$ are usually used as tuning parameters for the user to adjust to get desired performance. [25]

3.5.1 Initialization step

Before launching the algorithm, we need to guess the initial state variable $\mathbf{x}_0$ and state covariance matrix $\mathbf{\Sigma}_\mathbf{x}$.

3.5.2 Prediction step

The prediction step, also called the propagation step, estimates the propagation of the current state to the next time-step based on its transition model.
The prediction step is described by the following set of equations:

$$\mathbf{x}_{t+1}^{-} = \mathbf{F}\mathbf{x}_t^{+} + \mathbf{B}\mathbf{u}_t \tag{3.14}$$
$$\mathbf{\Sigma}_\mathbf{x}^{-} = \mathbf{F}\mathbf{\Sigma}_\mathbf{x}^{+}\mathbf{F}^T + \mathbf{\Sigma}_\mathbf{u} \tag{3.15}$$

In the above equations, the superscripts $-$ and $+$ denote respectively prior(predicted) and posterior(updated) estimates.

Equation (3.14) is used to compute the new predicted state estimate by using the state transition model on the previous state estimate through $\mathbf{F}\mathbf{x}_t^{+}$ and then by including any measurable external influence on the system through $\mathbf{B}\mathbf{u}_t$. Equation (3.15) computes the state covariance matrix showing an increase in uncertainty about the state of the system due to the state transition noise.

3.5.3 Update step

The update step, also called the correction step, corrects the predicted new current state using the noisy measurements from the sensors.
The update step is described by the following set of equations:

$$\mathbf{y}_{t+1} = \mathbf{e}_{t+1} - \mathbf{H}\mathbf{x}_{t+1}^{-} \tag{3.16}$$
$$\mathbf{K}_{t+1} = \mathbf{\Sigma}_\mathbf{x}^{-}\mathbf{H}^T(\mathbf{\Sigma}_\mathbf{e} + \mathbf{H}\mathbf{\Sigma}_\mathbf{x}^{-}\mathbf{H}^T)^{-1} \tag{3.17}$$
$$\mathbf{x}_{t+1}^{+} = \mathbf{x}_{t+1}^{-} + \mathbf{K}_{t+1}\mathbf{y}_{t+1} \tag{3.18}$$
$$\mathbf{\Sigma}_\mathbf{x}^{+} = (\mathbf{I} - \mathbf{K}_{t+1}\mathbf{H})\mathbf{\Sigma}_\mathbf{x}^{-} \tag{3.19}$$

Equation (3.16) represents the error in the predicted observation. Equation (3.17) computes the Kalman gain. Equation (3.18) updates the new state prediction by

incorporating the discrepancy between it and the measurements. Finally, equation (3.19) updates the state covariance matrix by reducing uncertainty about the state of the system from the incorporation of the measurements.

3.6 Stereoscopic Vision

3.6.1 Camera model [5] [37]

In this section, we will present the mathematical model of a camera.

The simplest way of representing a camera is through the pinhole camera model which mathematically describes the projection of a 3D object onto a 2D image, as illustrated in Figure 3.8. In this model, all the rays converge to a unique point, the camera center, before arriving on the image plane.

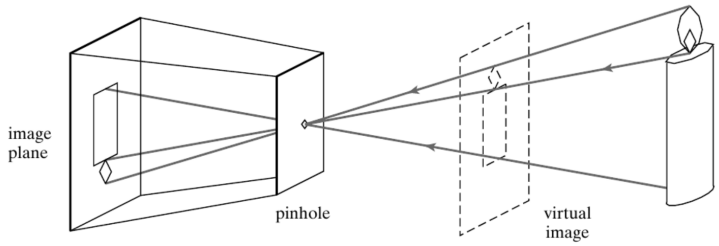


Figure 3.8: Pinhole camera [5]

The projection of a 3D point $P = (X, Y, Z)$ to its pixel coordinate $p = (x, y)$ on a non-inverted image plane as illustrated in Figure 3.8 can be expressed in a matrix form as the following transformation:

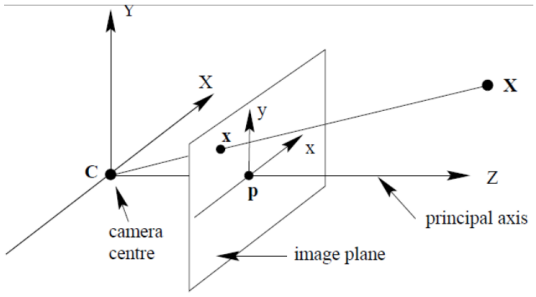


Figure 3.9: Central-projection model [5]

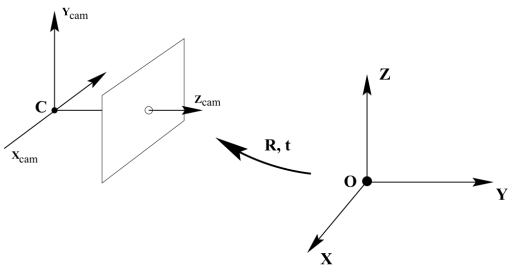


Figure 3.10: Camera coordinate system and arbitrary world coordinate system [5]

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} [\mathbf{I}_{3\times3}|\mathbf{0}_{3\times1}] \begin{bmatrix} \underline{R}^T & -\underline{R}^T t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.20}$$

$$= \mathbf{K}\mathbf{R}^T[\mathbf{I}|-t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.21}$$

$$= \mathbf{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.22}$$

The matrix $\mathbf{K}$ in Equation (3.21), called the matrix of intrinsic parameters or calibration matrix, is a transformation matrix that converts points from the camera coordinate system to the non-inverted image plane coordinate system as represented by Figure 3.9. The calibration matrix is composed of **5 intrinsic parameters** and every camera possess its own fixed set of unique intrinsic parameters. They depend on camera properties (such as central point offset, aspect ratio and skew) and can be given by the camera manufacturer or obtained through a calibration process using a known template of patterns.

The matrix $\mathbf{R}^T[\mathbf{I}|-t]$ in Equation (3.21), called the matrix of extrinsic parameters, is a transformation matrix from an arbitrary world coordinate system to the camera coordinate system as represented by Figure 3.10. Indeed, this matrix is

composed of a combination of a rotation matrix **R** having 3 degrees of freedom to describe it and a translation vector t also having 3 degrees of freedom representing together a total of **6 extrinsic parameters**, or degrees of freedom. They depend on the position and orientation of the camera.

The relation presented in (3.22), when intrinsic and extrinsic parameters are known, proposes for a particular 3D point a unique solution in the 2D image plane, however **for the inverse transformation to have a unique solution we will require additional information, the depth** λ. Indeed, without λ, the projection of a 2D point in the image plane to 3D world coordinates has an infinite number of possible solution lying alongside a straight line.

Real cameras do not work exactly as the pinhole model, the light rays pass through a lens before reaching the image plane which introduces radial distortion and a little bit of tangential distortion. To take it into account, the previous model can be extended by introducing **radial and tangential distortion coefficients**.

3.6.2 Binocular vision [5] [37] [39] [9]

In this section, we will briefly describe how to retrieve depth from two aligned cameras.

The idea of using two aligned cameras to calculate depth finds its roots in biology. We as humans use the same principle to perceive depth with our eyes. When we look at close objects with one eye, we see a difference between the two perspectives, but when we look at something far away, we see no difference. Our brain automatically processes these differences to enable us to perceive the distance of each object relative to us.

Let us consider two cameras with no distortion and perfectly horizontally aligned, then for a particular 3D object, the horizontal displacement between the coordinates of the projection of this object in each camera's image plane is inversely proportional to the depth as illustrated in Figure 3.11. This displacement is also referred to as **disparity**. When we compute the disparity between every pixels of both images, we obtain a disparity map.

Modeling our simple stereo system can be done using the pinhole camera model to represent both cameras. Figure 3.12 illustrates the **stereo camera model**, where c and c' are the camera's optical center, f is the focal length of both cameras, B is the distance separating the cameras, called the baseline. The points x and x' are the projection of a particular 3D point P in each camera's image plane.

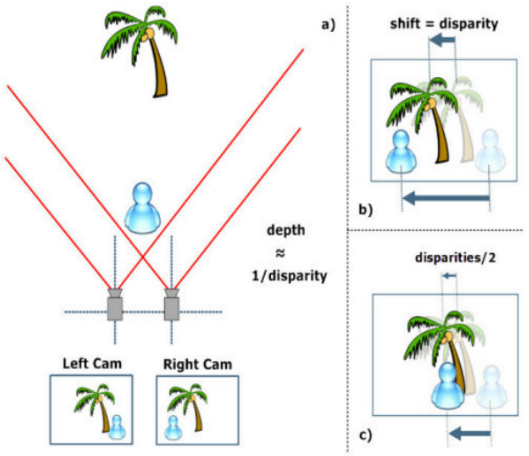


Figure 3.11: Stereo vision system and a visualization of the notion of disparity [5]

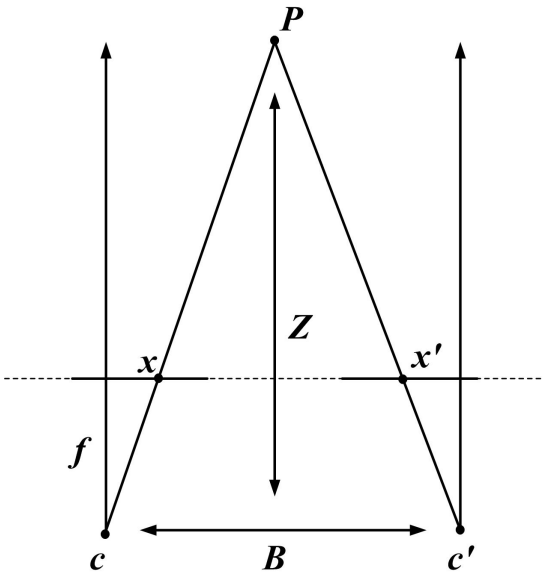


Figure 3.12: Stereo camera model

To determine depth Z of the point P from the stereo model, we will use **triangulation**. Using similar triangles technique with Pcc' and Pxx' we obtain the following relation:

$$\frac{B}{Z} = \frac{(B+x)-x'}{Z-f} \iff Z = \frac{Bf}{x'-x} \tag{3.23}$$

Where $x - x'$ is the disparity.

After getting the depth Z, we still need to find the transformation to get the relative position of the sign compared to the camera. A simple geometrical transformation can be used to calculate it, as the angles are known or can be calculated with respect to the image plane.

From here, the only problem left is that of **stereo matching**, in other words, finding corresponding pixels in a stereo pair of images. The basic approach to stereo matching is to select a window in the reference image and to search alongside the corresponding horizontal row in the target image for the position with the highest similarity. Similarity can be expressed by the use of similarity/dissimilarity measures, where one of the most interesting is the Normalized Cross-Correlation that is based on the Pearson correlation coefficient.

In the real world, it is impossible to align both cameras perfectly and the cameras have a lens that creates distortions. For the stereo matching process to work, we need to remove the distortions in the images and to virtually align them. To remove lens distortion, all we have to do to un-distort the image is to derive the distortion coefficients through a calibration process. Then to virtually align the cameras onto a common image plane, we will use a **stereo rectification** process which also requires to know the transformation to pass from the projection of a 3D point in one camera's plane to the other's, expressed by the **essential matrix**. The steps for the cameras to be distorted and aligned are illustrated in Figure 3.14.

The first step we have to take is **stereo calibration**, as it allows us to find the intrinsic parameters, the distortion coefficients and the essential matrix. Stereo calibration is done by presenting to the cameras simultaneously a known pattern under different orientations. We will use a ChArUco board as pattern, which is a chessboard with ArUco markers embedded inside the white squares as can be seen in Figure 3.13.

An AruCo marker is a square synthetic marker with a large black border and an internal binary matrix that identifies it. They are fast to detect, can be identified and allow for the application of error detection and correction, but their corners are hard to detect. Chessboard patterns can be refined more accurately but they are required to be completely visible, no occlusions are allowed. ChArUco boards rely on ArUco markers to interpolate the position of the chessboard's corners. This property allows the chessboard not to be always completely visible. Furthermore, as the interpolated corners belong to a chessboard pattern, they have a high sub-pixel accuracy.

The limitations of stereo vision appear when parts of some object is only visible in one of the images or when encountering objects that have no texture, have repetitive textures, are transparent or that have specular surfaces, as all these situation make the stereo matching process harder.
Another different type of limitation arise from the discretization of the image plane into a grid of pixels of fixed resolution, as it limits the maximum depth that we can measure. The maximum perceivable depth can be obtained by equation $D_{max} = \frac{B}{2} tan((90 - \frac{FOV_H}{H_{Pixels}})\frac{\pi}{180})$ deduced from the stereo camera model, where D_{max} is the maximum perceivable depth, FOV_H is the horizontal field of view of the camera and H_{Pixels} is the number of horizontal pixels.

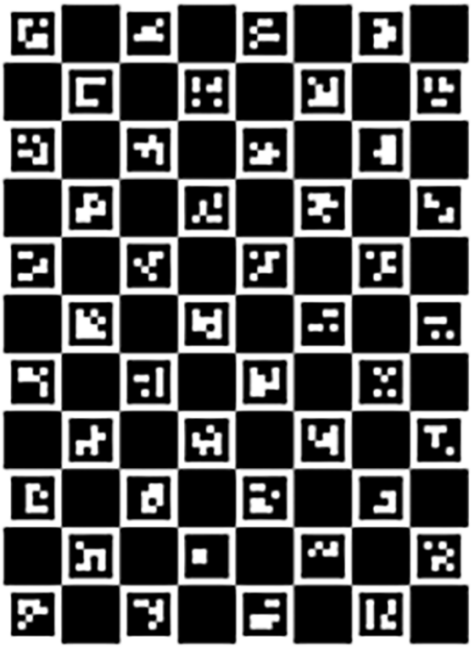


Figure 3.13: ChArUco calibration board

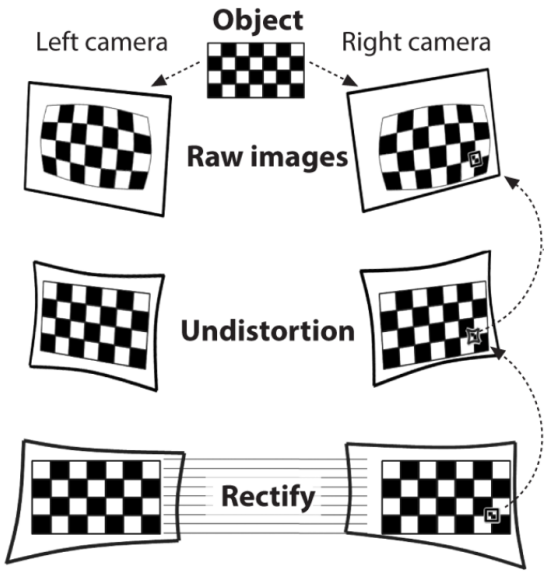


Figure 3.14: Stereo calibration and stereo images rectification [30]

Chapter 4

Approach

In this chapter, we will describe in more detail the methods used by our embedded system.

Building a traffic sign inventory system requires several steps. Figure 4.1 illustrates the processing pipeline of our system.

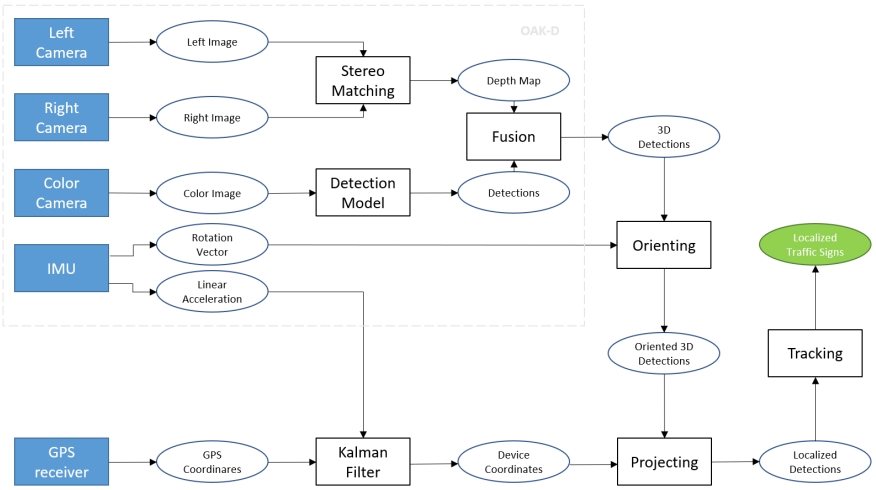


Figure 4.1: Structure of the traffic sign inventory system

4.1 Traffic Sign Detection model

As explained earlier, we based our detection model on the YOLO architecture. In this section we present our final model, but we will evaluate the different design choices that have lead us to it in the next chapter.

Our model takes 1280x704p RGB images as input. The backbone is composed of 14 repVGG blocks which will reduce the spatial dimensions by a factor 64. The neck will have a SPP block and use a PANet design at four scales. The four heads will be single convolution blocks. We will use LeakyReLU for all activation function.

For the loss function, we will use the standard YOLOv4 loss with CIoU for the localization loss and binary cross entropy for the confidence loss and classification loss.

For training, we will use the following data augmentations: Mosaic data augmentation, rotations, perspective changes, random flips (p=0.5), median blur (p=0.1) and random contrast changes (p=0.1).

We will use DIoU-NMS to filter predicted bounding boxes.

4.2 Localisation

4.2.1 Estimation of traffic sign position in the camera's coordinate system

After the object detection algorithm has detected the traffic signs inside of the current frame, we have to retrieve the 3D coordinates for each sign relative to the camera.

To obtain the 3D coordinates of a particular sign, we need to know the depth and the camera parameters. Since we have a stereo camera system, we only need to calibrate it to obtain these two parameters, as explained in the theoretical background. By performing a stereo calibration with the two monochrome cameras and the color camera, we obtain the correspondence between the bounding boxes and the depth map.

To not corrupt our results due to noise, we take the median depth of the 5 by 5 window at the center of the traffic sign's bounding box in the depth map. By

knowing the depth to the center of the traffic sign and the cameras' intrinsic parameters, we can easily compute the 3D position of the sign as said in the theoretical background.

To perform stereo matching we have the choice between two resolutions: 800x1280p and 400x640p. Their theoretical maximum perceivable depth is 38.25 meters and 19.12 meters respectively. Minimum depth is set to 50cm and maximum depth to 12 meters

After stereo calibration, we obtained an epipolar error of 0.13547.

4.2.2 Determining the device's orientation

The device's orientation is an important information to get if we want to correctly project the position of the traffic signs we detected onto a map.

The IMU inside of the stereo camera proposes a built-in ROTATION_VECTOR function based on the Madgwick filter [34]. The Madgwick filter is a method designed for low computational power systems, which uses the accelerometer, gyroscope and possibly magnetometer readings as inputs to produce a quaternion describing its orientation in space.

Quaternions are an extension of complex numbers having three imaginary dimensions instead of one and they are commonly used for representing orientations and rotations. A quaternion $Q = (q_0, q_1, q_2, q_3) = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$ where q_0 is a scalar value representing an angle of rotation, q_1, q_2, and q_3 correspond to an axis of rotation about which the angle of rotation is performed [3]. If you want to obtain a rotation matrix $\mathbf{R}$ from a quaternion $Q = (q_0, q_1, q_2, q_3)$, you can use the following:

$$\mathbf{R}(Q) = \begin{bmatrix} 2(q_1^2 + q_1^2) - 1 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & 2(q_0^2 + q_2^2) - 1 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & 2(q_0^2 + q_3^2) - 1 \end{bmatrix}$$

Otherwise, to directly apply a rotation on a vector v using a quaternion Q, we transform the vector into a quaternion $v = (0, x, y, z)$, use the quaternion Q and its conjugate $Q^* = (q_0, -q_1, -q_2, -q_3)$ to perform the operation $V = q.v.q^*$ where $V = (r, X, Y, Z) = (X, Y, Z)$

4.2.3 Estimating the device's position

Estimating the current position of the device is one of the most essential parts of the project, because even if we manage to detect traffic signs and assess their conditions perfectly, we cannot create an inventory if the device's position is inaccurate. Indeed, the position accuracy of the detected traffic signs is directly dependent on the position accuracy of the device.

The camera's IMU posses another built-in function, LINEAR_ACCELERATION, providing the linear acceleration of the device excluding gravity. Before using it with the Kalman Filter, the linear acceleration has to be oriented into the earth's coordinate system using the quaternion obtained previously and its unit should be converted from meters per second to degrees per second.

For this work, we will only consider the position and velocity alongside two axis, the latitude and the longitude. The state of our Kalman filter is then composed of 4 variables:

$$\mathbf{x}_t = \begin{bmatrix} x \\ y \\ x' \\ y' \end{bmatrix}$$

where x and y represent the latitude and longitude respectively, and x' and y' represent the velocities alongside the latitude and longitude respectively.

For the **initialisation step**, we set the filter by using as initial position the first GPS coordinates received (x_0, y_0) and assuming that the device is at rest.
The initial state $\mathbf{x}_0$ and state covariance matrix $\mathbf{\Sigma}_{\mathbf{x}0}$ are expressed as follow:

$$\mathbf{x}_0 = \begin{bmatrix} x_0 \\ y_0 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{\Sigma}_{\mathbf{x}0} = \begin{bmatrix} \sigma^2_{\text{GPS}} & 0 & 0 & 0 \\ 0 & \sigma^2_{\text{GPS}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where σ_{GPS} is given by the GPS manufacturer.

For the **prediction step**, our dynamic model is based on the law of uniformly accelerated motion, which can be expressed in matrix form using the state transition matrix and the control transition matrix where the control vector $\mathbf{u}_t = \begin{bmatrix} x'' \\ y'' \end{bmatrix}$ represents the linear acceleration measured by the IMU.

The state transition matrix $\mathbf{F}$ and the control transition matrix $\mathbf{B}$ are expressed as follows:

$$\mathbf{F}(dt) = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{B}(dt) = \begin{bmatrix} \frac{dt^2}{2} & 0 \\ 0 & \frac{dt^2}{2} \\ dt & 0 \\ 0 & dt \end{bmatrix}$$

where dt is the time that has passed between the new state from the previous.

To form the state transition noise covariance matrix $\mathbf{\Sigma_u}$, we measure standard deviation σ_{acc} of the linear acceleration when no force is applied on the IMU. An assumption about the error of the accelerometer that we make is that the error along the x axis does not affect the error along the y axis. We can then derive from σ_{acc} quantities for the position $\sigma_{\text{pos}} = \frac{\sigma_{\text{acc}} dt^2}{2}$ and the velocity $\sigma_{\text{vel}} = \sigma_{\text{acc}} dt$. Then the state transition noise covariance matrix $\mathbf{\Sigma_u}$ is represented as:

$$\mathbf{\Sigma_u}(\sigma_{\text{acc}}, dt) = \begin{bmatrix} \sigma_{pos}^2 & 0 & \sigma_{pos}\sigma_{vel} & 0 \\ 0 & \sigma_{pos}^2 & 0 & \sigma_{pos}\sigma_{vel} \\ 0 & 0 & \sigma_{vel}^2 & 0 \\ 0 & 0 & 0 & \sigma_{vel}^2 \end{bmatrix}$$

During the **update step**, we will use as evidence $\mathbf{e}_t$ the latitude and longitude from the raw GPS data. The GPS latitude and longitude being a direct measurement of part of our state variables, the GPS sensor model $\mathbf{H}_{\text{GPS}}$ is simply:

$$\mathbf{H}_{\text{GPS}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

The GPS sensor covariance matrix $\mathbf{\Sigma}_{e,\text{GPS}}$ is given by the GPS manufacturer.

$$\mathbf{\Sigma}_{e,\text{GPS}} = \begin{bmatrix} \sigma_{\text{GPS}}^2 & 0 \\ 0 & \sigma_{\text{GPS}}^2 \end{bmatrix}$$

where $\sigma_{\text{GPS}} = 1.75$ meters for our receiver.

We also had the idea to measure the vehicle speed using our tracking system. Since traffic signs are static objects, the difference in relative position with respect to the camera between two consecutive detections of the same sign can be used to calculate a displacement vector. Then, if we include the time between the two detections with the displacement vector, we can compute a velocity vector. This

velocity vector can be oriented with respect to longitude and latitude using the rotation vector presented earlier.
We tried it out and had good preliminary results but some more fine tuning is still necessary before integrating it in the Kalman filter.

4.2.4 From the camera to the world's coordinate system

Now that the coordinates of the traffic signs are known in the camera's field of reference, we need to project these coordinates to the world's coordinate system. For this purpose we need the two elements composing the extrinsic matrix, a rotation matrix $\mathbf{R}$ and a translation vector t. We already established earlier how to recover the rotation matrix using the camera's IMU.

After orienting the detected traffic signs along the axes of the world's coordinate system using the rotation vector, all we need is the translation vector. The translation vector in our case is simply the vehicle's current position estimated by the Kalman filter. However, we are not translating the traffic sign position in an Euclidean space but on the surface of the earth. The task requires an understanding of geodesic on an ellipsoid of revolution.

The shortest path between two points on the earth, usually treated as an ellipsoid of revolution, is called a geodesic. The direct problem consists of finding the end point of a geodesic from its starting point, its initial azimuth and its length. The inverse problem consists of finding the shortest path between two given points. Any geodesic problem is equivalent to solving the geodesic triangle, given two sides and their included angle (the azimuth at the first point) in the case of a direct problem, and the difference in longitude in the case of an inverse problem. In the context of this thesis, we used Karney's implementation. [31]

4.3 Tracking algorithm

To track detected signs over time, we decided to use a very simple and fast tracking method based on the IoU between new and previous detections. We assume that the latency of the model is low enough that a sign detected in two consecutive frames has not moved much and therefore it is possible to link the detections using their IoU score. Our tracking method can handle signs that disappear for a frame or two and allows us to filter out some false positives.

The tracking algorithm uses an user defined threshold and objects we call tracklets. A tracklet represent the detections for a same object. It is composed of an id, a state boolean that can be active or inactive, a persitence score, bounding box coordinates, and a list containing the data of all the previous detections that compose it (class, confidence score, 3D coordinates, ...).

Every time a new frame is available, we compute the IoU score between the set of detections for this frame and the set of tracklets. The detection having the highest IoU score with a particular tracklet is considered as corresponding to the tracklet. There are three possible cases:

- A detection finds no corresponding tracklet: The detection becomes a new tracklet whith no id, an inactive state and a persistence score of 0.
- A tracklet finds no corresponding detection: The tracklet's state becomes unactive and its persistence score is decreased by 1. If its persistence score becomes smaller than 0, then the tracklet is killed.
- A tracklet finds a corresponding detection: The detection data is integrated to the tracklet, the new bounding box of the tracklet is the bounding box of the detection and the persistence score is increased by 1.
  If the tracklet has an id, its state is set to active and the confidence score is set to the threshold.
  Else if the tracklet has no id and the persistence score reaches the threshold, an id is attributed to the tracklet and its state becomes active.
  Otherwise, nothing more is done.

When a tracklet is killed, it is removed from the set and its detection history is merged to improve the localization and identification of the sign. The sign class is considered to be the one which had the highest confidence score in the detection history and the coordinates of the sign are considered to be the median of the coordinates from the detection history.

In practice, our tracking solution works well when the vehicle is going straight forward, but it has more difficulty when it makes sharp turns. Also, if a traffic sign is obstructed for too many frames, tracking is lost and when the sign reappears, it is treated as a new sign.

Chapter 5

Experiments and Results

5.1 Dataset

In this section, we explain the traffic sign classes we use and how we collected the data used to train, validate and test our model.

We created 13 classes based on the shape and colors of traffic signs. We have made this choice because these features indicate the type of message a sign conveys. The classes and the traffic sign they represent can be seen in Appendix A.

Our data comes from three different sources. One part comes from a public database, another one was synthetically generated and the last part was collected manually. The class distribution in the training set is illustrated by Figure 5.1.

5.1.1 Public Data

Several public datasets collected in various countries are available for traffic signs detection and classification. They cover different type of signs and come in different size. For this work we use the Belgium Traffic Sign Detection Dataset (BTSD) [40]. It is composed of 9006 annotated images extracted from four video sequences captured in Belgium. Out of the 9006 images, 3101 are in the test set and 5905 in the training set. They contain annotations for 210 different classes.

We have kept only the signs that are part of our distribution. We keep the BTSD training set in our training data, but the test set is part of our validation data.

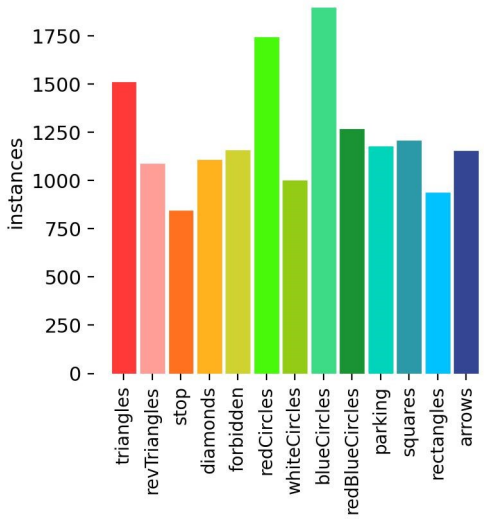


Figure 5.1: Training set class distribution

While exploring the data, we observed that multiple signs where not annotated. Example of missing annotation can be seen in Figure 5.2. Because missing annotations could hinder the learning and evaluation of our model, we reviewed the 9006 images of the dataset and added the missing annotations. We have added more than a thousand annotations in total.

Figure 5.2: Examples of image with missing annotation in the BTSD dataset.

5.1.2 Synthetic Data

In [16], the authors propose a method to generate synthetic traffic sign data using arbitrary natural images as background and templates of traffic signs. Augmenting the dataset with synthetic data has the advantage of drastically reducing the cost of collecting annotated training data. It can also be used as a mean to alleviate the class imbalance problem.

The algorithm begins by selecting one to six traffic sign templates and an image from the COCO dataset [23] to be used as background. The background image X has then its brightness and contrast changed by $\alpha X + \beta$ where α and β are random values. Then each template is processed and blended to the background using the steps illustrated in Figure 5.3:

1. the template is multiplied with the same α used on the background image
2. we apply geometric transformation to the template in the form of a 3D rotation and scaling
3. we adjust the brightness by adding the average brightness of the region on which the template will be added minus a constant
4. we add noise
5. we place the template in a random position with no intersection with other template
6. we fade the borders of the template for a smooth transition to the background

Finally a Gaussian blur σ is added to the resulting image.

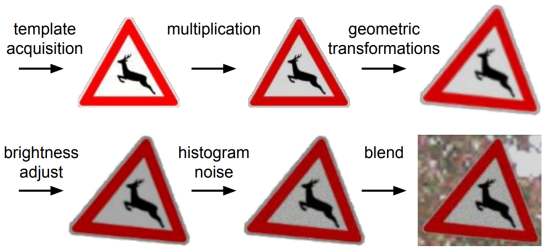


Figure 5.3: Steps of the template blending process [16]

They showed a 4.14 p.p. mAP improvement on BTSD by training a Faster R-CNN model on the real target images and the synthetic data.

We improved the proposed method by adding synthetic degradation to the traffic sign templates such as color fading and partial occlusions. For the color fading degradation, we convert the model to HSV color space, select pixels whose hue is

within a defined range (red, blue or yellow) and reduce the saturation and value of these pixels by a random amount. The partial occlusion degradation is done by placing a randomly colored text or rectangle on the template. An example of the images generated by our method can be seen in Figure 5.4.

Figure 5.4: Example of image generated using our modified method

We generated 6000 images that we incorporated in the training set.

5.1.3 Collected Data

A last part of the data was collected using the OAK-D camera in Belgium on highways, urban and rural areas. The images where saved at a frequency of 6 Hz.

From all the saved images, 643 images where manually selected and annotated for the purpose of being used as test set to evaluate our model.

Another 310 images where also manually selected and annotated to join the validation set.

Finally, 1132 images were selected using our traffic sign detection model trained on the BTSD and synthetic data. They were chosen because they contained detections with a confidence score between 0.4 and 0.6. We use this selection criterion in order to obtain the images for which the detection model is uncertain because these are the ones that provide the most interesting training information. These annotated

images will be used in our training set along with the BTSD and the synthetic datasets.

We took a particular attention to ensure that test, validation and training data where collected from different locations.

5.2 Traffic Sign Detection

In this section, we will evaluate the influence of different changes to the architecture of our model.

We perform all latency measurements with the stereo matching process in parallel. This ensures that we have measurements that are representative of real-world performance. The stereo matching process is done with a resolution of 800x1280p.

5.2.1 Impact of synthetic data

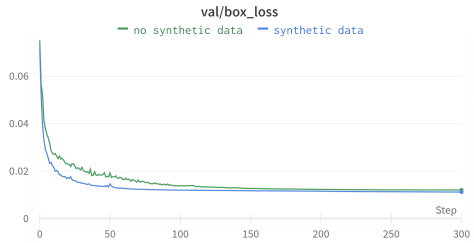
A first element to evaluate is the relevance of using synthetic data during training.

	Precision	Recall	F1-score
No synthetic	87.75	88.86	88.30
synthetic	93.25	91.56	**92.40**

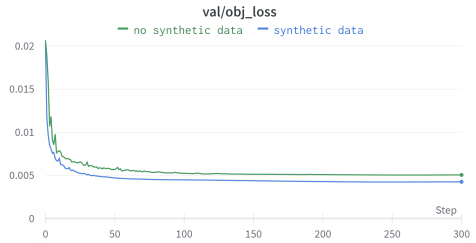
Table 5.1

From Figure 5.5, we can observe that training with synthetic data makes the model converge faster and lowers the losses.

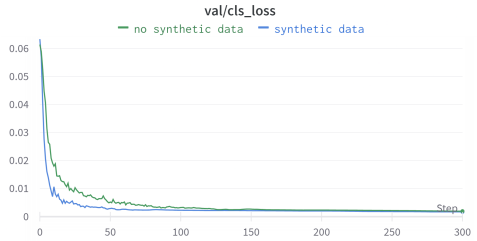
In continuity, in Table 5.1 we can observe that adding synthetic data improves the detection. Those results led us to use synthetic data for the following comparisons.



(a) Localization loss



(b) Object loss



(c) Classification loss

Figure 5.5: Localization loss, object loss and classification loss of the validation set for our model trained during 300 epochs with and without synthetic data.

5.2.2 Comparing backbones

Here, we will consider the RepVGG14, CSPDarknet14 and MobileNetV3-small backbones. In this comparison we are training the models during 300 epochs on 640p images. The RepVGG14 present in this comparison has twice the number of channels than our final solution. CSPDarknet14 is a scaled down version of CSPDarknet53 with 14 layers. The structure of CSPDarknet14 is similar to that of repVGG14.

We can state that the RepVGG14 and the CSPDarknet14 almost have the same scores, but the former is much faster. As for the MobileNetV3-small, it does not even compete with the two other, whether for the score or the latency. Therefore, we chose the RepVGG14 as backbone for our model.

	Precision	Recall	F1-score	Latency
RepVGG-14	89.18	89.59	89.38	**25.0 ms**
CSPDarknet-14	89.23	89.99	**89.61**	42.5 ms
MobileNetV3-s	85.73	83.94	84.83	53.3 ms

5.2.3 Comparing resolutions

We wanted to test the effect of resolution variations on our model.

Here the model used for the 640x352 and 1280x704 resolution is the same repVGG14 model used in the previous comparison. The model identified by 1280x704 - C/2 is our final model.

Using a 640p resolution provides us with a very low latency, but we can notice that by increasing the resolution, the performances are also enhanced, even if it is to the detriment of the latency. When we divide the number of channel by 2, the results are still better if we use a higher resolution.
Thereby, the highest resolution appears to be the best policy.

	Precision	Recall	F1-score	Latency
640x352	89.18	89.59	89.38	**25.0 ms**
1280x704	93.24	93.27	**93.25**	126.7 ms
1280x704 - C/2	93.25	91.56	92.40	44.4 ms

5.2.4 Comparing necks

We want to compare PANet and FPN on our final model.

The comparison between the PANet and the FPN necks shows us that, even if the latency does not vary between the two, the PANet gives us better detections, and is thus the one we chose for our model.

	Precision	Recall	F1-score	Latency
PANet	93.25	91.56	**92.40**	44.4 ms
FPN	92.56	91.35	91.95	**44.0 ms**

5.2.5 Comparing activation function

Here we are comparing the changes in latency, precision and recall happening when we change the activation function. Whether for ReLU or for LeakyReLU, the runtime does not change. However, the LeakyReLU seems to be more proned to learn, and then the best option.

	Precision	Recall	F1-score	Latency
ReLU	92.73	90.69	91.69	**44.3 ms**
LeakyReLU	93.25	91.56	**92.40**	44.4 ms

5.2.6 Performances on the test set

The following tabular shows the performance of our method on the test set. We notice that the results have decreased compared to the validation set. This is probably due to the lack of images collected with our camera in the training and in the validation set, compared to the test set.

Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95
all	643	911	0.848	0.864	0.909	0.706
triangles	643	157	0.872	0.954	0.97	0.78
revTriangles	643	56	0.879	0.905	0.961	0.721
stop	643	11	0.884	0.696	0.787	0.589
diamonds	643	43	0.757	0.907	0.943	0.7
forbidden	643	49	0.885	0.946	0.961	0.739
redCircles	643	158	0.908	0.872	0.953	0.774
whiteCircles	643	25	0.87	0.804	0.939	0.731
blueCircles	643	124	0.855	0.935	0.962	0.787
redBlueCircles	643	69	0.971	0.964	0.991	0.784
parking	643	43	0.741	0.791	0.839	0.648
squares	643	83	0.948	0.885	0.977	0.786
rectangles	643	16	0.64	0.688	0.612	0.519
arrows	643	77	0.817	0.883	0.919	0.627

In Figure 5.6, we can state that we have a huge loss in the rectangles detection. Actually, the three classes that are the least well detected are rectangles, parkings and arrows. It makes sense when we notice that, in our test set, the signs belonging to these classes are the most damaged/occluded of all.

From Figure 5.7, we observe that our model is susceptible to confuse background objects with real traffic signs.

In Figure 5.8 we see that our model is able to detect deteriorated signs and signs far in the distance.

More detection examples on the test set can be found in the Appendix B.

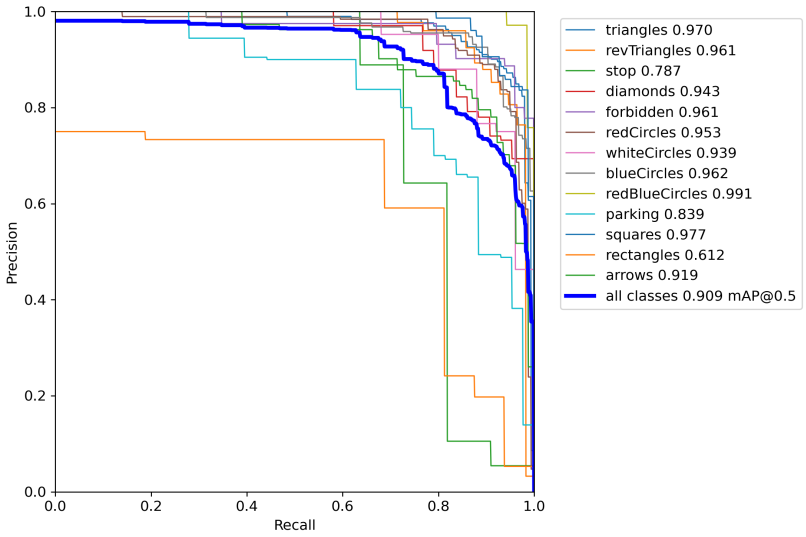


Figure 5.6: Precision-Recall Curve from the test set

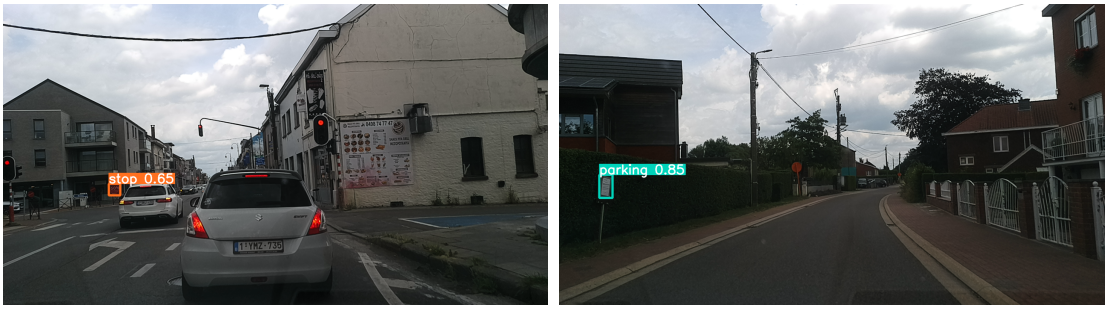


Figure 5.7: Examples of bad detections.

Figure 5.8: Examples of good detections.

5.3 Traffic sign localisation

5.3.1 Depth accuracy

Our experimental setup for evaluating the accuracy of stereoscopic depth estimation is as follows. We present a traffic sign at progressively increasing distances. We compare the readings from a laser measuring device to the Euclidean distance to the sign measured by our calibrated camera. We used our traffic sign detection model to detect the sign up to 6.13 meters where it lost track. After this distance we manually selected the position of the sign in the depth map to get our measurement. We used the 1280x800p resolution for stereo matching.

Laser measurement (m)	Stereo measurement (m)	Error (cm)
1,12	1,13	1,12
2,03	2,06	3,045
3,15	3,22	7,245
4,2	4,28	8,4
5,15	5,28	12,875
6,13	6,32	19,616
7,04	7,34	30,272
8,11	8,49	38,117
9,04	9,52	47,912
10,35	11,01	66,24

Table 5.2: Laser measurements, stereo measurement and the absolute difference between the two measurements

Table 5.2 shows the error computed as the absolute difference between the laser and camera measurements for increasing distances. We can see that the measurement error increases with distance. These results show that results from that the when computing the position of a traffic sign,

5.3.2 Traffic sign position accuracy

To evaluate the traffic sign localization accuracy, we drive with our embedded system on the road and compare the found traffic sign coordinates to the real coordinates that we measured manually. The "real" coordinates were measured

by putting our gps receiver next to the traffic sign and taking the median value after at least 2 minutes.

We found that our system gave the traffic sign coordinates with a precision of 0.53 meters on average and a standard deviation of 0.21 meters.

Chapter 6

Conclusion

6.1 Achievements

We presented a real-time embedded traffic sign identification solution implementing hardware and software, which is composed of several steps : detection, classification and localization of traffic signs. The object detection algorithm was based on YOLO architecture trained on real and synthetic data. Our model achieves a frame rate of more than 20 frames per seconds while presenting good detection performances.

We encountered some problems while implementing our solution. A first problem we encountered was the lack of well annotated data but we solved it by creating our own dataset, generating synthetic data and by completing the existing dataset. We also had to deal with the localization problem. Indeed, the GPS-data can be noisy, and they do not inform us on the traffic sign position but on the device's position. We used a Kalman's filter to enhance the GPS precision, and we combined it with stereoscopic vision to locate traffic signs precisely.

With the configuration decided after our different comparisons, we obtained results for the detection that were good because our model can deal with challenging condition signs but the localization method still needs to be improved.

6.2 Future Works

Our current solution, even if it provides good results, can be improved, as any method could be. While working on this project, we sometimes ran out of time and thus we did not dig enough in some directions. Some aspects of the problem need to be further explored, such as :

- The data-set size : the data-set used in this project did not provide enough images captured with the camera to correctly train the model. With more data, our model could gain in accuracy.
- Add condition analysis : the main goal of the project is, in fine, to allow better road maintenance. Unfortunately, we did not have enough time to add a detection model for damaged traffic signs.
- Localization : it could be improved by switching from the GPS localization system to the European Union's Galileo that has an higher localization accuracy.
- Having some post-processing done on the different signed identified by our system would allow to filter out duplicate detection and send an alert if a sign has not been detected for some time.

Bibliography

[1] Openvino: Vpu devices, Last time accessed: 05/07/2022. https://docs.openvino.ai/latest/openvino_docs_OV_UG_supported_plugins_VPU.html#doxid-openvino-docs-o-v-u-g-supported-plugins-v-p-u.

[2] Ntt website, Last time accessed: 10/06/2022. https://services.global.ntt/en-us/about-us.

[3] How to convert a quaternion to a rotation matrix, Last time accessed: 17/05/2022. https://automaticaddison.com/how-to-convert-a-quaternion-to-a-rotation-matrix/.

[4] B. Bousarhane. Vision-based traffic sign detection and recognition systems: Current trends and challenges. *Int. J. Data Analysis Techniques and Strategies*, 2021.

[5] M. Van Droogenbroeck. Computer vision (lecture notes), 2021-2022. https://hdl.handle.net/2268/184667.

[6] A. Bochkovskiy et al. Yolov4: Optimal speed and accuracy of object detection. 2020.

[7] C. Dewi et al. Yolo v4 for advanced traffic sign recognition with synthetic training data generated by various gan. 2021.

[8] C.-Y. Wang et al. Cspnet: A new backbone that can enhance learning capability of cnn. 2020.

[9] G. Hwan An et al. Charuco board-based omnidirectional camera calibration method. 2018.

[10] H. Zhang et al. Real-time detection method for small traffic signs based on yolov3. 2020.

[11] J. Greenhalgh et al. Real-time detection and recognition of road traffic signs. *IEEE Trans. Intell. Transp. Syst.*, 13:1498–1506, 2012.

[12] J. Redmon et al. You only look once: Unified, real-time object detection. 2015.

[13] J. Redmon et al. Yolo9000: Better, faster, stronger. 2016.

[14] J. Redmon et al. Yolov3: An incremental improvement. 2018.

[15] K. He et al. Spatial pyramid pooling in deep convolutional networks for visual recognition. 2014.

[16] L. Tabelini et al. Deep traffic sign detection and recognition without target domain real images. 2020.

[17] L. Wang et al. An improved light-weight traffic sign recognition algorithm based on yolov4-tiny. 2021.

[18] P. Hienonen et al. Towards condition analysis for machine vision based traffic sign inventory. 2017.

[19] Q.-C. Mao et al. Mini-yolov3: Real-time object detector for embedded applications. 2019.

[20] S. Liu et al. Path aggregation network for instance segmentation. 2018.

[21] S. Yun et al. Cutmix: Regularization strategy to train strong classifiers with localizable features. 2019.

[22] T.-Y. Lin et al. Feature pyramid networks for object detection. 2017.

[23] T.-Y. Lin et al. Microsoft coco: Common objects in context. 2017.

[24] X. Ding et al. Repvgg: Making vgg-style convnets great again. 2021.

[25] Y. Kim et H. Bang. *Introduction to Kalman Filter and Its Applications*. 2019.

[26] S. Russell et P. Norvig. *Artificial Intelligence: A Modern Approach, Third Edition*. Pearson Education, 2016.

[27] European Union Road Federation. Improved signage for better roads: An erf position paper towards improving traffic signs in european roads. 2018.

[28] P. Geurts and L. Wehenkel. Introduction to machine learning (lecture notes), 2020-2021. https://people.montefiore.uliege.be/lwh/AIA/.

[29] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. *International Joint Conference on Neural Networks*, (1288), 2013.

[30] A. Kaehler and G. Bradski. *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library.* O'Reilly Media Inc., 2016.

[31] C. Karney. Algorithms for geodesics. 2011.

[32] G. Louppe. Introduction to artificial intelligence (lecture notes), 2019-2020. https://github.com/glouppe/info8006-introduction-to-ai/tree/info8006-2019.

[33] G. Louppe. Deep leanring (lecture notes), 2020-2021. https://github.com/glouppe/info8010-deep-learning/tree/v4-info8010-2021.

[34] S. Madgwick. Ahrs algorithms and calibration solutions to facilitate new applications using low-cost mems. 2014.

[35] United Nations. "article 29": Convention on road signs and signals.

[36] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union. June 2019.

[37] K. Sadekar. Understanding lens distortion, 2020. https://learnopencv.com/understanding-lens-distortion/.

[38] StatBel. Fewer road traffic casualties in 2021 than in 2019, mainly at the beginning of the year, Last time accessed: 13/08/2022. https://statbel.fgov.be/en/themes/mobility/traffic/road-accidents.

[39] R. Szeliski. *Computer Vision: Algorithms and Applications, 2nd ed.* Springer, 2022.

[40] Radu Timofte, Karel Zimmermann, and Luc van Gool. Multi-view traffic sign detection, recognition, and 3d localisation. In *Ninth IEEE Computer Society Workshop on Application of Computer Vision*, pages 1–8, Snowbird, Utah, USA, December 2009.

[41] Safat B. Wali, Majid A. Abdullah, Mahammad A. Hannan, Aini Hussain, Salina A. Samad, Pin J. Ker, and Muhamad Bin Mansor. Vision-based traffic sign detection and recognition systems: Current trends and challenges. *Sensors*, 19(9), 2019.

[42] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression. In *The AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[43] Zhaohui Zheng, Ping Wang, Dongwei Ren, Wei Liu, Rongguang Ye, Qinghua Hu, and Wangmeng Zuo. Enhancing geometric factors in model learning and inference for object detection and instance segmentation. 2021.

[44] Zhe Zhu, Dun Liang, Songhai Zhang, Xiaolei Huang, Baoli Li, and Shimin Hu. Traffic-sign detection and classification in the wild. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

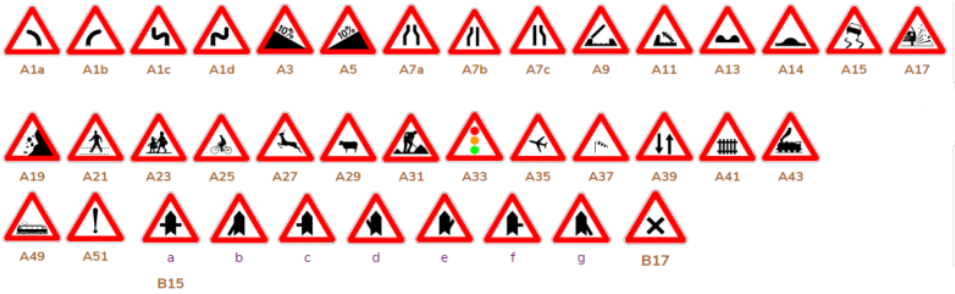
Appendix A

Traffic sign classes

This appendix explains which traffic signs are considered in this work and how they are distributed in the different classes. The classes where built based on the shape and colors of traffic signs because these features express the type of message the sign conveys.

A. Danger

- triangles



B. Priority

- revTriangles

- stop

- diamonds

C. Prohibitory

- forbidden

- redCircles

- whiteCircles

D. Mandatory

- blueCircles

E. Stopping and parking

- redBlueCircles

- parking

F. Indication

- squares

- rectangles

- arrows

Appendix B

Test set results examples

Figure B.1: Example 1: Ground truth Traffic Sign on the test set

Figure B.2: Example 1: Detected Traffic Sign on the test set