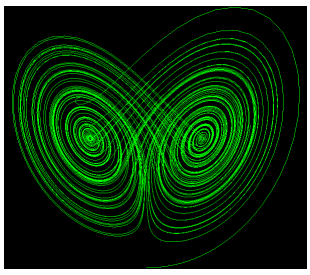
Departments of Mathematics, Physics and DACSO

MASTER THESIS

Modelling for Science and Engineering

Deep Learning models for depth estimation on autonomous vehicles

Alejandro Encalado Masiá



Juliol 2019

Universitat Autònoma de Barcelona

Contents

List of Figures

List of Tables

1 Abstract

This master thesis focuses on how to estimate the depth of a scene in real time for autonomous driving cars, using a stereo camera. Nowadays, thanks to the increasing computing power, we are about to substitute classical depth sensors devices, which usually are expensive, by other sensors such as cameras. Is in this context, Artificial Neural Networks (also known as Neural Networks) can perform these kinds of estimations without using these classical devices. In recent years, depth estimation using artificial intelligence has been developed for monocular vision and stereo vision, and both approximations have progressed significantly. However, according to Smolyanskiy [18], the gap between monocular and stereo vision remains significant due to the optical limitations of monocular vision. Whatsmore, in spite of the fact of this successes in computer vision for depth recognition using AI, for the moment, the inference obtained by this network remains is slower than the obtained by classical depth devices. Therefore, the main objective of this thesis is to implement a Deep Learning model to recognize depth in a scene using stereo vision, in the context of an autonomous car. To achieve this, we have integrated Deep Learning algorithms into a prototype of an autonomous car, built by members of the UAB and CVC. This prototype was previously able to determine the distance between objects by comparing their relative size using a monocular lens. However, we could improve this depth accuracy thanks to the integration of two CNN-based algorithms that until now have been used separately. One of the algorithms is responsible for detecting objects, in our case, bottles. Given an image, it is able to identify the object from the rest of the image uniquely. Whereas the other algorithm comes from one of the latest advances in depth recognition using Convolutional Neural Networks for stereo vision, using which we have managed to give this vehicle an estimation of the distance to obstacles more accurate than it was able to compute before. Using this new algorithm for depth recognition, the prototype is able to approach a test object or an object of interest autonomously without using any additional depth sensor apart from a couple of images in stereo. This achievement has been done by leaving the car to isolate the test object using the first neural network and obtain the depth information using the second one.

Despite this project uses several technologies and AI architectures, is the incorporation of the depth inference algorithm, the core of this thesis. To carry out the incorporation into the prototype's board, we had to apply specific predictive models to correct the discrepancy between the actual depth of the scene and that obtained by the neural network. This discrepancy exists because this network was not trained with the devices used in our vehicle. However, applying specific models and assumptions, we have been able to reduce this discrepancy. Also, we have compared the performance of the network with different parameters and devices to find the combination in which the vehicle can recognize the distance to a bottle faster without sacrificing accuracy.

Finally, the integration of the algorithm has been done in an already build a prototype which works using a low-cost GPU from Nvidia. In this context, we also discuss the complexity and troubleshooting when fitting a computationally expensive model in a low-cost computing system.

2 Introduction

It is universally acknowledged that Artificial Intelligence will entail a high impact on our lives. The increase of computing power alongside the development of new machine learning techniques such as Deep Learning is leading to astonishing new applications that could imply a significant improvement in how technology evolves. One of these great improvements that these techniques have recently deploy is in the context of autonomous vehicles. In this context, we present our thesis as a robotic project related to computer vision in the context of autonomous vehicles using a low-cost GPU, and what we are mainly concerned with this project is to implement and integrate Convolutional Neural Networks for depth estimation at runtime and check its performance over a known environment. Bearing this in mind, the purpose of this project is to achieve that the car's prototype can be able to understand and interact with the environment using a stereo camera. The idea behind this interaction is to make the car recognize objects of interest and position it in the 3D space, so, depending on the 3D spatial position of this object, the car could steer its wheels and throttle according to its distance, thus driving autonomously. For reaching this objective, we start implementing Deep Learning algorithms into a prototype of an autonomous car, built by members of the UAB and CVC. Before this master thesis, the car was able to steer the wheels depending on the position of an object of interests, which in this case, was a *bottle*. This object detection is done using a convolutional neural network algorithm for this porpuses. However, even the steering was accurate, the algorithm is unable to infer depth estimation. Nonetheless, the implementation of our algorithm leads to extract depth information using a stereo camera. The algorithm used in this project is based on Geometric and Context Network, which is a specific convolutional neural network architecture for mapping tasks, such as the problem which are concerned. Implementing this network we obtain the so-called *disparity map*, this map contains depth information of a given scene pixel-wise, that is, recover the same image as the input image but with the pixels colored depending on its disparity. Then, using this disparity map and the section of interest where the *bottle* is located using the object detector algorithm, we can assess the depth using some mathematical models.

First of all, we use an optic model to recover the $pseudo-depth$ of the *bottle* from the disparity map. Once we have this pseudo-depth, we can estimate the actual distance or *real* depth testing some other models based on the pixel value distribution of the pseudo-depth. The models proposed assume that the pixel value distribution of the pseudo-depth present two "peaky" points due to the geometric characteristics of the *bottle*. Then, these peaks are modeled using $Gaussian\ Mixture\ Model$ and $Highest\ peak\ model$. Finally, we validate these models with the OLS method using labeled data gathered specifically for this project. As a result of this evaluation, we find that the best model for our purposes is the $Highest\ peak\ model$.

Moreover, since this project is not only about modeling but robotics, we integrate this GC-Network into the existing computer architecture of the car's prototype, where the core of this prototype is an embedded GPU from Nvidia. One of the challenges of this work is to deploy a new architecture which handles not only both algorithms but the communication with the motor. This integration part is crucial in for project porpuses. The new algorithm has to communicate fast and accurate with the other parts of the machine.

Is for this reason that the integration of our algorithm comes with an update of the existing architecture of the car. This mentioned update comes with ROS, which is an operating system designed for robotic tasks.[1] As a summary, we can observe the following figure, which exemplifies the current architecture.

[1] In green, our proposal for depth recognition

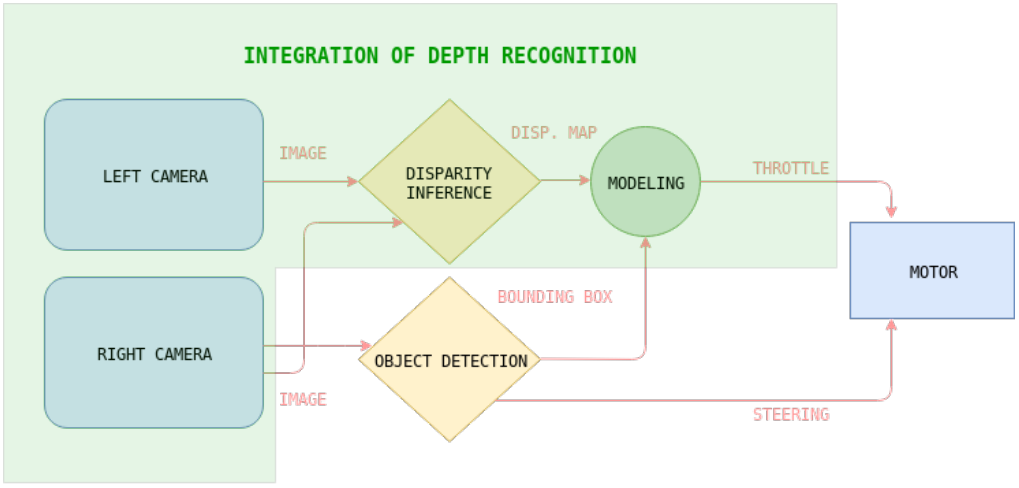


Figure 1: General pipeline of the project.

Finally, applying some performance tests using two different GPU's and different computing arithmetics find that we could reach up to 5Hz for depth estimation. So, using these embedded GPU'S, we are able to run these algorithms together fast enough for achieving autonomous driving for slow velocities, which means that shortly it is expected to reduce drastically the economic cost of an autonomous device keeping the error estimation of the depth small enough.

As for the object detection algorithm, since this network has been developed alongside this project, we are not going to focus on its details. For more information about the object detection algorithm refer to [5]

3 Theoretial framework

Computer stereo vision is a technique for reconstructing 3-D objects based on a pair of images. This technique is inspired by the concept of *stereopsis*, which is an ability present in many species of animals and refers to depth perception from information coming from the basis of two eyes. These eyes are usually separated in the horizontal axis, so the image comes from two different retinas. The differences between both images are referred to as *disparities*, where they are processed in the visual cortex of the brain to yield depth perception.

On the one hand, this depth reconstruction coming from disparities can be explained using simple optics phenomena, where it will be discussed below. Differently, the visual cortex is a section of the brain able to receive a specific pair of images and perform this depth inference according to the requirements of the animals. So, no optics or mathematical approach is needed explicitly in order to infer depth. However, the existence of a mathematical model which ensures that it is possible to infer depth from two stereo pair of images is necessary to build a neural network model.

According to this explanation, this project is focused on emulate the visual cortex in a GPU using Deep Learning algorithms. So this section is divided into two parts. In the first one, we discuss how we can infer depth using a stereo pair of cameras. The second one focuses on the convolutional neural network architecture used in this project.

3.1 Stereo vision for depth recognition

From the field of optics, we can ensure that if we have two image sensors (i.e: two cameras), we can compute the depth of a single real point as a function of the focal distance of this lens and the distance between them.

As we can observe in the picture below, a real point is projected over the focal plane. In this projection, the point has a different x position for each lens, using the rule of similar triangles, we obtain the following expression for Z, which represents the depth of this real point.

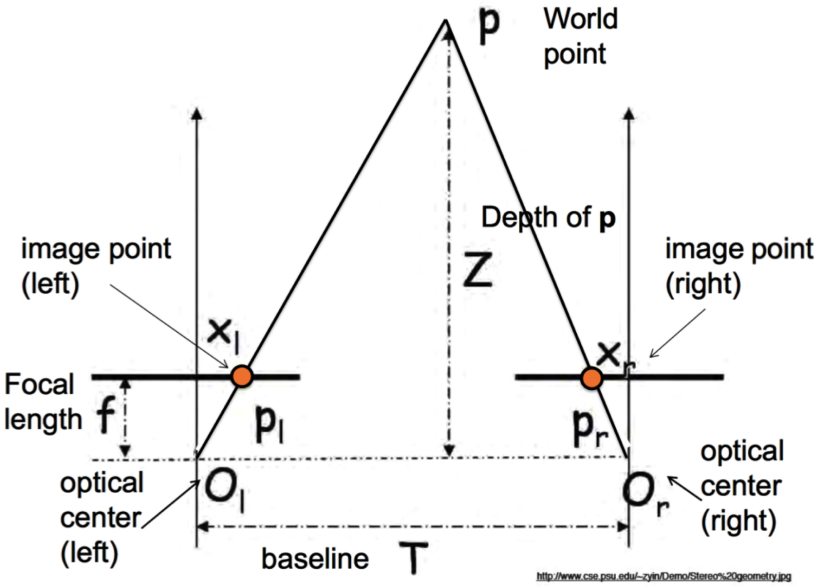


Figure 2: Optic diagram retrieved from [17]

$$Z = \frac{f \times T}{T - D} \tag{1}$$

Where D is the disparity ($D = X_l - X_r$), which is the distance formed by the virtual points (projected) over the left and right focal plane.

The last expression shows that this depth depends only on the disparity since the focal and the distance between cameras are known and can be treated as constant. So, the algorithm we use to depth estimation focuses only on this disparity computation. In the end, we receive an array where each pixel is set with its respective disparity. This image is named from now on as disparity map.

Since this expression exists, instead of using a mathematical model approach to obtain the disparity map, we use a convolutional neural network for obtaining this disparity map.

3.2 Deep learning algorithms

Deep learning algorithms (DL) are a set of techniques coming from the field of machine learning where are based on a neural network model. DL has the employs algorithms to imitate think processes, like abstraction. These algorithms are feeding with complex

data such as images or speech. Then, this data is passed through layers of computational units, which allows extracting features of this data.

This layer contains algorithms with at the end are supposed to extract "meaningful" features which can be used later for an understanding of the data. However, deep learning models are an extension of neural network models, so a brief explanation of neural networks is commented below.

3.2.1 Neural Network Models

In the context of machine learning, the neural network is constructed by a graph which emulates the connexions between brain neurons. The human brain has approximately 10 billions of neurons, each neuron connected to other 10.000 more neurons, is this kind of assembling which allows us to interact with the environment, think, and abstract. These neurons have three different main parts, the dendrites, where the electrochemical input is received, the soma, which is the central part of the neuron and the axon, where the electrochemical output is sent if any.

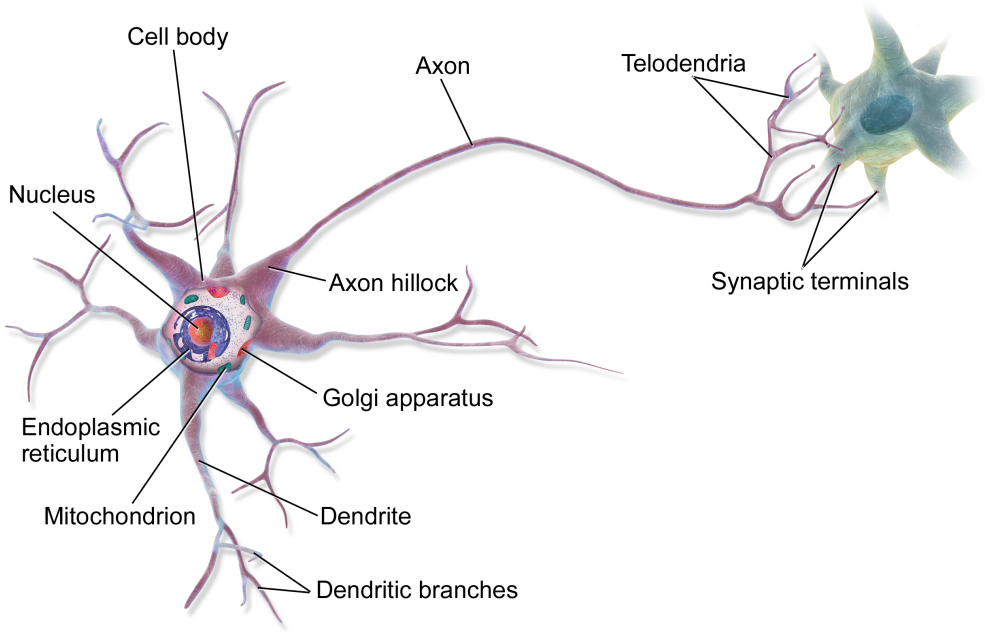


Figure 3: Structure of a neuron. [6]

So a simple model of the brain can be explained as a graph, where each node represents a single computational unit, called perceptron. Also, the connections between them are represented by the weights.

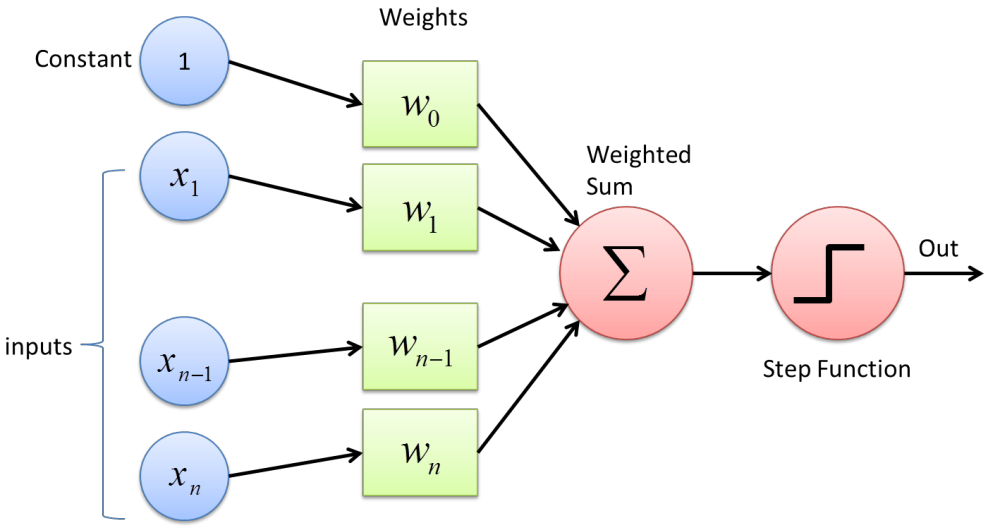


Figure 4: Structure of a perceptron. [12]

The image above exposes how this unit works. It merely adds together all its inputs multiplies by a weight. Then, the result passes through an activation function, which usually is a non-linear mathematical function. After this process, the output is sent as an input of another neuron or can be used as an interpretable result for prediction. This perceptron can perform simple mathematical and logic operations, but the strong point of this assembling is that a set enough perceptrons correctly linked together can perform any non-linear operation and classification. The neural network is constructed by adding two types of layers of perceptrons together. The first one is the so-called input layer, and it is just a vectorized expression of the input data. The last layer is called the output layer, here is where we receive the result coming from the network. Finally, in the middle, we have the hidden layers, which are a bunch of layers where its input comes from the last layer made of perceptrons and the output is sent to the following layer, again made of perceptrons.

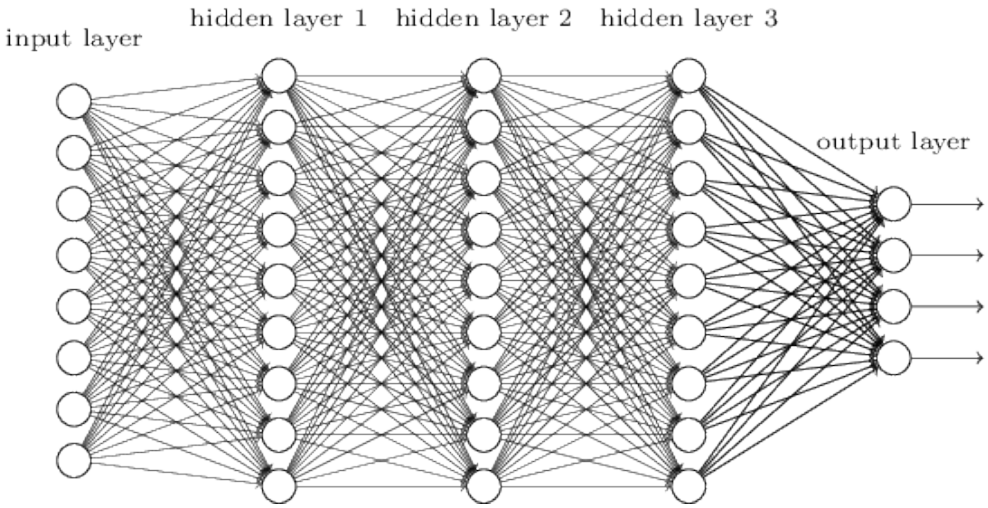


Figure 5: Neural network representation. [13]

Its weights rule the linkage between perceptrons, and these can be updated via *learning process*. That is, allow the network to perform an operation and tell it if the answer is correct or incorrect. In case this answer given by the network is incorrect, it modifies its weights through a process called *backpropagation*.

After each weight modification, the network can give a closer correct answer to the problem that is designed for, given the data. This process is analogous as the animal process of learning, where we as animals are able to analyze specific situations and perform the proper action after a test period.

As a result, a neural network can be interpreted as a composition of non-linear functions so that it can be modeled as a single function, which can be derived by its weights using the chain rule.

This backpropagation is an algorithm for update the weights that rely on gradient descent algorithms and exploits the chain rule present in a neural network. The main feature of backpropagation is its iterative, recursive, and efficient method for calculating the weights updates to improve in the network until it is able to perform the task for which it is being trained. Backpropagation requires the derivatives of activation functions to be known at network design time and is commonly used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function.[2]

The data scientist can establish this loss function but usually is chosen the mean squared error between the real value and the output value for regression problems.

3.2.2 Convolutional Neural Networks

Convolutional neural networks are a type of deep learning algorithms, which its architecture is based on the visual cortex of an animal brain. Studies from 1950's to nowadays shows that this particular region of the brain has some characteristics which make this region excellent in vision tasks such as object segmentation, object recognition, pattern recognition and depth recognition. The biological architecture of the brain cortex of the mammals has the following properties:

[2]Source [1]

- Contains specific neurons that fire when is applied some specific stimulus. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.[3] For instance, there is a small number of neurons linked together that responds only to an edge shape. These neurons are so-called receptive field.
- It has multiple layers which are optimized to detect different responses. The primary ones are made for recognizing shapes and colors. Then, there are other layers which recognize motion, depth, face recognition, and other features.
- It has a hierarchical organization, that is, the different layers combine for extracting meaningful information of the scene.

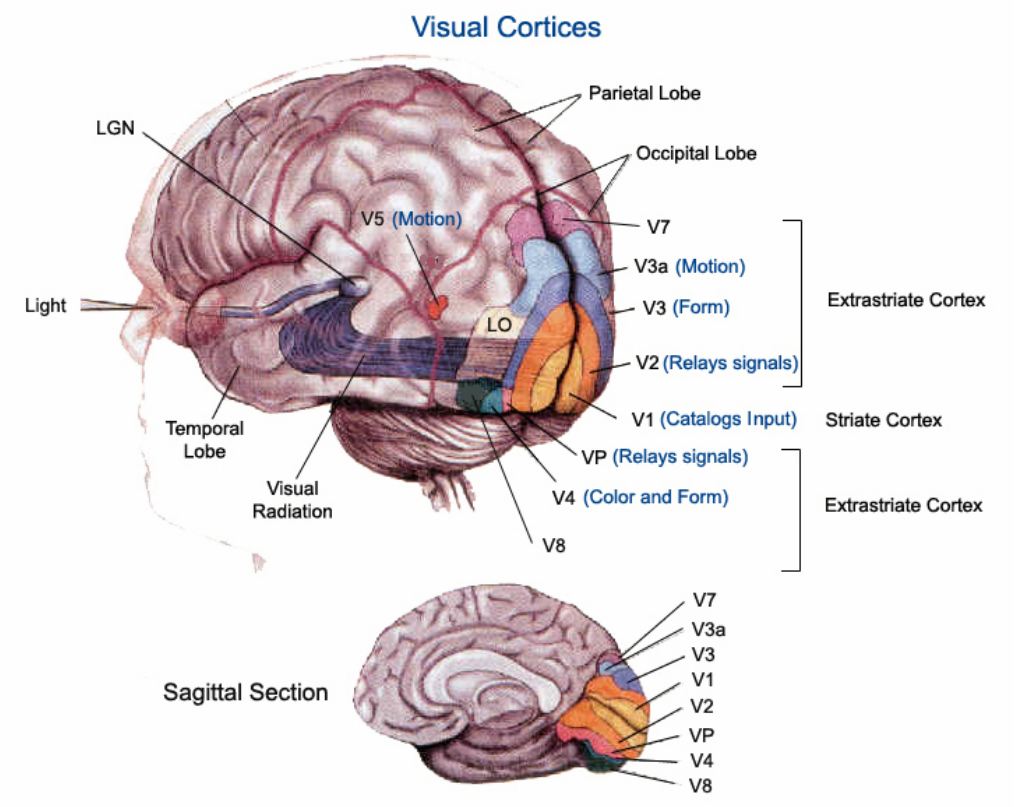


Figure 6: Visual cortex representation

Figure 6 shows how these layers are located in the human brain. Light information comes from the optic nerves to the V1 layer and it is processed through the rest of the layers in a hierarchical way. This assembling leads to build a neural network with the same characteristics; this new kind of architecture is called Convolutional Neural Network. The main differences over a conventional neural network are the following:

[3] source [2]

- The input data is a tensor, that is, an n-dimensional array. Usually, the tensor for an RGB image has dimension $Heigh \times Width \times 3$ Where 3 corresponds to the intensity value of Red Green or Blue. Whatsmore, the feature extraction of a single image can lead to a higher dimensional tensor, formed by stacking activation maps, which is discussed below.

- The kernel of the layer models the receptive field. This affects only a small region and is able to extract features from it. Its most important properties are local connectivity and shift invariant. The last one means that the weight of this kernel is independent of the spatial region which is applied for each layer. The operation that the kernel applies to the data is called by convention *Convolution*. However, the mathematical operation is *Cross-Correlation*

- The activation maps model hierarchical architecture, and each kernel is able to extract information from 1 single layer and map it into the activation map, which is the result of the convolution. Composition of these maps leads to another tensor, which dimensions can differ from the input dimensions depending on several aspects, like the way the convolution is applied and/or the number or kernels applied. This new tensor is sent as input for the next layer of the network.

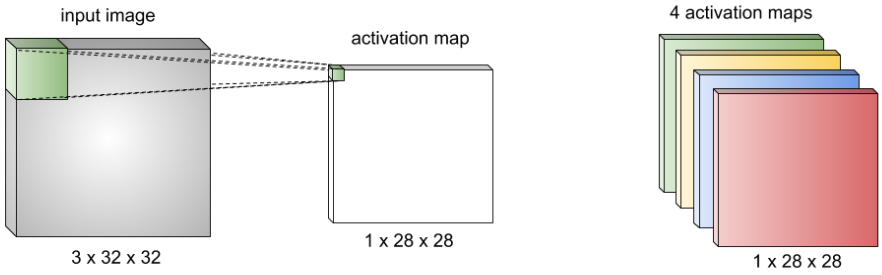


Figure 7: Example of convolution and activation map

The figure above exemplifies how a kernel of dimension $3 \times n \times m$ is applied to a tensor with dimension $3 \times 32 \times 32$ the result of the operation then is a real number. This numbers fills a tensor with dimension $1 \times 28 \times 28$. This new tensor dimension is due to the characteristics of the convolution. This tensor is the *activation* map. Finally, these maps add together in function of the number of kernels applied to the same image. The process of the convolution operation is called convolutional layer.

Apart from these layers, there is another type that doesn't depend explicitly on the weights, and its purpose is to reduce dimensionality. These layers are called pooling layers. This operation consists in apply a non-linear function to the spatial region of interest and mapped to a 2-D array for each layer. These functions are usually the mean, the average or the maximum value obtained by each region.

Alongside the convolutional and pooling layers, which are the discussed above, there are some other layers that are similar to the present in the common neural networks assemblings. CNN also presents activation layers, that are an extension to the activation function. The core of these layers is the same as in the activation functions. Some popular functions used are ReLU, sigmoid, or arctanh.

A basic design for convolutional neural network architecture is like the design for a fully connected neural network but adding the principal features commented above. Whatsmore the way the network updates its weights is via backpropagation as well.

4 Modeling

In the lasts sections we have seen how can we obtain depth information from a stereo pair of images. From the field of optics, we learn that with a stereo pair of images we can obtain the disparity map, which is an image where each pixel represents the $dif erence$ or disparity from both images given a scene. Whatsmore, this disparity map can be transformed in a *depth map* using a formula which depends on the features of the camera.

Thus, the modelization pipeline proposed in this project for depth estimation of a specific object is the following:

- Use a pretrained CNN which outputs a disparity map from a stereo pair of images
- Model this disparity map in order to obtain the desired depth information.

However, this structure alone presents some leaks. First of all, as we want to use these algorithms for autonomous driving, we can not focus on the depth map of an entire scene, but in the objects of interests. Moreover, the information of the disparity map is pixel-wise, which means that each pixel of the map has information about the depth. However, we need depth information about the whole object of interest, so the model we use not only needs to transform from disparity map to depth map but has to extract meaningful information from this last one.

Nevertheless, one of the ideas to solve these troubles is to use an object detector algorithm. Thankfully, the car's prototype we use for this project comes with a pretrained CNN for object detection, which is able to draw a bounding box where the object of interest is in, given an image. With this assembling now we can perform two tasks.

- Using the bounding box, we can assess the relative position in the X Y plane of the object of interest in the scene, so, we can use this information for computing the *steering*
- From the bounding box of the object and the corresponding section of the disparity map, we apply some models for getting an estimation of the depth of the object, which in this case, this depth is the distance of the object from the car. The information of this distance leads to the computation of the *throttling*.

Looking at 2, we can see the broad pipeline of this project, as we explained above. However, since we are interested in depth recognition, the following figure shows the modelization pipeline followed by this thesis.

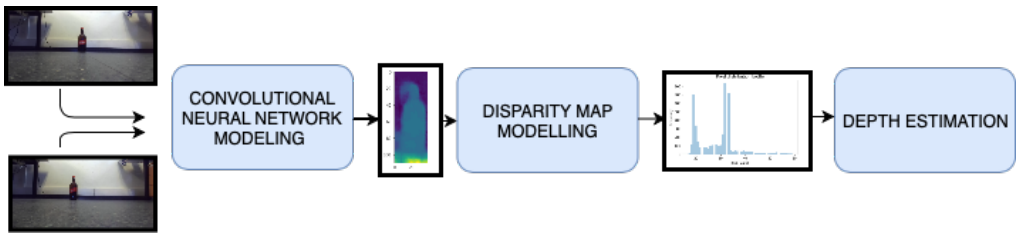


Figure 8: Modeling pipeline of this project

As a summary, in this section, we discuss how the CNN models can be used for obtaining a disparity map of a bottle, then how can we model this disparity map in order to obtain the estimated depth from this bottle, and finally, how can we test these models for assessing the most accurate depth estimation.

4.1 Convolutional Neural Network modeling

After discussing the theoretical framework of the convolutional neural networks, in this section, we focus on the modelization used for this project for disparity map inference. The model used was extracted from the repository [9] by Nikolai Smolyanski, Alexey Kamenev and Stan Birchfield. In this repository, we have the code of the CNN and the paper [18] where this architecture stands on. This network is based on the GC-Stereo network (Geometric and Context Network), which was the leader of Kitty 2015 benchmark at the time the paper was written.

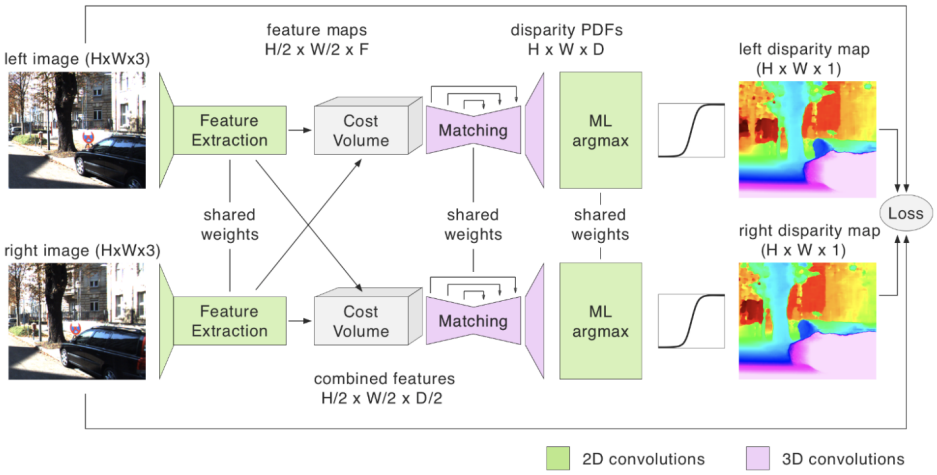


Figure 9: CNN architecture. Source [18]

This schema shows the assembling of this network briefly. First of all, the images are processed by 2D feature extractors. In our case, we used ResNet18 2D feature extractor. The dimensions of the input images are $Height \times Width \times 3$, where 3 corresponds to the channel input (RGB). Then, The output of the ResNet18 are tensors with dimensions $H/2 \times W/2 \times F$, where $F = 32$ is the number of features. These tensors are used to create two cost volumes. One for the right image and other for the left. The left-right cost volume is created by sliding the right tensor to the left, along the epipolar lines of the left tensor, after first padding the left feature tensor on the left by the maximum disparity. Finally, the features are concatenated at the pixel position. The result is a 4D cost volume with dimensions $D/2 \times H/2 \times W/2 \times 2F$, where D is the maximum disparity. As the figure shows, the weight for the feature extractor is shared due to the nature of the problem. GC network is based on this sharing between weight, where this technique leads to capture spatial features.

The cost volumes are passed through a 3D convolution/deconvolution that performs stereo matching by comparing features. Then, again, the weight is shared and learned together. After the last decoder layer, upsampling is used to produce both left and right

tensor (dimensions $D \times H \times W \times 1$) containing matching costs between pixels in the two images.

Finally, for obtaining the best disparity for each pixel, this architecture, instead of using a soft argmax function, use a machine-learned argmax (ML-argmax) function.

This function consists of a sequence of 2D convolutions that produces a single value for each pixel after passing through a sigmoid function.

The last step is to apply a loss function using the outputted disparity maps from the network.

$$L = \lambda_1 E_{image} + \lambda_2 E_{LIDAR} + \lambda_3 E_{lr} + \lambda_4 E_{ds} \tag{2}$$

Without entering into details, this loss functions takes care about photometric consistency, that is, it ensures that left and right disparity maps are consistent with each other.

The terms that appear in the loss function are E_{image} the input images, E_{LIDAR}, disparity map comming from LIDAR. E_{lr}, pixelwise distance cost between left and right images, and E_{ds} disparity maps.

The training and testing of this custom network were done using the KITTI dataset. This dataset comes from an equipped standard station wagon with two high-resolution color and grayscale video cameras, while a laser scanner provides the actual ground truth. Then, the images are recorded by driving around the city of Karlsruhe, in rural areas and on highways. Up to 15 cars and 30 pedestrians are visible per image. With this enabling, the network could be feed with a stereo pair of pictures of this dataset and trained to obtain the disparity map, obtained by the laser scanner. This training required over 40 GPU days and about 29K training image, containing its LIDAR ground truth were used for this porpuses.

4.2 Modelization of disparity map

Once we are able to retrieve a disparity map using a convolutional neural network, here we discuss the way we model this map for assessing the actual distance to the object. Recalls that in our case, the object of interest is a *bottle* . The idea behind this approach is the following. Given a single disparity map corresponding to a bottle placed on the floor, we can distinguish three objects from the scene: The floor, the bottle, and the wall. The pixels which belong these objects have a specific pixel density distribution due to its geometric characteristics. Knowing the distribution corresponding to a particular shape, we can isolate it from the pixel histogram coming from the disparity map so as to obtain some statistics of interest.

In this step, we face two problems.

- The disparity map comes from a CNN pretrained with a different stereo camera that the one used in this project. The meaning of this is that even the disparity map shows a "meaningful" scene, this map doesn't have a correspondence with the real depth of the scene when using equation 1 and the parameters of our assembling.
- We need to obtain the correct pixel density distribution from a bottle

The way of solving these last two problems is the following. For obtaining the correct correspondence between the disparity map and the *real* depth we assume that, even this is depth is "uncalibrated" due to the problem commented above, from the "false" disparity map, we can apply the following transformation:

$$D_{pseudo} = \frac{1}{Disp} \tag{3}$$

Here we define a new object called D_{pseudo} which is the "pseudo" distance computed by the algorithm. This distance is just the inverse of the disparity. From here, we assume that this pseudo distance has a relationship with the real distance with the following form:

$$D_{real} = g(D_{pseudo}) \tag{4}$$

That is, the real distance is a function of the pseudo distance. To find this function, we design another experiment to recover the real depth from a sample of bottles. Having the pseudo-depth and the real depth for these samples, we can fit $g(D_{pseudo})$ and calibrate the result.

As for obtaining the pixel density distribution from the disparity map, what we propose is obtain this distribution by simulating a cylinder in a controlled environment. Then, with this simulation, we can obtain a simulated depth map and use it for knowing the real pixel distribution of the bottle. In our case, we choose a cylinder instead of a bottle for simulation because is more comfortable to implement in the simulator environment and the geometric characteristics of both objects are similar, so we don't expect a significant difference in between the simulated depth map and the real one.

Whatsmore, recall that we are dealing with depth maps instead of disparity maps. However, this is not relevant for this simulation porpuses, due to both maps can be transformed using 3 and we don't care about the constants in this step.

The simulation environment we choose is Gazebo simulator. Gazebo is an open-source 3D robotics simulator build which uses high-performance physics engines. It also provides a realistic rendering of environments, including high-quality lighting, shadows, and textures. It can model sensors that "see" the simulated environment, such as laser range finders, cameras (including wide-angle), Kinect style sensors... For this reason, Gazebo Simulator is suitable for simulating porpuses, as we need. This simulator allows to place a depth camera and connect it to a ROS node to get its output. This output from the depth camera is the so-called *Point Cloud* in computer vision.

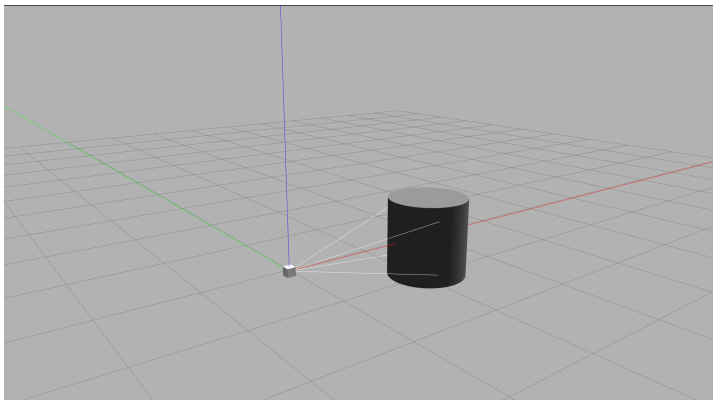


Figure 10: Gazebo simulation environment. This scene simulates how the camera takes a depth picture from a cylinder.

A point cloud is a set of data points in space. These are generally produced by 3D scanners, which measure a large number of points on the external surfaces of objects around them. As the output of 3D scanning processes, point clouds are used for many purposes, including to create 3D CAD models for manufactured parts, for metrology and quality inspection, and for a multitude of visualization, animation, rendering, and mass customization applications. In our case, this point cloud is used for computing

a depth map.[4] This depth map is computed applying a projection function that maps each 3D point (X, Y, Z) into the (X, Y) plane. Having this map, we can convert it into an image, where each pixel of the image corresponds to a certain depth in the Z axis. From this map, we can retrieve the pixel histogram from the cylinder. Then, using this as a reference distribution, we can obtain the statistics of interests for the real bottle.In our simulation, we choose a cylinder at 2 meters far from the sensor and with dimensions of 1m of height and a radius of 0.5 m. From this image, we could obtain the following point cloud.

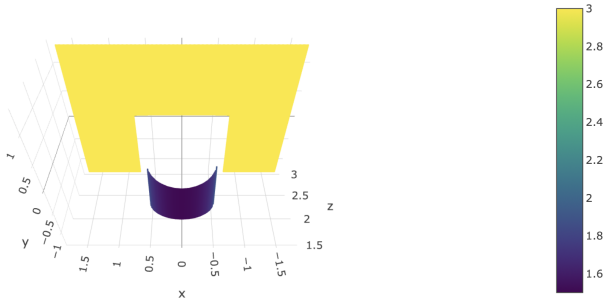


Figure 11: Point cloud for a simulated cylinder

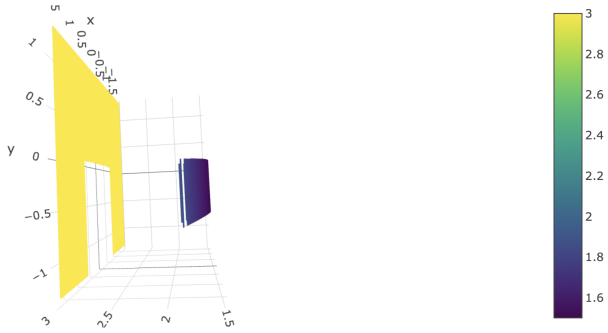


Figure 12: Point cloud for a simulated cylinder

The figure shows the contour of the cylinder centered at 2 meters where the closest part to the camera is at 1.5, which is what we expect. However, even the wall was not simulated, we can observe it from the point cloud representation due to the limitations of the camera.

[4]source [10]

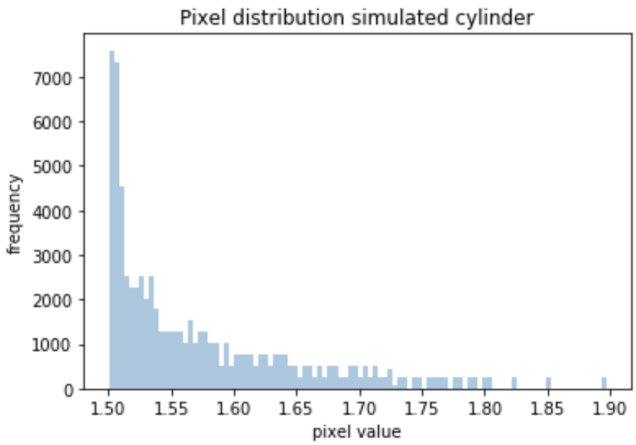


Figure 13: Point cloud for a simulated cylinder

Taking a look to the pixel histogram, one of the candidate functions for fitting this histogram is the inverse function (i.e $f(x) = \frac{k}{x-a}$). Where a is the divergence point, which represents the closer part of the cylinder. However, since we are not interested in all the distribution, we can focus only on the divergence point. This point of divergence can be modeled as a delta function, which has the following definition:

$$\delta(x-a) = \begin{cases} \infty, & \text{if } x = a, \\ 0, & \text{if } x \neq a. \end{cases} \tag{5}$$

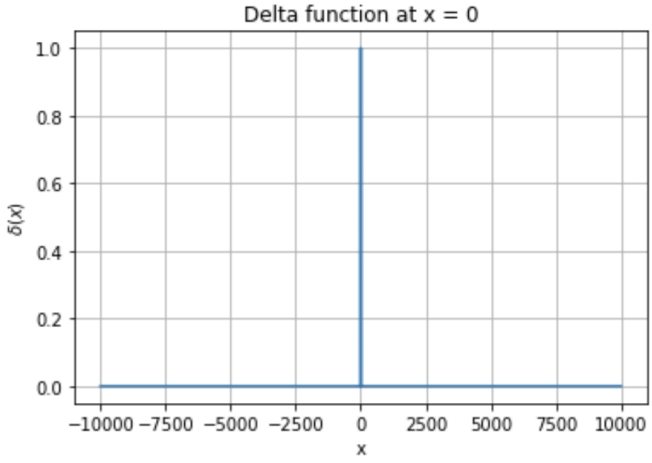


Figure 14: Dirac delta function representation

Dirac delta function is used to model the density of an idealized point mass or point charge as a function equal to zero everywhere except for zero, and whose integral over the entire real line is equal to one. This function has a value of ∞ at the point a, the same as the divergence point of the inverse function. So, approaching the divergence point in the inverse function from the left side has the same behavior as the delta function.

$$\lim_{x \to a} \frac{1}{x - a} = \delta(x - a) \tag{6}$$

Another property of the Dirac function is that it can be expressed in the following way:

$$\delta(x - a) = \lim_{\sigma \to 0} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - a)^2}{2\sigma^2}\right) \tag{7}$$

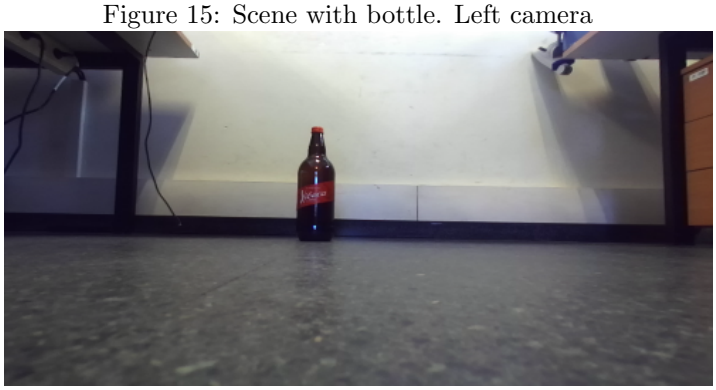
Adding both expressions together, we can assume that the divergence point in the pixel histogram can be reduced to a peaky gaussian if the variance of the normal distribution is small enough. In summary, there are 4 main assumptions for computing the depth given the disparity map.

- There are two quantities to compute, D_{pseudo} and D_{real}. Where D_{pseudo} is computed by the pixel histogram of the disparity map.
- As our model only detects bottles, these can be modeled as cylinders, and we are interested just in the nearest part of the cylinder, which corresponds to the highest "peak" at the histogram
- This peak can be modeled using δ Dirac function
- δ Dirac function can be considered as a Gaussian function with a "small" variance.

All in all, the following part is to examine the pixel histogram coming from the disparity map of a real scene. In our case, we have chosen the following image for the case study.

.5

Figure 15: Scene with bottle. Left camera



.5

Figure 16: Scene with bottle. Right camera

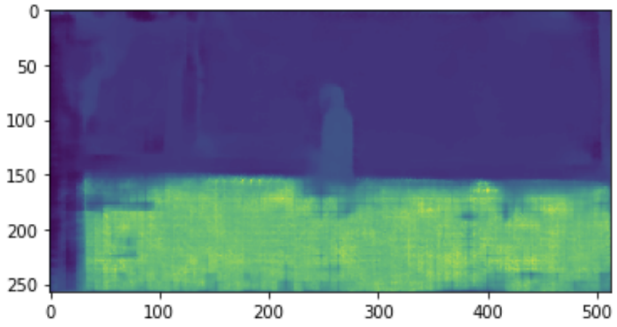


Figure 17: Disparity map from the figure 16

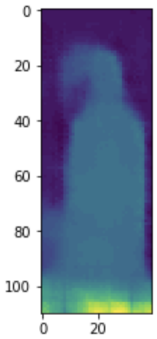


Figure 18: Cropped section from the last figure corresponding to a bottle

The first two images correspond to the scene reported. As we observe, the essential elements of the scenario are the wall, the floor, and the bottle. There are other elements such as a table or a drawer, but as there are no important, we only focus only on the bounding box that surrounds the bottle.

The last two images display the disparity map computed by the CNN based on ResNet18 2D feature extraction, which is explained in section *Convolutional Neural Network modeling*. Then, using the bottle detector network, the scene can be cropped as the last figure.

The cropped image contains the section of the wall, the floor, and the bottle itself. If we apply the cloud point transformation to this image, we can observe the following scenario.

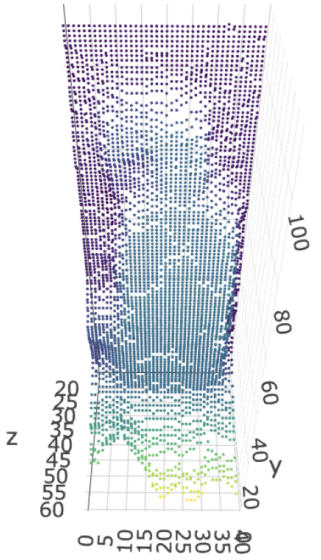


Figure 19: Frontal view of point cloud corresponding to a bottle from disparity map.

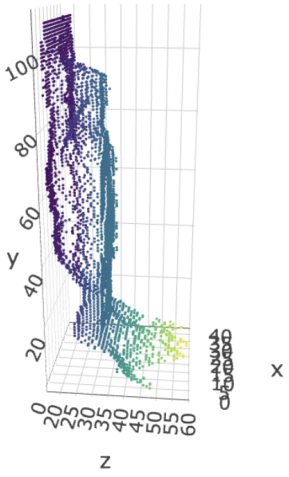


Figure 20: Side view of point cloud corresponding to a bottle from disparity map.

The 3D representation of the bottle leads us to distinguish more comfortable the three main components of the picture. This spatial representation not only is a more helpful way to view the object but could also provide us other ways to isolate the bottle from the rest of the scene.

Finally, the pixel histogram representation of this bottle is the following.

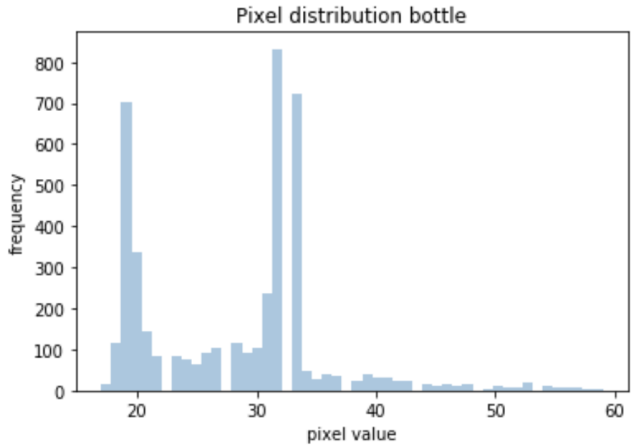


Figure 21: Pixel histogram of the cropped disparity map.

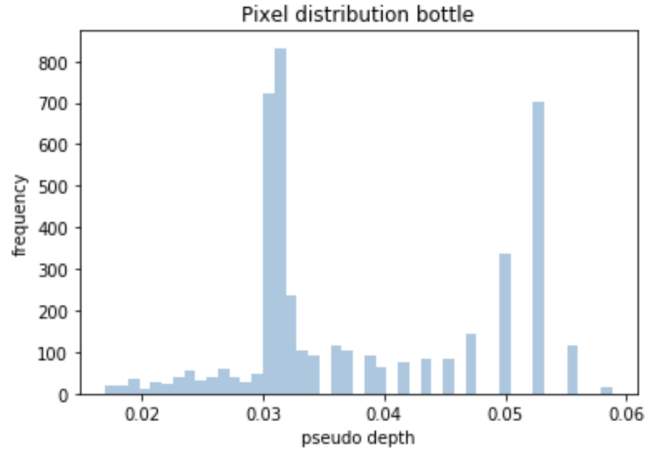


Figure 22: Pseudo-depth histogram of the cropped disparity map.

The first histogram represents the number of pixels which has the same value within a range in the cropped disparity map. The second one is the same, but instead of representing the pixels, we represent the pseudo-depth.

For the first image, we can distinguish two main peaks. The first one corresponds to the wall and the second one to the bottle. However, since we are dealing with pixel value, this interpretation is not clear enough. Nevertheless, if we move to the second figure, we can argue why the first peak represents the bottle and the last one the wall. This histogram counts the number of points in the image which remain at the same depth within an interval. Even though this is not the real depth, this quantity is ordered. So that means that objects which are closer to the camera have a value for this depth smaller than objects which are further.

Whatsmore, it has a sense that these objects display a peak in this histogram. Recall that the cropped image focuses only in the bottle. Moreover, due to its shape, almost all the closer points of this bottle lies on a plane, so there is a high number of these points, which has the same depth. Whatsmore, a similar thing happens to the wall, even there are fewer wall points in the image, those points lies in the same depth due

to the geometric characteristic of the object.

4.2.1 Gaussian mixture model

One of the models he could use for assessing the estimated pseudo-depth from disparity map is Gaussian mixture model. This model can be used only if we assume that δ Dirac functions represent not only the bottle but the wall. Then we can conclude that these two peaks correspond to the closest part of these objects from the camera. To achieve this fitting, we used Gaussian mixture models to fit the data into two normal distribution functions. In order to explain how Gaussian Mixture Models works, we can take a look at the description present in [4]

"One hint that data might follow a mixture model is that the data looks multimodal, i.e. there is more than one "peak" in the distribution of data. Trying to fit a multimodal distribution with a unimodal (one "peak") model will generally give a poor fit, as shown in the example below. Since many simple distributions are unimodal, an obvious way to model a multimodal distribution would be to assume that it is generated by multiple unimodal distributions. For several theoretical reasons, the most commonly used distribution in modeling real-world unimodal data is the Gaussian distribution. Thus, modeling multimodal data as a mixture of many unimodal Gaussian distributions makes intuitive sense. Furthermore, GMMs maintains many of the theoretical and computational benefits of Gaussian models, making them practical for efficiently modeling very large datasets. Gaussian mixture model is parameterized by two types of values, the mixture component weights and the component means and variances/covariances. The mixture component weights are defined as phi_k for C_k with the constraint that $\sum_{i=1}^{K} = \phi_i = 1$ so that the total probability distribution normalizes to 1. If the number of components is known, expectation maximization is the technique most commonly used to estimate the mixture model's parameters. In frequentist probability theory, models are typically learned by using maximum likelihood estimation techniques, which seek to maximize the probability, or likelihood, of the observed data given the model parameters. Unfortunately, finding the maximum likelihood solution for mixture models by differentiating the log likelihood and solving for is usually analytically impossible. Expectation maximization (EM) is a numerical technique for maximum likelihood estimation, which is an iterative algorithm and has the convenient property that the maximum likelihood of the data strictly increases with each subsequent iteration, meaning it is guaranteed to approach a local maximum or saddle point."

EM algorithm consists in two steps, the first step, known as the expectation step or E step, consists of calculating the expectation of the component assignments C_k, for each data point given the model parameters. The second step is known as the maximization step or M step, which consists of maximizing the expectations calculated in the E step concerning the model parameters. The entire iterative process repeats until the algorithm converges, giving a maximum likelihood estimate.

This GMM algorithm can be easily implemented in python. For instance, sklearn provide the algorithm already build and optimized.

Finally, applying it to the data coming from the pseudo-depth distribution, we can obtain the following.

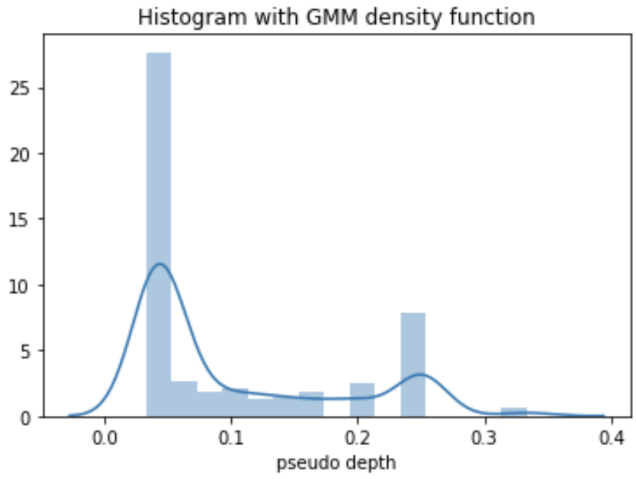


Figure 23: Pseudo-depth histogram with pixel GMM fitting.

Moreover, since this comes from the cropped image of the bottle and most of the pixels belong to it, we assume that the Gaussian with the smallest mean corresponds to the region of the bottle which is closer to us, so from here we can estimate the pseudo-distance to the bottle using this mean.

4.2.2 Highest peak model

For obtaining the depth from disparity map, we have seen that the histogram could be fitted with a set of normal functions if some conditions are fulfilled. However, not in all cases, the histogram shows the desired distribution due to noise or other factors that we can not control. As we are interested in the closer part of the bottle, in the previous section we argue how we can fit the histogram to a Gaussian model to get the mean value, which is the "peaky" point of the histogram. Another way to obtain this pseudo depth from the pixel histogram is as simple as select the bin which corresponds to the maximum pixel count. This approximation is only possible if we assume that the bottle has approximate the same depth distribution as shown in the cylinder models and the ratio between the wall and the bottle within the bounding box captured by the bottle detector algorithm is less than 1. If these conditions are fulfilled, then it would be more pixels belonging to the depth than to the wall, so the maximum peak in the pixel histogram coming from the disparity map would be the highest one.

4.3 Model testing

What we are mainly concerned with here is to obtain the estimated distance from the prototype's car to the bottle given the estimated pseudo-depth from the models mentioned before. As we have discussed in the previous sections, we are now able to detect the bottle and isolate the disparity map from it. Also, we have discussed some models for getting an estimated distance from this disparity map. However, recall that this disparity map doesn't reflect the real distance, but the pseudo-distance from the bottle, due to the characteristics of our assembling. Consequently, in this model testing tasks, we collect disparity maps from a generic bottle and label them with the real depth, then this collection of data is used for calibrating the model. Once the model is calibrated, we could use it for retrieving the actual distance from the disparity map in real time. The experiment we have done for calibrating the three models we have

discussed is the following. Having the disparity map from the bottle given a scene, the only way we have to measure the actual depth from the car's camera to the bottle is by using some measuring device. Our choice for measuring this distance is to take a picture from the top of the scene and count the pixels from the camera to the bottle. Knowing the number of pixels and the scale, we can obtain the real distance. For doing this experiment, we need to set up the following material.

- stereo camera connected to a device able to run the CNN for retrieving the disparity map.
- A piece of sheet with a fixed length mark. This paper is used for setting up the scale when analyzing the picture.
- A camera placed in the top of the scene. In our case, the camera used was from the smartphone.
- A set of scripts for retrieving the three images at the same time.
- A bottle.

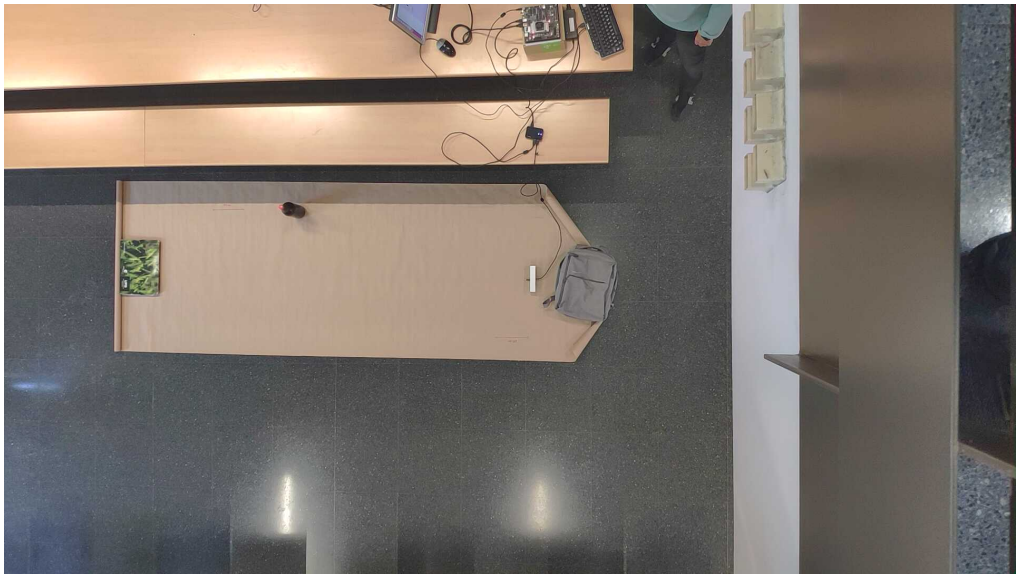


Figure 24: Top image used for the calibration of the model.

The figure above shows the assembling of the experiment. On the one hand, on the floor, we have the stereo camera pointing towards the bottle. This camera is connected to Nvidia Jetson TX2, which is able to focus on the disparity map of the single bottle. At this moment, the device can cluster bottles and its disparity map. On the other hand, we have a smartphone placed on the top of the scene and perpendicular to the floor. This smartphone is running an application which allows exposing the images from the camera in a local IP address and port, so, connecting the smartphone and the Jetson to the same internet connection, we are able to access to the images from the smartphone using a python script. In the top of these two parallel applications, there is another script that recovers the disparity map of the single bottle, and at the same time, gets an aerial image from the scene.

After that stage, the first thing we need to do is measure the distance from the camera to the bottle. For doing this, we used an Image processing software called ImageJ. This tool was developed at the National Institutes of Health and the Laboratory for Optical and Computational Instrumentation (LOCI, University of Wisconsin). Also, ImageJ was designed with an open architecture that provides extensibility via Java plugins and recordable macros.

With ImageJ we can easily set up a scale from the picture using the formula below.

$$D_{real} = N_{pixels} \times \frac{D_{scale}}{N_{scale}} \tag{8}$$

Once we have set the scale, we can measure the distance from the camera to the bottle drawing a line in between them. Recall that this measure is valid only if the top camera is perpendicular to the floor, so, it is essential to maintain the proper orientation of the camera during the experiment.

In the end, with this assembling, we obtain the actual distance and its corresponding disparity map using one hundred measures. So, according to the expression shown in 4, we can fit a linear function which relates the actual distance and the estimated distance obtained by each of the three models.

4.3.1 Performance of the models

To evaluate the goodness of the models used for depth estimation, we performed a linear regression model based using OLS. The function we want to fit with the data recovered with each of the models is the following

$$D_{real} = m \times D_{pseudo} + C \tag{9}$$

The models used for this fitting are the following.

- Model 1: Gaussian Mixture model using the original disparity map from the cropped section corresponding to the bottle
- Model 2: Gaussian Mixture model using the resized disparity map from the cropped section corresponding to the bottle.
- Model 3: Highest peak model using the original disparity map from the cropped section corresponding to the bottle.

Notice that when using the cropped section of the bottle, the size of this section depend on the distance where the bottle is. This fact may affect the performance of the model.

Whatsmore, we discard the labeled data which its disparity map was not properly obtained. In our case, we need to discard some of the images from the 100 we took.

The figures below and the table shows the goodness of the fit for the expression before 9.

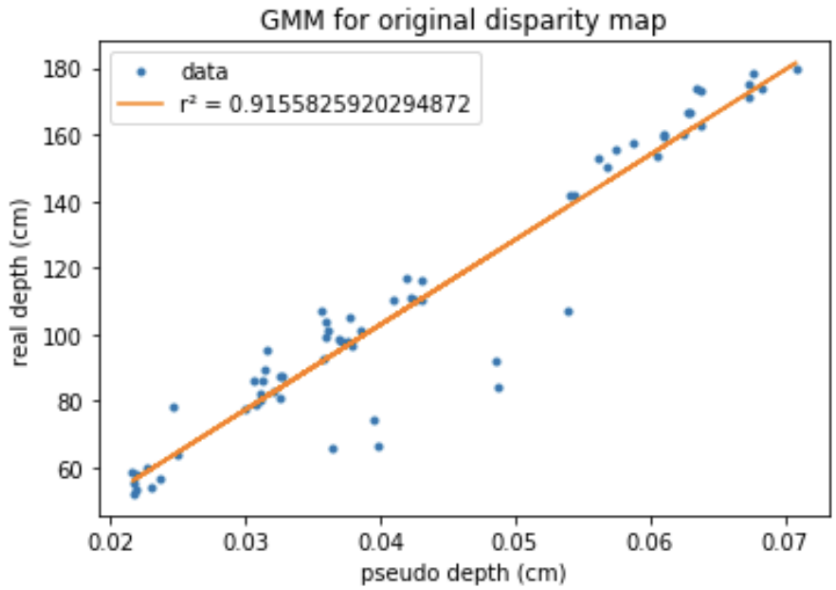


Figure 25: Linear regression using OLS for Model 1.

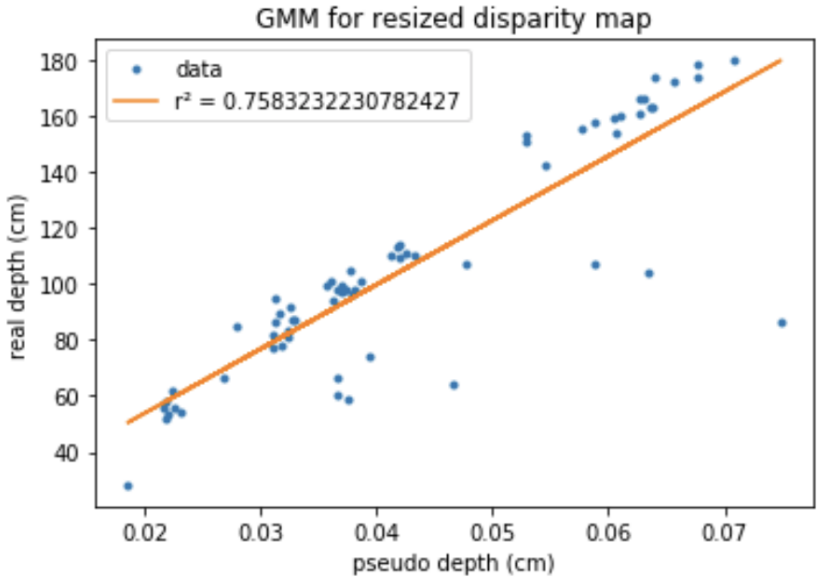


Figure 26: Linear regression using OLS for Model 2.

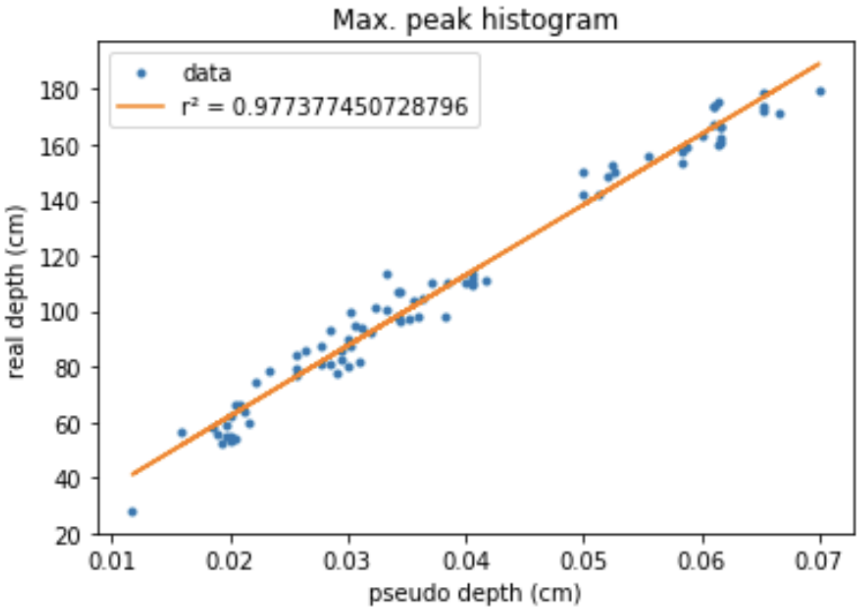


Figure 27: Linear regression using OLS for Model 3.

Model Type	Coefficient (m)	Constant (C)	Prob F-statistic	Adjusted R^2
Model 1	2555.718 ± 97.771	0.569 ± 4.412	$1.59e-35$	0.914
Model 2	2298.072 ± 162.168	7.762 ± 7.357	$2.08e-21$	0.755
Model 3	2536.120 ± 43.138	11.508 ± 1.782	$1.37e-67$	0.977

Table 1: Performance of the models

Looking at the table, we can observe how the model, which has a higher R-squared value corresponds to model 3. Whatsmore, this model has the smallest Probability for the F-statistic, which means that we can reject the null hypothesis.

With these results, we conclude that the best model for proper depth calibration is the Model 3, which corresponds to the Highest peak model. This model is the simplest model we used, and apparently, the best one. Not only due to its better results compared with the other ones but also is the fastest in computation terms.

In contrast, we observe a significative difference between Model 1 and 2 for the R square value. As we observe, model 2 performs the worst. We believe that this lack of performance in these two models is due to the inaccurate approximation to two Gaussian distributions in some cases and the effect of resizing the image in the histogram representation.

5 Integration on the car's prototype

In the previous chapters, we have explained the modeling guideline that this project follows to infer depth to a specific object using a stereo camera and a set of models, which includes a convolutional neural network. From now on, we will discuss how can we import this pipeline into a low energy computational platform which satisfies the following constraints.

- The device used must be powerful and fast enough to perform the inference close to real time.
- The integration of our depth estimation algorithms has to be alongside the bottle detector algorithm, which is currently operative in the car's prototype.

In order to satisfie both constraints, in this chapter, we will explain first what is the best device to carry out these computations. Then, we will comment how we set up our convolutional neural network for retrieving the disparity map and how this network, alongside the models for depth estimation from disparity, are enabled together with the existing software architecture.

5.1 Nvidia Jetson Devices

As we know, the nature of State-of-the-art Neural Network has an extremely high amount of parameters to adjust via back-propagation. As well, a large amount of training data is required to achieve the accuracy we need for our models. Due to the nature of Convolutional neural networks, which are made by a large number of identical neurons, they are suitable for parallelization. This parallelism maps naturally to GPUs, which provide a significant computation speed-up over CPU-only training.

Is because of this fact that we will use embedded computing boards which carry high-performance Nvidia GPU'S.

We chose Nvidia because this company provides several libraries for improving the performance of Neural Networks in GPU. One of these libraries is cudNN, which is made for optimizing the behavior of the GPU when running Deep Neural Networks. Whatsmore, when the network is running the inference step, the operations computed can also be executed in the GPU due to its parallel nature. Again, NVIDIA provides an inference platform accelerator and runtime called TensorRT. TensorRT delivers low-latency, high-throughput inference and tunes the runtime application to run optimally across different families of GPUs.

Since this project starts do not start from zero, we need to implement an integrate our algorithms in an existing computing embedded board, which is the core of the car's prototype. The boards the prototype is using comes the Nvidia Jetson series, specifically, Nvidia Jetson TX2 and Nvidia Jetson Xavier.

Even these boards come from the same series, the architecture is significantly different. In the figure below we can observe the differences mentioned.

TX2 characteristics	Description
CPU	ARM Cortex-A57 (quad-core) @ 2GHz +NVIDIA Denver2 (dual-core) at 2GHz
GPU	256-core Pascal @ 1300MHz
Memory	8GB 128-bit LPDDR4 @ 1866Mhz 59.7 GB/s
Power consumption	7.5W

Table 2: Characteristics for TX2.[8]

Xavier characteristics	Description
CPU	8-core NVIDIA Carmel 64-bit ARMv8.2 at 2265MHz
GPU	512-core NVIDIA Volta @ 1377MHz with 64 TensorCores
Memory	16GB 256-bit LPDDR4x @ 2133MHz 37GB/s
Power consumption	10W / 15W / 30W profiles

Table 3: Characteristics for Xavier.[7]

The table shows that there is a significant difference between boards, so we expect to obtain different performance results when running the algorithms. Concretely, as Nvidia Jetson Xavier has more GPU computing capability, we hope to have a better performance improvement than when using TX2. Nevertheless, there are two main issues related to the usage of these two boards.

Figure 28: Nvidia Jetson Xavier

Figure 29: Nvidia Jetson TX2

The first one is that the prototype carries Jetson TX2 currently, so the test where the car performs the autonomous driving will be only available with this board. However, we could play some other tests using both devices. The last one is that since these computers have a different architecture, the compiling tasks of the code will be different as well, and in some cases, we are not able to perform the same test due to this issue. For instance, for NVIDIA Jetson Xavier, there is not available the option to run the code using 16 fp.

5.2 Convolutional Neural Network set up

Taking into account the above-mentioned points, this first task focuses on the understanding of the algorithm and the familiarisation with it. As we commented above, the algorithm used for reconstructing the disparity map from the stereo camera is a custom architecture of GC-Network and is available from this repository [9], with the following characteristics.

- It has several pretrained algorithms ready for implementation.
- The repository comes with specific compilation tools.
- It comes with little documentation of how to deploy its algorithms.

As we see, this algorithm comes with a complete set of tools to deploy the convolutional neural network in a computing capable device. Also, it comes with already pretrained networks, which means that they provide us the weight files, so there is no need to train it ourselves.

Model	Input size	Titan Xp (TF)	Titan Xp (TRT)	Jetson TX2 (TRT) FP32 / FP16	D1 error (%)
NVSmall	1025x321	800	450	7800 / NA	9.8
NVTiny	513x161	75	40	360 / NA	11.12
ResNet-18	1025x321	950	650	11000 / NA	3.4(*)
ResNet-18 2D	513x257	15	9	110 / 55	9.8

Figure 30: Diferent feature extractors avaiable in the repository

The figure above shows the different algorithms the repository provide us. The table shows the model name, the dimensions of the input image, the time in milliseconds spent by the network at the inference process for different devices(Titan XP and Jetson TX2), different frameworks (TensorFlow and Tensor RT) and different fp modes (32 and 16). Also, in the end, it shows the error evaluated using the KITTI dataset. As we could see, the best choice seems to be the ResNet 18, which has the lowest error rate. However, this error is not useful because this model was fine-tuned on 200 training images so that it will be overfitted. Whatsmore, the time spent by the process is higher compared to the rest of the networks, making the inference slower.

This is why we decided to use the fastest one, which in this case is the ResNet-18 2D architecture. This network is based on the same structure of custom GCC network explained above. However, the difference lies in the feature extractor step. This step is made by residual networks layers developed in 2018, which applies 2D convolutions to the input image.

In order to implement the algorithm for a real-time inference of the depth, first, we need to convert the network provided by the repository, which is built under Tensorflow framework, to Tensor RT.

First of all, we need to explain what is Tensorflow and tensorRT framework before proceeding with the explanation.

On one hand, TensorFlow (TF) is an open source software library for numerical computation using data-flow graphs. It was originally developed by the Google Brain Team within Google's Machine Intelligence research organization for machine learning and deep neural networks research. Its nature allows dealing with heavy numerical computations. TensorFlow is cross-platform. It runs on nearly everything: GPUs and CPUs, including mobile and embedded platforms, and even tensor processing units (TPUs), which are specialized hardware to do tensor math on.[5] This technology is used in this project for the writing of the network in Python, however, it runs C/C++ at the backend, so the performance is better than if it were written in plain python. As well, it supports CPU and GPU processing. As we mentioned, is based on a data flow graph. This graph is made by nodes and edges. The nodes represent a mathematical operation, such as addition, multiplication, or even complex function operations, where the edges represent n-dimensional arrays (Tensors) Once the network is created we can train it and perform tests under this framework in order to ensure that can be staged to the inference step.

On the other hand, TensorRT (TRT) is a library for performance and optimization of convolutional neural networks, such as TensorFlow, this library is applied only in the inference step of the project. TensorRT acts stacking and merging several layers of neural networks coming from TensorFlow code, and allows to run the graph in a Nvidia GPU. This library is essential for deploying a TensorFlow graph since this framework is

[5]Source [15]

not optimized for runtime tasks.

Knowing the characteristics of TF and TRT, the compilation step could be done in the mentioned boards or in some other machine able to perform the compilation tasks. In our case, before starting to work with the Jetson TX2 and Xavier, we decided to compile the Tensorflow code in a dedicated server. Since these kind of servers are not usually prepared for these tasks, to solve all the dependency problems, we used Docker framework.

Docker is a software development platform which uses virtualization technologies. So is a framework which allows running applications in any environment desired for our project, in our case, we simulate the environment and characteristics of Jetson Xavier. These applications run over the so-called containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies and ship it all out as one package[6]

Thanks to this environment, we could easily install the dependencies for converting the TensorFlow code into TRT code. This dependencies were for instance, gcc 5.0, OpenGL, OpenCV, CUDA 9.0, cuDNN 7.5 , TensorRT 4.0 Gtests... The main problem with this libraries is that usually are used by other applications, so, if we could not isolate an environment with this dependencies, we could easily break the rest of the applications that the server runs.

In consequence, this implementation step was done in an Ubuntu 18.04 Server with an NVIDIA Titan X graphics card, so as commented below, using an isolated Docker environment, we could install the dependencies needed. For this conversion from Tensorflow to TensorRT code, the dependencies were TensorFlow Framework cudNN 7.0 and TRT-4.0. Having these dependencies installed, the following step is the conversion of the code. This conversion could be quickly done running a conversion script which is also provided by the repository. Whatsmore, this conversion also allows using a version of 32 fp when the network performs the inference or 16 fp.

Now, having the executable of the network and the weights files, also provided by the repository. We can test it before integrating to the rest of the devices. For testing porpuses, what we can run the executable using the following arguments:

- Stereo pair of images
- Weight file
- output path

These mentioned tests are done to ensure that the Network works appropriately in the desired environment, so for a simple test, we using two images and 32fp based on ResNet 18 2D feature extractor, and the results are the following.

[6]Source [14]

Figure 31: Example of left image of the KITTI dataset

Figure 32: Example of right image of the KITTI dataset

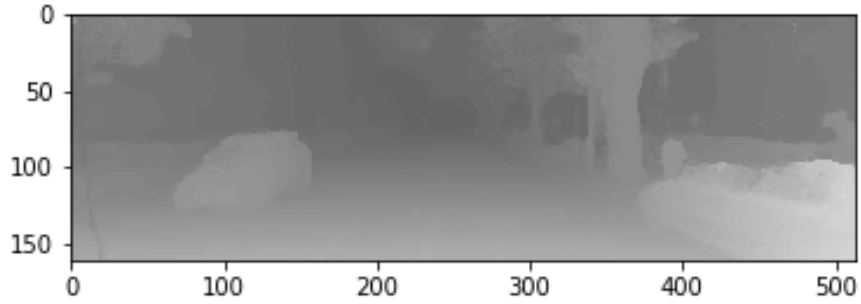


Figure 33: Example of disparity map image outputed by the network

As we observe, both left and right pictures above lead to the disparity map image shown in the last figure. This image maps in grayscale the disparity value from the input images, which leads to getting an approximate depth view. For instance, we can distinguish some objects, such as cars or trees which appear to be closer. However, there are some other objects that the net is unable to retrieve its depth information, such as wall windows or farthest cars. As we observe, both left and right pictures above lead to the disparity map image shown in the last figure. Finally, as a summary, in this setup step, what we only need to do is to compile the code using the proper docker environment or in the proper device, installing the dependencies and declare the correct architecture of the target GPU. For a detailed explanation of how can we achieve this, check our repository.[3]

5.3 Integration on ROS architecture

In this final step, we will discuss which are the following steps to follow in order to have this network integrated over the existing architecture which the Jetson holds. Before the implementation of the algorithm for depth recognition, the prototype stands on a set of scripts and functions that were able to process signals from the camera and then, use these signals for motor communication. Recall that in the introductory section, we explain that what we want to achieve is autonomous driving, so what we need to communicate to the motor are the steering and the throttle.

The old architecture of the car used the position of the bottle to steer the wheels, and the changes on the size of the bounding box for the throttle, where it was assumed that a bigger bounding box leads to a closer bottle. Is in this context where our network needs to be fitted on this assembly. So, the main objective of this step is to change the way the prototype infer the distance from the bottle, as we have been discussing in the previous chapters. Whatsmore, even the old architecture was robust enough; it was not

modular, so another network cannot be deployed in without making several changes to it.

This is the reason why we migrate all the existing architecture to ROS. ROS is the acronym for Robotic Operative System, and, even though is not a fully Operative system, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control,message-passing between processes and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive or post sensor data, control, state, planning, actuator, and other messages.[7] This tool allows controlling the input and output data from each of the algorithms we are using to make a simultaneous inference of the position and depth of the bottle.

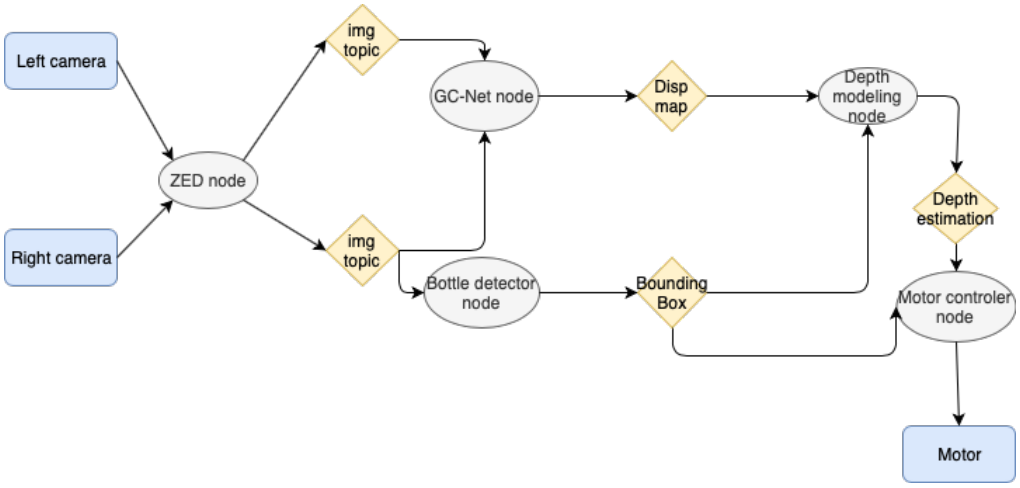


Figure 34: Old ROS architecture.

[7]Source [11]

Looking at the graph above we can take an overview of the ROS architecture with the implementation of our design. As we can observe, there are three kinds of objects in this graph.

- The yellow ones with the squared shape are ROS topics.
- The grey ones, the ROS nodes,
- The blue ones, the hardware.

As we observe, the graph describes a workflow, where the data is processed in each ROS *node* and available for its usage in each ROS *topic*. These nodes are pieces of software, written in any language, such as C, C++, Python... This pieces of software in our case are the CNN, the models used for the depth estimation and the modules for motor communication or camera communication. The nodes loop over time and are fed with data coming from the topics and sent to other topics, from which other nodes can, again, obtain the information. These topics manage the data available using *publisher/subscriber* "protocol". That means that every node can *publish* data in any new topic, and from this topic, other nodes can *subscribe* to this data for receiving it as input.

So, with this assembling, the architecture of the project follows this workflow.

- Each camera sends its input to the ZED node. This node publishes the left and right images into left and right image topics
- From left and right topics, GC-network node subscribes to obtain left and right images. At the same time, the bottle detector node subscribes to the left image topic. At the end of this iteration, GC-network node publishes the disparity map, and the bottle detector node the *bounding box*.
- From the disparity map node, the depth modeling node uses this map and the bounding box for depth estimation which is published into the depth topic. [8]
- Finally, the car controller node uses the information coming from the depth estimation topic and from the bounding box for computing the steering and the throttle.

For deploy this architecture, the first thing we have to do is set up the stereo camera and the nodes corresponding to it. In this project, we used the ZED camera from StereoLabs. ZED is a binocular camera which comes with a set of tools and support for ROS. Thanks to this, we can be able to awake a ROS node which can capture images from right and left cameras and publishing it to a topic.

[8] Recall that the depth modeling node uses the *Highest Peak Model* for depth estimation.

Figure 35: Image of ZED camera.

Once this node is available, we can set up the GC-Network node. For launching this node we can use the files provided by [9]. The source code provided consist of two main files.

On the one hand, it provides a C++ program that creates the ROS node for the depth inference and runs the executable compiled in the previous chapter. On the other hand, an xml file for setting the parameters of the node, such as the name of the network, the path to the weight files, the maximum frequency of the output or the subscriber nodes for this algorithm. Nevertheless, this repository does not provide information about how to deploy the ROS architecture, nor how to awake the rest of the necessary nodes, such as the camera node.

As a result of this architecture, we can perform a depth inference from a given scene in live. As we can see in the following figure, using the mentioned CNN, we can achieve this disparity map from the following images.

Figure 36: Example of left image running in NVIDIA Jetson TX2 and taken with ZED camera

Figure 37: Example of left image running in NVIDIA Jetson TX2 and taken with ZED camera

Figure 38: Example of disparity map produced by the DNN node running under NVIDIA Jetson TX2

Finally, the last step is to implement the existing bottle detector node, depth modeling node, and the car controller node for achieving a fully operative autonomous driving car which follows bottles.

Figure 39: Final design for the car's prototype.

6 Performance analysis and quality results

From now on, we have an operative car, but since we are interested in autonomous driving, we need to test the *frequency* at which the algorithms run for positioning the bottle in the 3D space, and the accuracy of this positioning. We test the custom GC-network with ResNet-18 for feature extraction using 32 and 16 floating-point arithmetic, respectively. These two networks have been tested on NVIDIA Jetson TX2 and Xavier, respectively, and using different ROS architectures.The first architecture represents our GC-Network algorithm running alongside. This experiment has been done as a benchmark for our network when using different devices and arithmetics. Afterward, we perform the same tests but with the full ROS architecture explained in the previous section. The following pictures show in red the interesting topic where the frequency is tested.

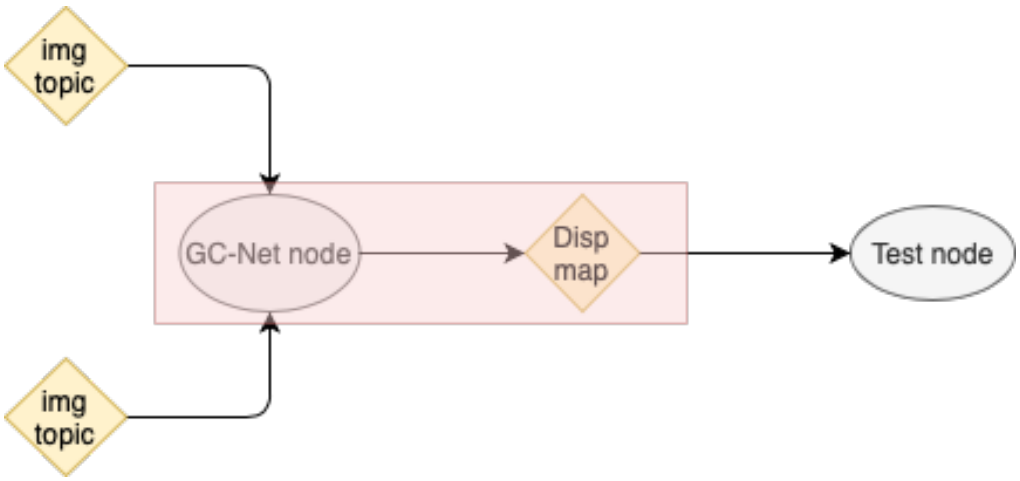


Figure 40: Graph of ROS architecture of GC-network. Frequency tested for the red colored node/topic.

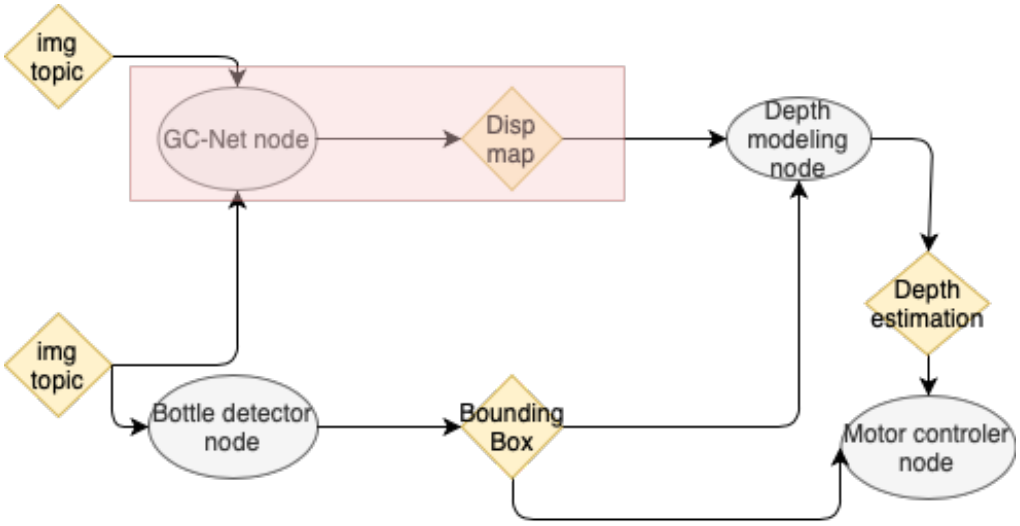


Figure 41: Graph of ROS architecture of bottle positioning. Frequency tested for the red colored node/topic.

FP arithmetic	ROS architecture	GPU usage (%)	Freq. rate (Hz)
FP16	Depth recognition only (Benchmark)	99 at 1122 MHz	5.6
FP32	Depth recognition only (Benchmark)	99 at 1122 MHz	3.6
FP16	Depth recognition + Bottle detection	99 at 1122 MHz	3.0
FP32	Depth recognition + Bottle detection	99 at 1122 MHz	1.5

Table 4: Summary for TX2.

As we can see, the best performance is or the algorithm running the 16 fp arithmetic. However, the GPU usage is always the maximum no matter which is the algorithm mode or architecture. As we can observe, the best frequency achieved for the network is 3.0 Hz. Whatsmore, the performance of the full architecture is about 53 % less fast when using the fp 16 arithmetic and about 41 % when using the 32 fp. The same we can do for Xavier, however in this case we cannot compute using 16fp.

FP arithmetic	ROS architecture	GPU usage (%)	Freq. rate (Hz)
FP32	Depth recognition only (Benchmark)	99 at 905 MHz	11.0
FP32	Depth recognition + Bottle detection	99 at 905 MHz	5.4

Table 5: Summary for Xavier.

As we observe, the improvement using Xavier over TX2 is significant. In this configuration, we multiply by a factor 3.6 the performance of the bottle positioning when using 32fp. With a drop of improvement with respect to the benchmark of 50% approximately.

Once we finish with the performance tests, we need to evaluate the quality results. That is, we need to check that the error measure when using both arithmetics and devices. In this evaluation, we assume that if the disparity map is properly taken, the real distance from the car to the bottle is computed using *Model 3*, and the error of the measure is obtained using the error propagation formula.[9]

However, due to several causes, not all cycles the network is able to capture the proper bottle section in the disparity map. These inefficiencies lead to wrong records for the depth estimation. Consequently, in order to assess the error measures, we recorded each depth measure for a certain period while the bottle remains fixed. Analyzing these measures we can evaluate the error interval for depth estimation in real time.

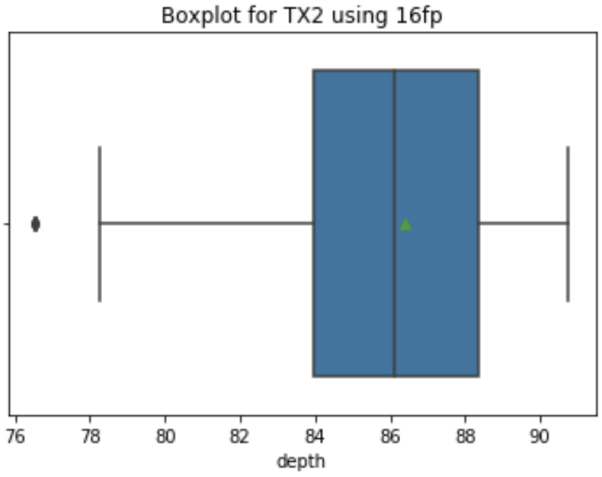


Figure 42: Boxplot for asses depth accuracy using TX2 with 16fp.

[9]error propagation forumla : $\epsilon^2 = \sum_{i=1}^{N} (\frac{\delta f_{(X_i)}}{\delta X_i} \Delta X_i)^2$

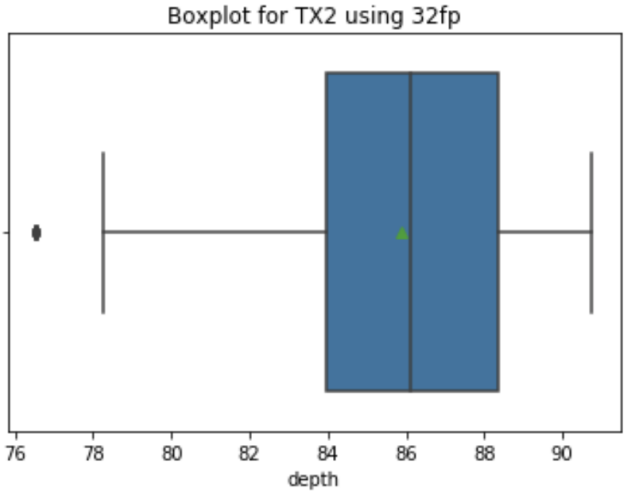


Figure 43: Boxplot for asses depth accuracy using TX2 with 32fp.

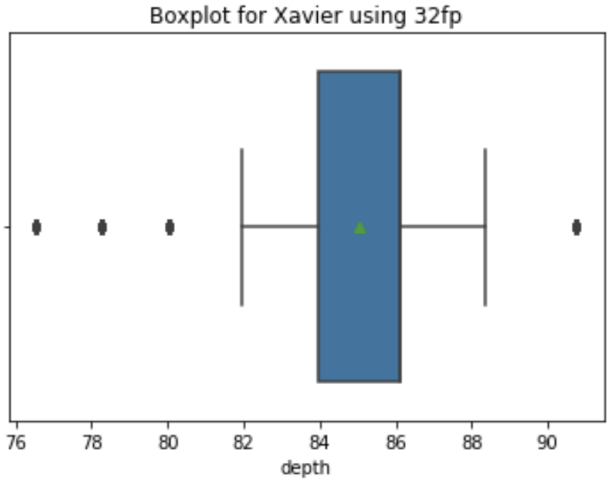


Figure 44: Boxplot for asses depth accuracy using TX2 with 16fp.

The boxplots shown above are obtained recording the distance to a bottle at about 85 cm during 2 minutes. As we can observe in the pictures below when using the TX2 the interquartile range are almost the same, this means that change the arithmetic of the assembling seems that do not affect the accuracy of the depth error.

As for TX2, the experiment shows that the mean measure of the bottle is near 86 cm, which what we expected. As well for the Xavier, the measure is slightly different we made the experiment on a different day, so the position of the bottle may not be the same. However, we can observe that the mean is located approximately at the same distance.

As we can see, in any case, shows a normal distribution, these boxplots show that the error measurement is asymmetric, where we have more depth records which appear closer to the car than further.Whatsmore, both boxplots present some outliers. Due to this fact, we could asses that the error measure could be at least about -10 cm from

the mean in the case of TX2 and -2 cm for the Xavier. With these results we can see how using TX2 may lead to more inaccurate depth lectures due to this asymmetries and high confidence intervals. However, again, we have significant improvement when using Xavier.

All in all, Nvidia Jetson Xavier performs better in both aspects. It is faster and more accurate than Nvidia Jetson TX2. The improvement over the velocity of the network using Xavier is expected due to its compute capabilities. This is because this device was developed explicitly for Deep Learning applications. Moreover, the improvement over the depth error is quite astonishing. We think that this improvement is due to the fact that being faster leads to a better match between the bounding box section and the cropped area of the disparity map. However, for the porpuses of this thesis, we can ensure that Nvidia Xavier is a better option for depth estimation.

7 Conclusions

In conclusion, in this master thesis, we focus on two main topics in order to achieve autonomous driving. These topics are: Which models can we use for positioning an object in 3D space, and more concretely, how we asses depth. And, how we can implement these models in a capable device for achieving these tasks. For answering the first question, we have seen that there exists a powerful machine learning tool called Deep Learning. With these models, we are able to extract meaningful information from a scene using a stereo camera in our case. However, due to the restrictions of our assembling, the output of this Deep learning model has to be modeled as well. Having this, we propose three different models in order to assess depth estimation from the disparity map. These models are based in the histogram representation of this disparity map and we discover that we can obtain impressive results for depth inference when using the so-called *Highest Peak Model*. However, this model is strongly dependent on the geometric characteristics of the object of interest[10].

Apart from this, we have been able to integrate these models in an existing car's prototype, so, we update the way this car interacts with the environment, acquiring a better depth estimation using our proposed models. In this context, we deployed a new software architecture for this porpuses and we tested the performance and the quality of the results in real time. What we observe is that even the car still has a slow frequency rate for depth estimation, it is able to perform simple self-driving tasks at slow velocities. The task performed by the car at the end of this project is to identify a bottle and follow it. Nevertheless, this assembling still has certain lacks what needs to be solved for better improvement on this driving tasks. For instance, the CNN for depth recognition is strongly dependent on the light conditions of the scene, and this leads to wrong results in depth inference. Whatsmore, the integration of the two algorithms should be done in a single network, which should be able to identify objects and extract the distance from it. However, by this time, these kinds of deep learning networks do not exist despite the vast improvements in object recognition in real time, such as YOLO detections [16]. All in all, even the difficulties of an accurate estimation of this distance measurement, we find that self-driving using stereo vision is achieving significative results and we think that in future research a real autonomous driving will be reach.

[10] In our case, a bottle

References

[1] Backpropagation. https://en.wikipedia.org/wiki/Backpropagation. Accessed: 2019-06-27.

[2] Convolutional neural network. https://en.wikipedia.org/wiki/Convolutional_neural_network. Accessed: 2019-06-27.

[3] Depth estimation project with jetson tx2. https://github.com/FaradayDetu/DepthEstimation. Accessed: 2019-07-01.

[4] Gaussian mixture model. https://brilliant.org/wiki/gaussian-mixture-model/. Accessed: 2019-05-21.

[5] Jetson inference. https://github.com/dusty-nv/jetson-inference. Accessed: 2019-06-27.

[6] Neuron. https://en.wikipedia.org/wiki/Neuron. Accessed: 2019-06-27.

[7] Nvidia jetson agx xavier delivers 32 teraops for new era of ai in robotics. https://devblogs.nvidia.com/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/. Accessed: 2019-06-21.

[8] Nvidia jetson tx2 delivers twice the intelligence to the edge. https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/. Accessed: 2019-06-21.

[9] Nvidia redtail project. https://github.com/NVIDIA-AI-IOT/redtail/tree/master/stereoDNN. Accessed: 2019-03-19.

[10] Point cloud. https://en.wikipedia.org/wiki/Point_cloud. Accessed: 2019-06-27.

[11] Ros. https://en.wikipedia.org/wiki/Robot_Operating_System. Accessed: 2019-06-27.

[12] Sagar sharma.what the hell is perceptron? https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53. Accessed: 2019-06-27.

[13] Sagar sharma.what the hell is perceptron? http://neuralnetworksanddeeplearning.com/chap5.html. Accessed: 2019-06-27.

[14] What is docker? https://opensource.com/resources/what-docker. Accessed: 2019-04-15.

[15] What is the tensorflow machine intelligence platform? https://opensource.com/article/17/11/intro-tensorflow. Accessed: 2019-03-29.

[16] Divvala S. Girshick R. Farhadi A. Redmon, J. You only look once: Unified, real-time object detection. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[17] F. Sanja. Depth from stereo. Sanja, F.,http://www.cs.toronto.edu/~fidler/slides/2015/CSC420/lecture12_hres.pdf.

[18] Kamenev A. Birchfield Smolyanskiy, N. On the importance of stereo for accurate depth estimation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018.