Universidade de Aveiro Departamento de Engenharia Mecânica
2020

Rúben Daniel Ferreira da Costa

Detection and classification of road and objects in panoramic images on board the ATLASCAR2 using Deep Learning

Deteção e classificação de estrada e objetos em imagens panorâmicas a bordo do ATLASCAR2 usando redes com Deep Learning

Universidade de Aveiro Departamento de Engenharia Mecânica
2020

Rúben Daniel Ferreira da Costa

Detection and classification of road and objects in panoramic images on board the ATLASCAR2 using Deep Learning

Deteção e classificação de estrada e objetos em imagens panorâmicas a bordo do ATLASCAR2 usando redes com Deep Learning

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob orientação científica de Vítor Manuel Ferreira dos Santos, Professor Associado com Agregação do Departamento de Engenharia Mecânica da Universidade de Aveiro, e de Miguel Armando Riem de Oliveira, Professor Auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro.

Este trabalho foi desenvolvido com recursos do IEETA - Instituto de Engenharia Eletrónica e Informática de Aveiro, no âmbito do Projeto UIDB/00127/2020, e usou fundos comparticipados pelo DEM/TEMA - Centro de Tecnologia Mecânica e Automação no contexto do Projeto UIDB/00481/2020, ambos financiados pela FCT - Fundação para a Ciência e Tecnologia.

o júri / the jury

presidente / president

Prof. Doutor Marco Paulo Soares dos Santos
Professor Auxiliar Convidado da Universidade de Aveiro

vogais / committee

Doutor Paulo Jorge Sequeira Gonçalves
Professor Coordenador do *Instituto Politécnico de Castelo Branco*

Prof. Doutor Vítor Manuel Ferreira dos Santos
Professor Associado com Agregação da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Em primeiro lugar, gostaria de agradecer ao professor Vítor Santos por me ter orientado durante este semestre, transmitindo sempre confiança e interesse pelo trabalho desenvolvido, e ao professor Miguel Riem pelo espírito crítico transmitido.
Um agradecimento especial aos meus pais por todo o apoio e motivação dados ao longo do meu percurso académico e à Margarida pela confiança, incentivo e também pela paciência demonstrada durante todos estes anos.
Gostaria de agradecer também aos membros do LAR, sobretudo ao Bernardo e ao Tiago pela ajuda fundamental, conhecimentos transmitidos e disponibilidade demonstrada ao longo da dissertação, ao Diogo pelas conversas e paciência durante os testes no ATLASCAR2, ao Rui pelas dúvidas tiradas e ao Engenheiro Rui Heitor por se mostrar sempre prestável a ajudar.
Por fim, mas não menos importante, um agradecimento ao Limas e ao Rui por todas as aventuras durante estes 5 anos, à Lurdes pela amizade e sessões de estudo, ao Padrinho por aceitar todos desafios e ao Pedro pelo rigor incutido.

keywords

Autonomous Driving; Deep Learning; Panoramic Images; Image Segmentation; Road and Object Detection.

abstract

The field of autonomous driving has been increasingly explored and the future of transport partly depends on the use of this type of vehicle. For an autonomous car to navigate on the public road, it must be able to detect everything around it, ensuring that the actions taken do not compromise the safety of any person. Within the Atlas project, this dissertation aims to create a model that allows the detection of road and objects in panoramic images, thus increasing the ATLASCAR2 field of view. In view of this need, a system was developed for the creation of panoramic images through images acquired by the cameras mounted on the car and, to make the detection of the road and other objects, deep learning was used to train the models in order to ensure great accuracy and detail in detection. This work presents the results obtained with the trained models, presenting a comparison between the use of different architectures and datasets. In addition, an evaluation of the capacity of these models was also performed in the city of Aveiro.

palavras-chave

Condução Autónoma; Deep Learning; Imagens Panorâmicas; Segmentação de Imagem; Deteção de Estrada e Objetos.

resumo

A área da condução autónoma tem sido cada vez mais explorada e o futuro dos transportes passa, em parte, pela utilização deste tipo de veículos. Para conseguir navegar na via pública, um carro autónomo deve ser capaz de detetar tudo o que o rodeia, garantindo que as ações tomadas não põem em causa a segurança de ninguém. No âmbito do projeto Atlas, esta dissertação prevê a criação de um modelo que permita a deteção de estrada e objetos em imagens panorâmicas, aumentando assim o campo de visão do ATLASCAR2. Tendo em vista esta necessidade, foi desenvolvido um sistema para a criação de imagens panorâmicas através de imagens adquiridas pelas câmaras montadas no carro, e para fazer a deteção da estrada e de outros objetos, recorreu-se a "deep learning" para treinar os modelos, de forma a garantir grande precisão e detalhe na deteção. Neste trabalho são apresentados os resultados obtidos com os modelos treinados, apresentando uma comparação entre a utilização de diferentes arquiteturas e "datasets". Para além disso, também foi realizada uma avaliação da capacidade destes modelos na cidade de Aveiro.

Contents

List of Tables

Intentionally blank page.

List of Figures

Intentionally blank page.

Listings

Intentionally blank page.

Chapter 1

Introduction

1.1 Motivation

Nowadays, more than 90% of accidents in the European Union are caused by human error [1] and a large part of the population spends more than 20 hours in road congestions each year [2]. In this context, the development of the field of autonomous driving (AD) is essential to minimize road accidents, increase productivity, reduce congestion and, consequently, carbon emissions. To achieve all these objectives, vehicles must be equipped with a diverse set of components that allow them to perceive, detect and make decisions, taking into account what is happening in their environment.

While for persons their side and rear mirrors help to increase the field of view, in a stand-alone vehicle it is necessary to "replace" these with cameras in order to have a visual perception of all the space around it. In addition, all images must be processed in real time, so that the vehicle can travel without compromising anyone's safety. At the moment, there are only partially autonomous vehicles since the human being has to supervise all the actions the car takes in order to ensure its own safety.

1.2 The Atlas Project

The study of this dissertation is part of the Atlas project. This project was created by the Group of Automation and Robotics at the Department of Mechanical Engineering of the University of Aveiro [3], and aims to develop a system that allows the autonomous driving of a car on the road. The project started with several participations in the National Robotics Festival, for which several autonomous robots were developed that earned several victories in the competition.

After the knowledge and success achieved, the Atlas project decided to start the development of ATLASCAR1, shown in figure 1.1, by equipping a Ford Escort with several sensors and making modifications to its hardware, so that it could explore the environment around it.

After some years of research and development of new sensory systems and algorithms, the Atlas project replaced the ATLASCAR1 with an electric vehicle, a Mitsubishi i-Miev, calling it the ATLASCAR2. In figure 1.2 it is possible to see the vehicle with several sensors mounted such as LIDAR sensors, cameras and GPS.

Figure 1.1: ATLASCAR1

Figure 1.2: ATLASCAR2

1.3 Problem Description

With the development of technology, new techniques of road and object detection have been used, and one of the main methodologies has been the application of Deep Learning models. In previous years, systems capable of detecting objects and people using these techniques have been developed within the Atlas project, as well as various systems for detecting navigable limits using classic image processing techniques or processing data collected by LIDAR sensors.

Despite all the work that has already been done, there is still a need to expand the field of view of the ATLASCAR2, making detections on the entire periphery of the car. A ring of cameras is being developed to mount on the roof of the ATLASCAR2, in order to get a panoramic view of the environment, and through this hardware it is possible to make detections on the entire periphery of the car. With panoramic images it is possible to better define the actions that the car can perform, especially in situations

of intersections, where a front camera cannot see the road on the sides or obstacles coming from them. Another advantage of the panoramic images is the possibility of following obstacles in motion around the car. Accompanying the need to expand the field of view, it is also necessary to create a more robust system for detecting road and obstacles, using models with Deep Learning, being the main challenge of this dissertation the segmentation of panoramic images.

1.4 Objectives

As mentioned in the previous section, one of the major needs of the project is to increase the field of view of ATLASCAR2. Thus, the main objective of this dissertation is the creation of a model that allows the identification of road and objects in panoramic images. To achieve this purpose, the following intermediate objectives will have to be achieved:

- Make the acquisition of panoramic images from the camera ring;
- Select and propose architectures and datasets for the network training, road and targets detection and segmentation in real time;
- Select the suited hardware and software required to run the model;
- Implement the model and test it in a real case.

1.5 Document Structure

This document is divided into 6 chapters. This first chapter presents the context of the problem, its motivation and the main objectives. Chapter 2 describes the state of the art of this type of technology and some work developed within the Atlas project or similar projects. The hardware and software used during this dissertation are presented in chapter 3 and chapter 4 describes the solution developed during the dissertation. Chapter 5 discusses the results obtained during the experiments. Finally, chapter 6 presents the conclusions and the work that can be developed in the future.

Intentionally blank page.

Chapter 2

Related Work and State of the Art

As technology advances, models for road and object detection have made great progress. This chapter explores the development of these techniques and what is currently the state of the art in this area. In addition, related work developed at LAR and other projects considered interesting in this context are presented.

2.1 Classic Object Segmentation and Detection

Before the appearance of deep learning techniques, object and road detection were based on machine learning algorithms in which an engineer had to select the best features for the algorithm. For road detection, methods used classic techniques such as Canny filters and Hough transforms [4] or through the variations of color existing in the road to detect edges or lane marks [5]. However, there was great difficulty in detecting the road limits in situations where there were no markings, or where the pavement was not homogeneous.

In object detection, common methods used were Histogram of Oriented Gradient (HOG) [6], Scale Invariant Features (SIFT) [7] or Haar-Like features, among others. However, despite the capacity of these methods, they require feature engineering to obtain good performance and are very focused for a given context.

In 2012, a convolutional neural network architecture was proposed [8] that obtained better results than the previous techniques in the ImageNet LSVRC-2012 contest and since this date, and with the constant technological evolution, the use of deep networks has been growing more and more. In the next section some general concepts about this technology will be presented.

2.2 General concepts of Deep Learning

In recent years, deep learning has become the most widely used technique for computer vision [9], speech recognition [10] or Natural Language Processing [11] problems. Deep Networks have already shown great performance in this type of challenges, achieving better performance with larger amounts of data provided to train the network. Before showing some applications of this type of networks some general concepts are presented in the next subsections.

2.2.1 Artificial Neural Networks

Neural networks were first developed in the 1950s inspired by the behaviour of the human brain, since they are able to interpret the context of real world situations in a way that computers cannot. An artificial neural network is an attempt to simulate the network of neurons that form a human brain so that the computer is able to learn things and make decisions in a similar way to humans [12].

Typically, an artificial neural network has many of units called perceptrons. A perceptron receives a set of inputs that are multiplied by weights and then added. After that, a bias is added to this sum and a net weighted sum is obtained, which is given to an activation function that then produces an output as shown in figure 2.1. These perceptrons are grouped in several layers, giving rise to a network with multiple layers [12].

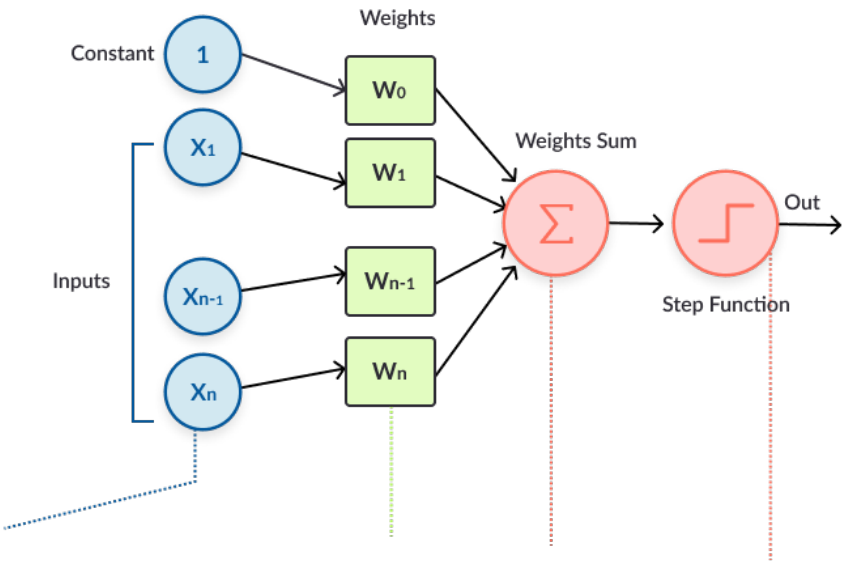


Figure 2.1: Perceptron model [13].

The input layer receives a data input, which the network must process or learn about. From the input unit, the data passes through several hidden layers, where neurons receive a series of inputs from the previous layer and produce an output for the next one. This process is repeated until the output layer obtains the result for the input that was given to the neural network.

2.2.2 Activation functions

As mentioned before, the activation function receives the weighted sum of a neuron and determines the output from it, usually between 0 and 1 or -1 and 1. There are numerous activation functions, which can be divided into two large groups: linear and non linear activation functions. Most neural network models use the non-linear activation function, since they provide the model with the ability to learn and perform more complex tasks [14]. Some of the most frequently used activation functions are shown next.

Sigmoid function

The logistic sigmoid function is commonly used in neural networks trained by back-propagation algorithms. One of the advantages of this function is its smoothness and its range is between 0 and 1. The formula for this function is

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{2.1}$$

and its curve is represented in figure 2.2.

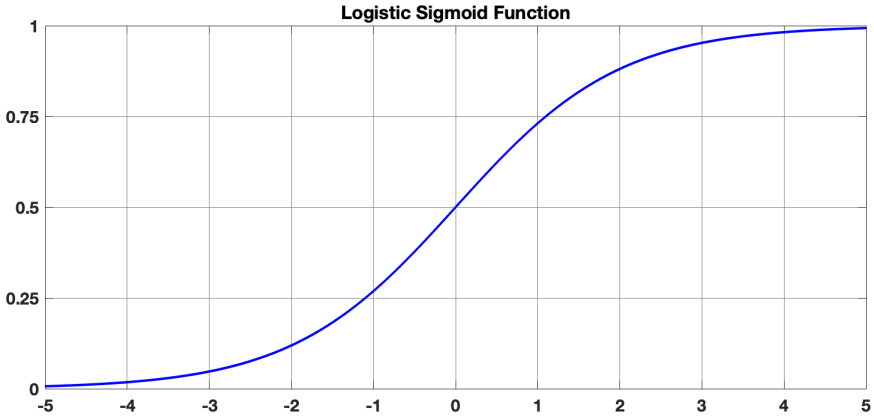


Figure 2.2: Graphical representation of Logistic Sigmoid function.

Hyperbolic Tangent Function

The hyperbolic tangent function, also known as the tangent transfer function, is similar to the logistic sigmoidal function. The biggest difference from this one to the previous one is that it is zero centered and its output values are between -1 and 1, giving better training performance for multi-layer neural networks [14]. Figure 2.3 shows the hyperbolic tangent function graphical representation.

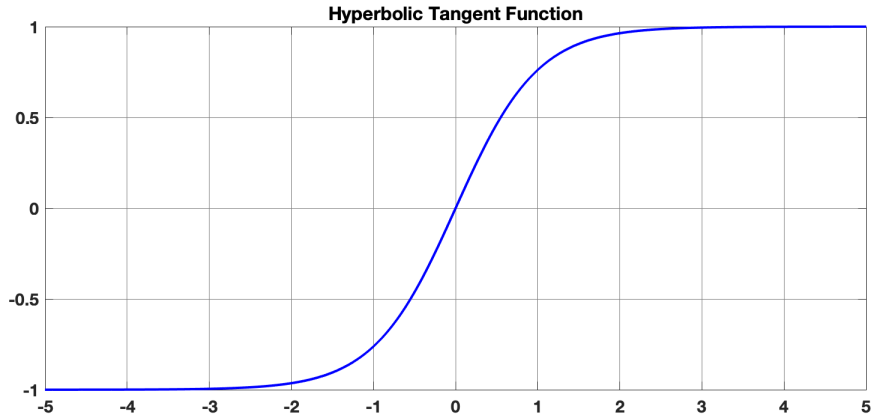


Figure 2.3: Graphical representation of Hyperbolic Tangent function.

Rectified Linear Unit Function

The rectified linear unit activation function has been the most commonly used for Deep Neural Networks and it has several variations. The ReLU function is computed as shown in equation 2.2, and in equation 2.3 is presented the formulation of one of its variations, the Leaky ReLU.

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{2.2}$$

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases} \tag{2.3}$$

The main difference between them is the α parameter, which was introduced as a solution for gradients not to be zero during training. These functions enable a faster training process and achieves better performance and generalization than the functions mentioned previously [14].

Parametric rectified linear was proposed in 2015 [15] , with a similar formulation to Leaky ReLU, but with this new function it is possible to back-propagate and learn the most appropriate value for α, allowing an improvement of performance when compared to previous versions. Figure 2.4 shows the graphical representations of these functions.

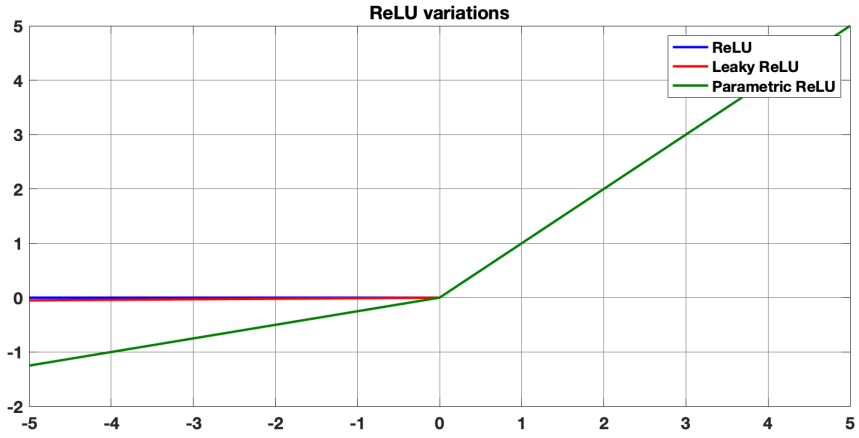


Figure 2.4: Graphical representations of different ReLU functions.

Normalized Exponential Function

The Normalized Exponential Function, most known as Softmax function, is commonly used in the output layers of the deep learning architecture to classify inputs into multiple categories. This function normalizes the outputs for each trained class between 0 and 1, and divides by their sum, giving the probability of the input value being in a specific class.

Swish function

The Swish function was proposed in 2017 [16], and its formulation is as follows:

$$f(x) = x \cdot \sigma(x), \tag{2.4}$$

where $\sigma(x)$ in this function is the sigmoid function presented earlier. The authors mention as main advantages its smoothness when compared to ReLU and that it does not suffer vanishing gradient problems. The authors also reported that Swish performs better than ReLU with a similar level of computational efficiency, and in some experiments the replacement of ReLU with Swish units improved top-1 classification accuracy on ImageNet. The graphic representation of this function is shown in the figure 2.5

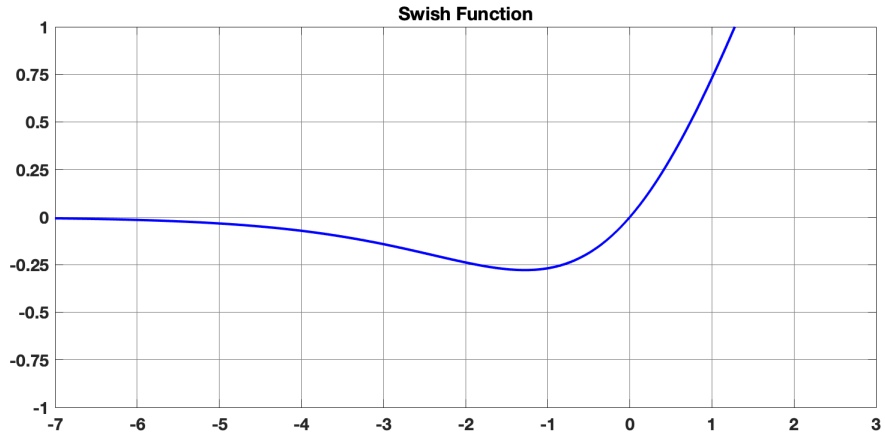


Figure 2.5: Graphical representations of Swish function.

2.2.3 Loss functions

The loss functions quantify the difference between reality and predicted into a single number, where that number is the error from the network's predictions [12]. The goal during training is to find the parameters that minimize the "loss", turning this process into an optimization problem. In the following functions i represents the pixels in each image and N the total number of pixels. The variable c represents each class and the total number of classes are represented with C. The $\hat{y}$ represents the ground truth and y the prediction of the model.

Cross Entropy

The Cross Entropy loss is the most commonly used function for multiclass classification. First, this function calculates the loss by making a pixel-wise evaluation of the predicted probabilities of all classes and then averages over all pixels [17]. This function is computed as follows:

$$CE(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_i^c \log(\hat{y}_i^c) \tag{2.5}$$

Balanced Cross Entropy

Since in the cross entropy loss function each pixel has the same weight in the loss calculation, the balanced cross entropy function adds a weight, w_c, depending on the class frequency in the dataset [17]. With this, problems related to class imbalance are reduced. The loss function is represented in equation 2.6.

$$BCE(y, \hat{y}) = -\frac{1}{N}\sum_{I=1}^{N}\sum_{c=1}^{C} w_c y_i^c \log(\hat{y}_i^c) \tag{2.6}$$

Soft Dice Loss

The soft dice coefficient loss function is commonly used in segmentation problems and is formulated as:

$$SD(y, \hat{y}) = 1 - \sum_{c}^{C}\left[\frac{2\sum_i^N y_i^c \hat{y}_i^c}{\sum_i^N y_i^c + \sum_i^N \hat{y}_i^c}\right] \tag{2.7}$$

This function is based on the Dice coefficient, which is a measure of overlap between two samples, in this case between the prediction and groundtruth [18].

Focal Loss

This function aims to eliminate problems related to class imbalance by introducing a factor to the cross entropy function as presented in equation 2.8. With this factor, the function increases the impact of miss-classified examples, focusing more on these cases during training.

$$FL(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}(1 - y_i^c)^{\gamma} y_i^c \log(\hat{y}_i^c) \tag{2.8}$$

Online Hard Example Mining Loss

In Online Hard Example Mining, the examples are scored for their loss. After this, non-maximum suppression (NMS) serialization is applied, which filters the highest loss examples and builds a mini-batch with them. Like the focal loss, OHEM focuses more on these difficult examples and forces the network to train on them, ignoring the easy examples. In the article [19], the influence of different loss functions on the training of an image segmentation model was compared, and the model trained with this function obtained good results in classes such as road, car and sidewalk in Cityscapes Validation set.

2.2.4 Optimization

In the last subsection some loss functions that allow us to calculate the error between what the model detected and the actual value are presented. However, to reduce this error as much as possible, it is necessary to make consecutive improvements in each iteration, making the training process an optimization problem. This consists in defining

which are the best network parameters that give a prediction closest to the real, minimizing the value of the loss function. Next are presented some algorithms and parameters that help optimize the training process.

Gradient Descent

Since loss functions depend on a large number of parameters, it is usual to opt for iterative methods to train the models. In the Gradient Descent method the cost function parameters are initialized, and each iteration is updated in the opposite direction to the gradient since this will ensure the greatest possible decrease in the loss [12].

However, the parameters are only updated at the end of each epoch, i.e., after it go through the whole training dataset, which increases the convergence time when using very large datasets.

Stochastic Gradient Descent

The Stochastic Gradient Descent is similar to the previous method but instead of going through the whole training set to update the parameters, only one data point at a time is considered, cycling through the training data [12]. This method ends up converging faster than the previous one however its convergence course tends to be more irregular as shown in figure 2.6.

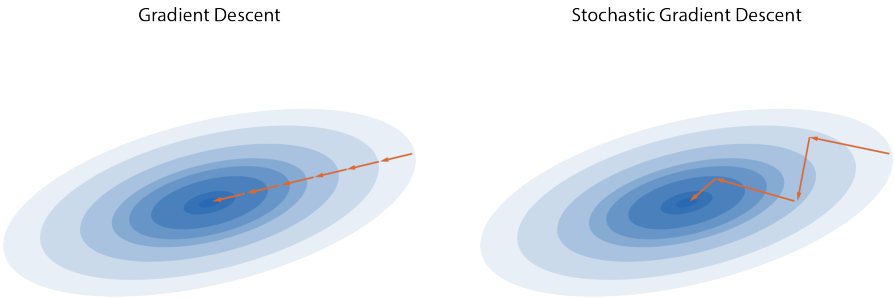


Figure 2.6: Comparision between convergence of Gradient Descent and Stochastic Gradient Descent.

Besides the optimization functions, there are several parameters that can be changed so that the training of the network is better and more efficient. Some of these parameters are described below.

Epoch

An epoch is when the entire dataset is passed through the neural network [12] and to train a model it is necessary to do it more than once. If the number of epochs is too low the model will underfit, not being able to adapt to the training dataset, leading to poor performance. However, if the number of epochs used for network training is too high, the model will overfit, since it learns to identify only the data for which it was trained, and is unable to generalize. A graphical comparison of these situations are shown in figure 2.7.

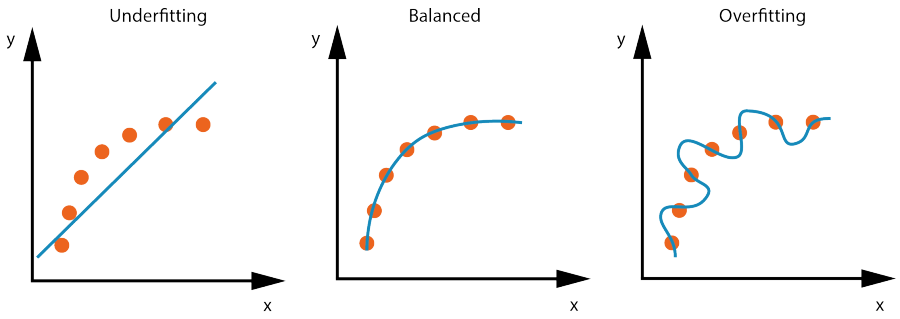


Figure 2.7: Influence of epochs on training process. The dots represent the values to which an approximation is intended, while the blue line corresponds to the model.

Learning rate

The learning rate affects how fast a network adjusts parameters during the training process in order to minimize the loss [12]. A high learning rate decreases the time of the training process but may not converge because the variations the parameters suffer are too large, while a learning rate that is too low should converge to a minimum but takes too long to achieve it, being a computationally heavier process. An illustration of these situations is shown in figure 2.8.

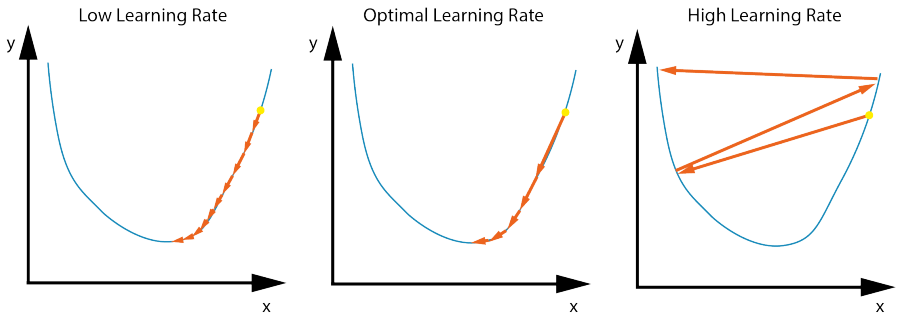


Figure 2.8: Influence of learning rate on training process. The blue line represents the function to be minimized, while the arrows represent the steps taken at each iteration during the training process.

In addition, it is also common to use learning rate schedulers in order to improve performance and reduce the training time. Usually higher learning rates are used at the beginning of the training process, so that the model learns the weights, and then the learning rate is progressively reduced in order to fine-tune them.

Weight Decay

To solve complex problems networks with a large number of parameters are needed. In order to have models with a lot of parameters but trying to decrease complexity, weight decay was introduced.

To penalize the complexity of the model, weight decay is multiplied by the sum of the squares of all the parameters of the network. This prevents the weights from growing too much, unless it is really necessary to do so.

The aim of the optimization process is to decrease the difference between the reality and what was detected. To quantify the capacity of a model on a given dataset it is often used the mean Intersection over Union (mIoU) metric that is presented next

Intersection over Union

Intersect over Union (IoU) is a metric that allows us to assess how similar model prediction and ground truth are. This metric is the ratio between the intersection area of the prediction and the ground truth and the area of their union. To evaluate the quality of the models the mean intersection over union (mIoU) is used, which is calculated by averaging the IoUs obtained in the classes used. In figure 2.9 this metric is presented more clearly.

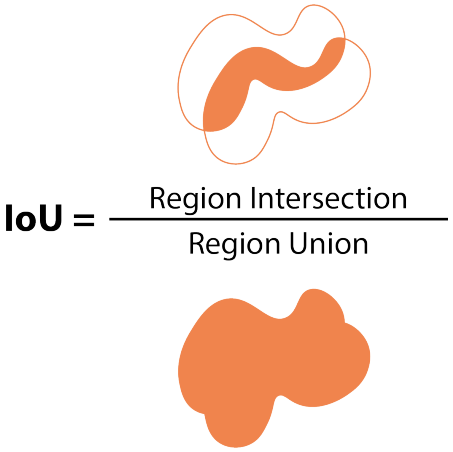


Figure 2.9: Representation of the intersection over union formula.

2.2.5 Convolutional Neural Networks

The Convolutional Neural Networks (CNNs) are powering major advances in computer vision, demonstrating great ability to solve problems like object recognition and image classification, being also able to solve problems related to natural language processing and at analyzing sound. This type of neural networks appeared in the 90s, inspired by the visual cortex in animals. With CNNs it is possible to organize the neurons in three dimensions, being them the width and height of the image, and the third dimension the depth of the image, which in this case is associated to the RGB channels of the image [12]. A general architecture of a convolutional neural network has 3 major groups, such as an input layer, feature extraction layers and classification layers. The input layer receives the raw input data of the image, the feature extraction layers use convolutional layers followed by pooling layers in order to find the features of the images and the classification layers receive these higher-order features and convert them into probabilities of each class.

Convolutional Layers

Convolutional layers are the core of CNNs since the convolution operation extracts the features from the images. A convolutional layer receives an input, applies a convolution

kernel (or filter) and returns a feature map (or activation map) as output. The application of the kernel is made through the width and height of the input volume in a sliding manner and an activation map unit is activated when the filter detects a feature [12]. Figure 2.10 shows some steps of this process.

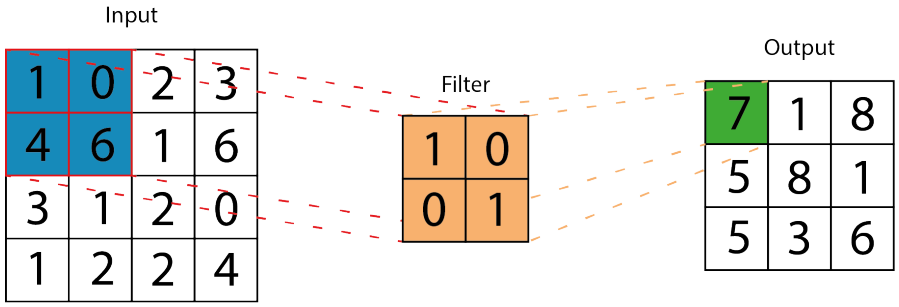


Figure 2.10: Application of a filter in a convolutional layer.

Pooling Layers

Generally, pooling layers are used after convolutional layers and are used to reduce the dimensions of activation maps. These layers apply a filter, usually 2x2 without overlapping, and store the largest or average of the 4 values multiplied by the filter, depending on whether it is a max-pooling layer or an average-pooling layer, respectively [12].

Fully Connected Layers

Fully Connected Layers are used to calculate class scores or probability at the end of a Convolution Neural Network. These layers take the results of the convolution and pooling process and convert it into a single probability vector, where each value is relative to a class [12].

2.3 Image Segmentation

Since one of the objectives of this dissertation is to create a model capable of doing the segmentation of road and objects in real time, in this subsection we will present this concept and some architectures and datasets to do it.

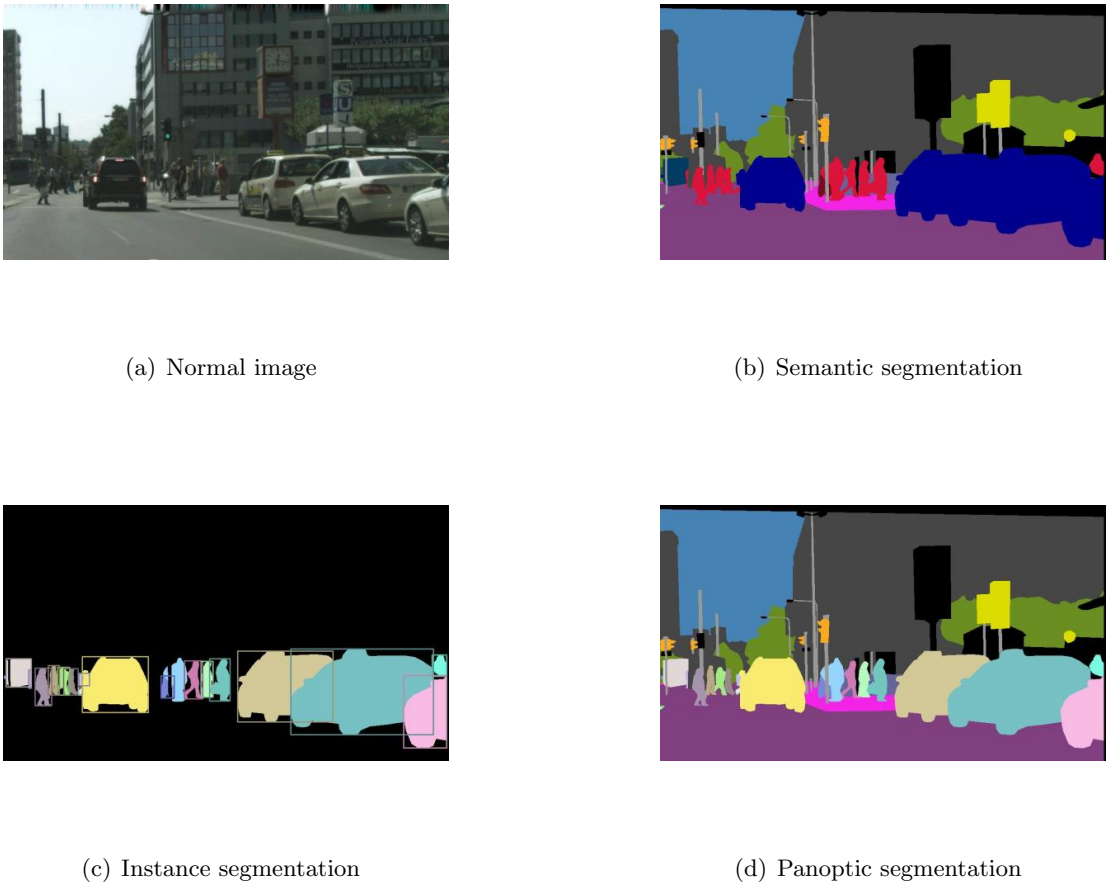
2.3.1 Semantic and Instance Segmentation

Image segmentation consists of identifying and classifying every pixel in the image and associating it to a given class. However, there are two distinct types of segmentation, semantic segmentation and instance segmentation. The first is to associate all pixels in the image to a category, representing all objects in the same category by the same color, while in instance segmentation the model separates objects within the same category, and there may be pixels that are not associated with any class.

When analyzing figure 2.11 it is easier to understand the differences between these two types of segmentation, realizing that the second method tends to identify things,

objects that it can detect and delimit, while the semantic segmentation studies the stuff, being this amorphous regions or uncountable regions such as sky, grass and road.

In 2019, Kirilov et al. [20] proposed a new task named panoptic segmentation that unites the two types of segmentation mentioned above and also presented a simple algorithm based on the combination of an instance model and a semantic segmentation model as the first approach for this type of task.



(a) Normal image

(b) Semantic segmentation

(c) Instance segmentation

(d) Panoptic segmentation

Figure 2.11: Comparison between different types of segmentation in an image [20].

2.3.2 Architectures for image segmentation

Fully Convolutional Network

The Fully Convolutional Network proposed by Long [21] was one of the first works for semantic segmentation using deep learning. This architecture is represented in figure 2.12 and is composed only of convolutional layers and is able to receive an image with an arbitrary size and return the correspondingly-sized segmentation map. The authors' architecture combines deeper layer weekly information with more superficial layer appearance information, thus achieving state-of-the-art segmentation performance when

the model was tested on PASCAL VOC, NYUDv2, and SIFT Flow.

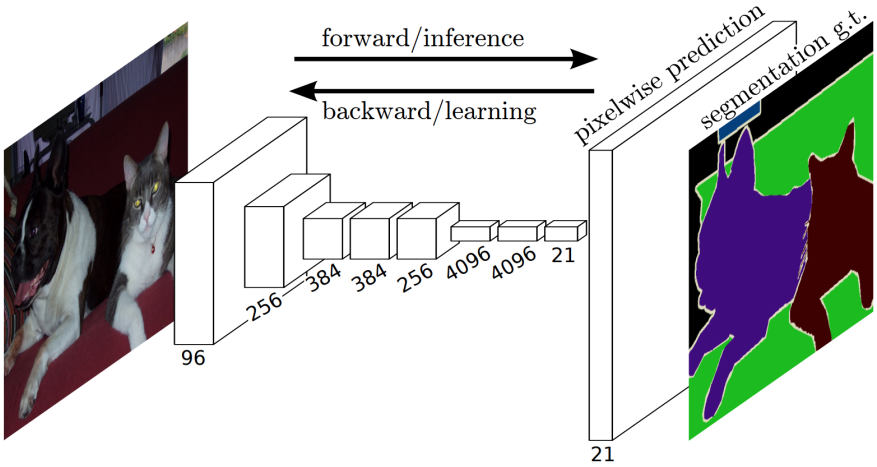


Figure 2.12: Representation of the fully convolutional network architecture [21].

SegNet

Segnet's architecture [22] consists of an encoder-decoder network followed by a pixel classification layer, as shown in figure 2.13. The encoder network is similar to the 13 convolution layers in VGG16 network and is responsible for the extraction of the features. On other hand, the decoder part convert the feature maps from the encoder into feature maps with the same size as the input. In this task, the network uses the pooling indices computed in the max-pooling layers of the correspondent encoder so it doesn't need to learn to upsample. This network can be trained from end to end, since it has few parameters.

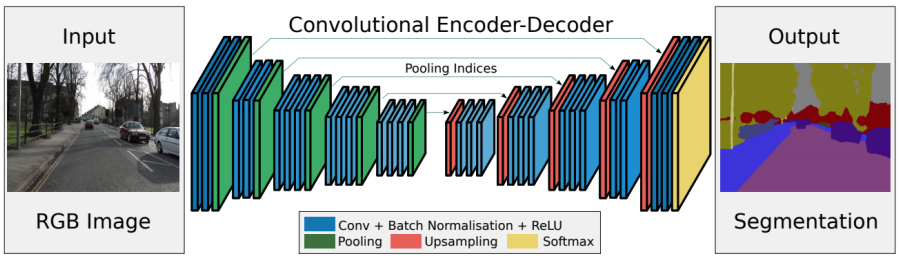


Figure 2.13: Illustration of the encoder-decoder architecture of SegNet [22].

ENet

Efficient Neural Network, as mentioned in the article [23], is based on ResNet architecture [24] and is composed of a main branch and several branches that separate from it, which later merge again through the addition of elements. These branches are called bottlenecks and consist of three convolutional layers, where in the first one a decrease in size is made, in the second one a main convolution is applied and in the third one an

increase in size is made. Between all the convolutions Batch Normalization and PReLU are positioned and for the regularizer is used Spatial Dropout.

When the bottleneck is downsampling, a max pooling layer is added to the main branch. The first 1x1 projection is replaced with a non-overlapping 2 x 2 convolution and the activations are zero padded to match the number of maps of features. In the decoder, the max pooling is changed to a max unpooling layer and a spatial convolution without bias is performed instead of padding. The initial block of this network and the bootleneck are presented in figure 2.14. This network achieved better frame rates when compared with SegNet under the same conditions and has a similar or better performance in terms of accuracy.

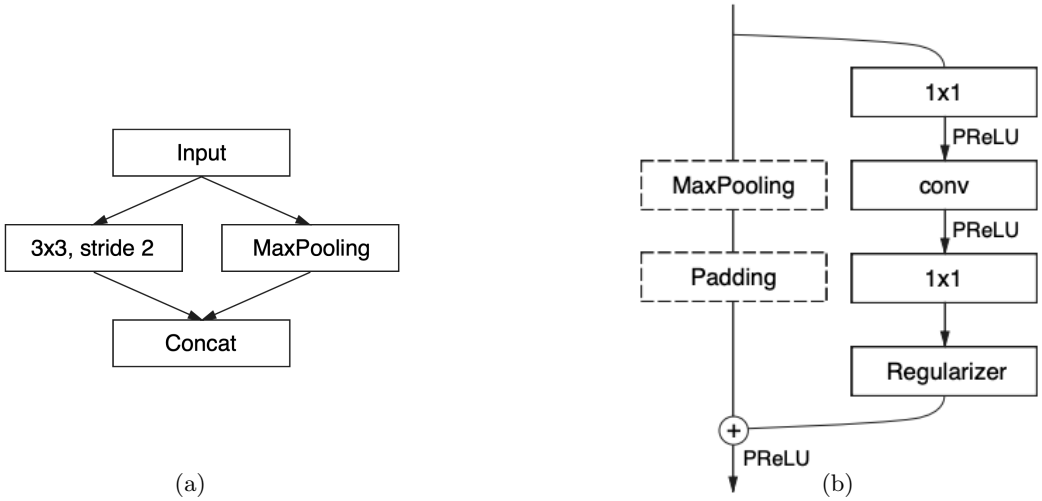


Figure 2.14: Representation of ENet initial block at the left image and its bottleneck module at the right [23].

BiSeNet

Bilateral Segmentation Network was proposed by Yu [25] and has two main parts, a Spatial Path (SP) and a Context Path (CP). The first one consists of three convolution layers followed by batch normalization and ReLU and is responsible for preserving spatial information and generate high resolution features. The latter uses a lightweight model in order to obtain a large receptive field and global average pooling to provide global context information to the receptive field.

As these two parts complement each other, since the first one detects low-level features and the second one high-level features, the authors used a Feature Fusion Module in order to combine the features that both parts detect. An overview of this network is presented in figure 2.15 and obtained a result of 68.4% mIoU in the CityScapes test data set at 105 FPS surpassing SegNet and ENet at both points.

ContextNet

Similarly to BiSeNet, ContextNet [26] combines a deep network to obtain a global context of the image with a shallow network that captures high level features. The former

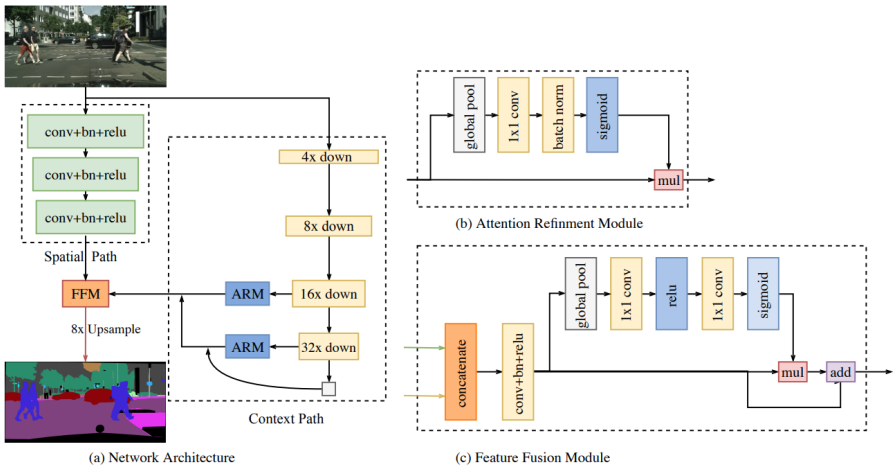


Figure 2.15: Bilateral Segmentation Network Architecture representation [25].

receive a low resolution input and its structure combines two layers of convolution and 12 bottleneck residual blocks. The latter uses the full resolution image and consists in four layers, where the first one uses a standard convolution and the others use depth-wise separable convolutions. The structure of this architecture is illustrated in figure 2.16.

In order not to increase the runtime, the authors decided to use feature addition to merge the features from both branches. This network achieved better accuracy in class and category mIoU than SegNet and ENet with a framerate similar to the latter.

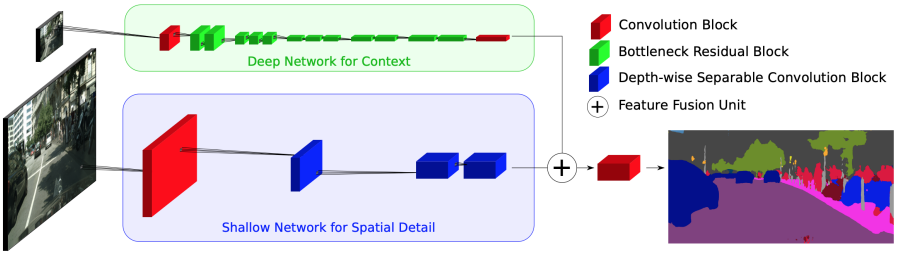


Figure 2.16: Illustration of the ContextNet architecture [26].

Fast S-CNN

Fast Semantic Segmentation Network combines ideas from two-branch and encoder-decoder networks [27]. This network consists of a learning to downsample module, followed by a global feature extractor and a feature fusion modules and ending with a classifier, as presented in figure 2.17.

The former first module has three layers, the first being a standard convolution layer and the the remaining depth-wise separable convolutional layers. The global feature extractor consists of inverted residual bottleneck blocks followed by a pyramid pooling module and captures the global context of the image. Feature Fusion Module combines high level with low level features just by adding them in a similar way to what is done in ContextNet and the final module has two depth-wise separable convolutions and an

standard convolution.

Fast S-CNN achieved 68.0% in class mIoU, which is better than SegNet, ENet and ContexNet and close to 71.4% of BiSeNet. Relatively to runtime this network outperformed all the others.

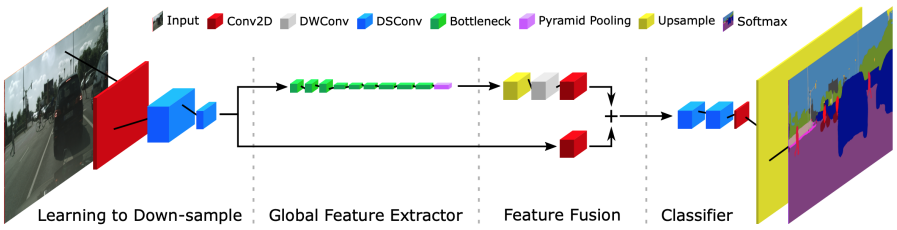


Figure 2.17: Overview of Fast S-CNN architecture [27].

LEDNet

LedNet is a lightweight network with an asymmetric encoder-decoder architecture with less than 1 million parameters [28]. The encoder network is based in ResNet network adding to this one a channel split and shuffle in each residual block to decrease the computation cost. In order not to increase the complexity of the network, the authors used an attention pyramid network(APN) as decoder. An illustration of this architecture is presented in figure 2.18.

The authors achieved a score of 70.6% mIoU in terms of class and 87.1% in category mIoU and were able to run the model at 71 FPS. This network is 30x smaller than SegNet.

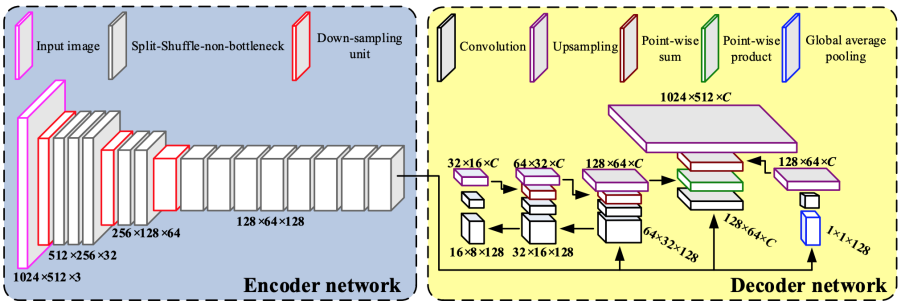


Figure 2.18: Representation of the asymmetric encoder-decoder architecture of LEDNet [28].

ESNet

The Efficient Symmetric Network is a nearly symmetric encoder-decoder network with around 1.6M parameters [29]. This networks is presented in figure 2.19 and the encoder of this network contains 3 blocks, where each one consists of a down-sampling unit and several Factorised Convolution units or Parallel Factorised Convolution Units in the case

of the third block. The decoder is similar, consisting of two blocks, identical to the first two of the encoder, however the down-sampling is replaced by an up-sampling unit.

In terms of segmentation accuracy, this network showed better results than SegNet and ENet while approaching the frame rate achieved by the latter, even having about 5 times more parameters.

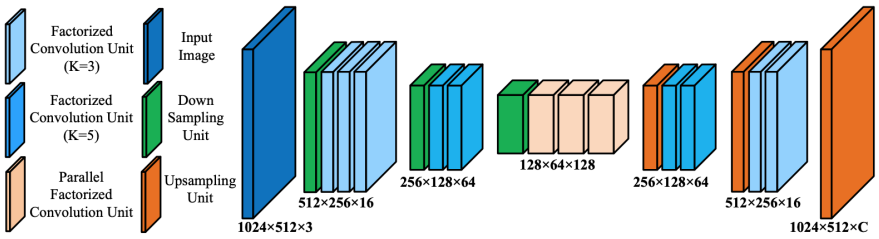


Figure 2.19: ESNet architecture overview [29].

Table 2.1: Architectures comparision. *validation set of PASCAL VOC 2011

Architecture	Parameters (millions)	Class mIoU (% on CityScapes Validation Set)
FCN	134.5	56.0*
ENet	0.4	58.3
SegNet	29.5	56.1
BiSeNet	49.0	74.7
ContextNet	0.9	66.1
Fast S-CNN	1.1	68.0
LedNet	0.9	70.6
ESNet	1.6	70.7

2.3.3 Datasets for Image Segmentation

Camvid

CamVid is a dataset for road segmentation that consists of 700 per-pixel labelled images with a resolution of 960x720 . There are 32 semantic classes but normally only eleven classes are used such as building, road or car [30] The main disadvantages of this dataset is that it is too small, the images are low resolution and there is not much variation of environments and weather conditions.

CityScapes

The CityScapes dataset is larger than the previous dataset, with 5000 annotated images with fine annotations and a resolution of 2048x1024. This data set has 19 classes and the images were captured in different cities of Germany during different seasons [31].

ApolloScape

The ApolloScape dataset features about 140K pixel-level semantic labelling images collected in 4 regions of China. This dataset also has about 90K images with instance-level annotations and its images have a resolution of 3384x2710 [32].

Berkeley DeepDrive

Berkeley DeepDrive, as known as BDD100K, is one of the largest datasets with 100k videos with a resolution of 720p. Similarly to the previous datasets, the images were recorded through different weather conditions and provides fine-grained pixel level annotations for 10k images, where 7K are for the training process, 1K for validation and 2K for testing [33].

Mapillary Vistas

This dataset consists of images from almost all continents with a minimum resolution of 1920×1080. It has 25K images in different conditions, where about 90% of the images are from urban areas and the rest from highways, rural areas or off-road [34].

Audi Autonomous Driving Dataset

Similar to some of the datasets mentioned above, this dataset contains images collected in various road scenarios and weather conditions. It was collected in southern German cities and contains more than 41K of segmented images with a resolution of 1920×1208 pixels, where about 31K come from the front camera of the car and the rest are similarly distributed to the other cameras present in the car [35].

2.4 Related Work Developed at LAR

ATLASCAR2 is part of a project that has been under development during the last years, and different types of classifiers have already been developed.

One of them [36] tried to take advantage of networks with Deep Learning and focused on the detection of pedestrians and cars on the public road. Taking into account the need to do the classification in real time, the author opted for a SqueezeNet-based architecture, and compared his model with other detectors such as the SSD and YOLO. The author made a separate car and pedestrian detection evaluation, and subsequently merged both, in which the model achieved an average accuracy of 43.2% using the KITTI evaluation method.

Another of the most recent works [37] has focused on detecting the limits of the road. Two cameras mounted on the car were used, a multi-camera and multi-algorithm architecture was developed, and the author created a ROS package to detect road lines,

Table 2.2: Workstation Specifications.

Dataset	Image resolution	No. images for segmentation	Different Scenarious
CamVid	960x720	700	No
CityScapes	2048x1024	5k	Yes
ApolloScape	3384x2710	140k	Yes
BDD100K	1280x720	100k	Yes
Mapillary Vistas	1920x1080	25k	Yes
A2D2	1920x1208	41k	Yes

where the detection of lines is done through the application of a colour threshold and a Sobel Edge Detector.

In addition, some deep learning models were explored for the detection of road lines and road segmentation, where very interesting results were obtained, especially concerning the depth of detection of road lines. A combination of classic techniques and modern approaches explored by the author is presented in figure 2.20.

As mentioned in 2.2.3, the influence of several training parameters on the performance of the final model was also studied by members of LAR. Among the experiments performed, the authors highlighted that changes in datasets, loss functions, optimizers or the application of other techniques such as data augmentation can have a great impact on the accuracy of a segmentation model.

2.5 Related Work Developed in Similar Projects

While for a driver rear and side-view mirrors are essential for safer driving, cameras are the most accurate way to create a visual representation of the environment around a car. Waymo [38], the company from Google's autonomous car project, uses in its cars a series of peripheral cameras and a 360 vision system that allows the vehicle to have the perception of what is around it by detecting pedestrians and traffic signs up to 500 meters away.

The purpose of this dissertation is to create a system that allows ATLASCAR2 to perceive everything around it. In order to be able to visualize the environment, it is intended to take advantage of a ring of cameras that is currently under development, which will consist of a set of cameras with a field of view of 60 degrees each. As mentioned in [39], the use of several cameras instead of single front cameras is much safer for autonomous driving since that, to drive safely, a car needs to be aware of everything around it. Brown et al. [40] presented a way to create panoramic images through individual images using Invariant Features. The authors initially extract SIFT features from all images and calculate which images have most features in common. After these tasks the RANSAC is used to obtain the homography, and a probabilistic

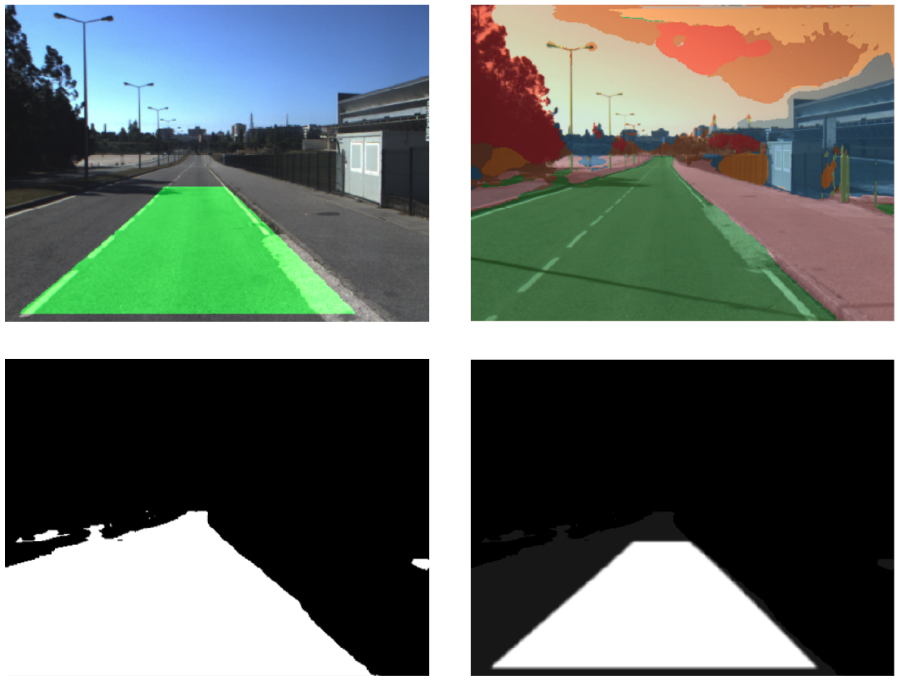


Figure 2.20: Combination of classic techniques and modern approaches to road detection [37]. The upper left image is relative to the application of a classic method while the one on the right is the result of applying a model that uses Deep Learning. The lower left image shows the combination of road detection using the two methods and the right image shows the confidence map resulting from the combination of the two methods.

model is used to find consistent image matches. A process similar to this was also used in [41] and in [42] different ways of performing image stitching are discussed. In chapter 4 the system developed for the creation of our panoramic images is presented.

In article [43], a system was created in order to detect the obstacles around the car using stereo vision. Two Ricoh Theta cameras were used to create a spherical panoramic image and the authors proposed an obstacle detection algorithm using depth estimation. However, in this work, the type of obstacles was not differentiated and the authors pointed out as possible improvements the inclusion of the detection of road marks, traffic signs and road detection.

In [39], the authors presented the sensor setup that allowed them to acquire data about driving, the environment around the car and the driving route to the destination. Through the developed setup, a dataset with 60 hours of driving data was created on Swiss roads. The Drive360 dataset contains images of the 8 GoPro Hero 5 cameras positioned on the roof of the car rotated 45^{o}, map data for navigation and driving manoeuvres captured via the CAN bus. The authors also created their driving model and made the comparison between single and surround view, with the second method performing better overall.

It was also studied in [44] the effect of the use of panoramic images in the process of segmentation on board autonomous cars. For that, a dataset of panoramic images was created with images generated by computer graphics previously combined as can be seen in figure 2.21 .The authors found that there was a need to train the model with

panoramic images to achieve better results, the images with a 180 degree field of view having the greatest benefit.

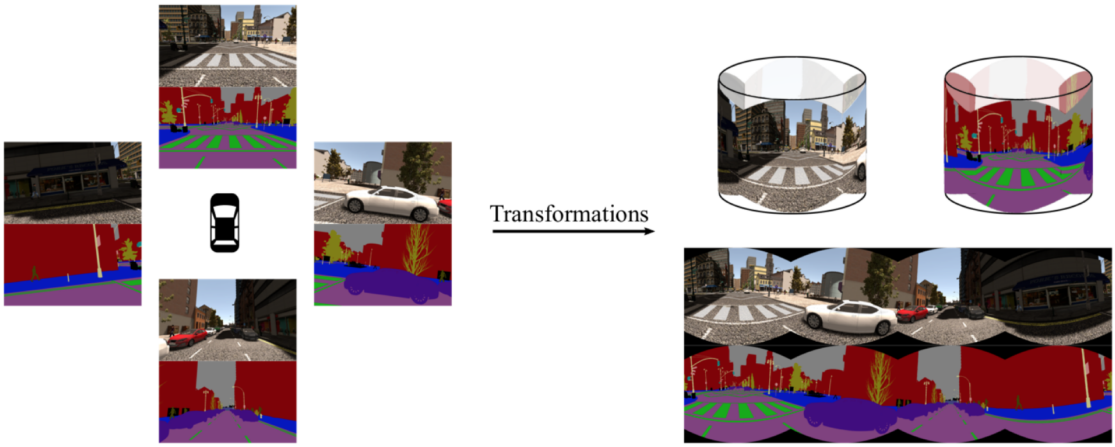


Figure 2.21: Illustration of the creation of a synthetic dataset by stitching individual images.

A Panoramic Annular Semantic Segmentation framework were presented in [45]. The authors used Mapillary Vistas dataset making small changes in the images so that the model could generalize to the panoramic images. For validation, the authors used the Mapillary Vistas Validation set and a panoramic testing set, which was created by collecting the images with a Panoramic Annular Lens system with about 400 finely annotated images for 4 classes. For validation, the authors used the Mapillary Vistas Validation set and a panoramic testing set, which was created by collecting the images with a Panoramic Annular Lens system[11] with about 400 finely annotated images for 4 classes. In the deployment phase, the panoramic image obtained is unfolded and divided into several segments that are given as input to the model. The resulting features maps are put together later in order to obtain a final panorama segmentation map. The authors studied the influence of the image division on these segments and realised that smaller classes needed more segments. Some examples of segmentation of the panoramic images are presented in figure 2.22.

Figure 2.22: Results of the segmentation of panoramic images presented in [45]. The first picture is a raw panoramic annular image, the middle one is the unfolded panoramic image and the last one is a segmentation map.

Authors in [46] evaluated the influence of data augmentation techniques related to skew and gamma correction, in order to create a more robust model to light variations and to the perspectives of the collected images. The authors used images from three NVIDIA GMSL 100º cameras and generated the panoramic images by stitching the individual images. ENet architecture was adopted and the models were first trained on the Cityscapes dataset and then fine-tuned in a dataset created by the authors. The Gamma correction was most noticeable when the validation dataset was the one created by the authors as it has more shadows with an improvement of up to 3.5% on mIoU, while the skew corrections were more relevant to the images collected by the side cameras. An example of the type of results obtained by the authors can be seen in figure 2.23

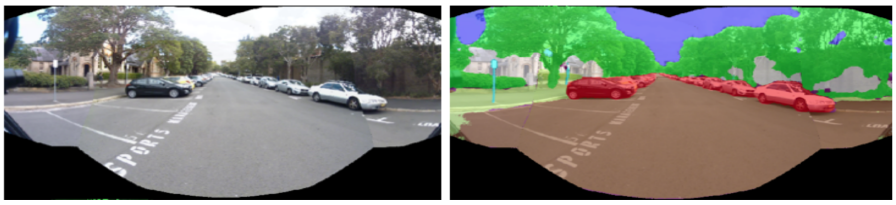


Figure 2.23: Panoramic image segmentation performed in [46].

In [47], the authors created a dataset of spherical panoramic images, collected by a ladybug5, to detect cars, people, lines or traffic signs present on the public road. With the creation of this dataset, the authors tried to evaluate how some of the main networks used in the detection of objects performed when trained with the Pano-RSOD, and the algorithm that obtained the best results was the YOLOv3. After this evaluation, the authors compared how the training with the created dataset had an influence on the results obtained and compared the performance of nets trained in other datasets, and none of them was close to the results obtained with the RSOD-textit.

2.6 Summary

Throughout this chapter the main areas of development in this dissertation were presented. The state of the art in image detection was described, introducing some general concepts about deep learning and exploring the field of image segmentation, referring to some architectures and datasets that allow us to create models for it. Finally, some works related to the Atlas project were mentioned and other projects that use panoramic images for road segmentation and objects on board autonomous vehicles were also presented.

Intentionally blank page.

Chapter 3

Experimental infrastructure

This chapter describes in detail the hardware and software used in this dissertation. Regarding the hardware, the cameras used, the server where the models were trained and also one of the GPU options to make the model inference are presented. In the second section, the software used to create the panoramic images and to train and process the models is introduced.

3.1 Hardware

3.1.1 Logitech C270 HD Webcam

The cameras chosen for this dissertation were the Logitech C270 and can capture video at a resolution of 1280x720 at 30fps with a 60º Field of View. These cameras have been selected for their quality/price ratio and their small size, allowing their use in the camera ring being developed also within the atlas project. A picture of this camera is shown in figure 3.1.

Figure 3.1: Logitech C270 HD Webcam.

3.1.2 LAR Workstation

For the training process of the models was used the new workstation of LAR (Laboratório de Automação e Robótica), which was developed for deep learning. The access to this workstation is done remotely by SSH and some characteristics of this computer as well as a picture of it are presented in table 3.1 and figure 3.2, respectively.

Table 3.1: Workstation Specifications.

Processor	AMD Threadripper 2850 Extreme
Graphic Board	4 x NVIDIA RTX2080TI
Memory	28GB DDR4 RAM
Storage	512GB SSD + 4TB HDD + 4TB SSD

Figure 3.2: LAR Workstation.

3.1.3 Jetson AGX Xavier Developer Kit

One of the objectives of this dissertation is the selection of a hardware that allows us to make the inference of our models. Jetson AGX Xavier Developer Kit was chosen for this task, since this hardware was recently used in a data matrix identification project [48] using Deep Learning networks by one of the LAR researchers and is currently available. The NVIDIA Jetson AGX Xavier, presented in figure 3.3, enables the performance of a GPU workstation in an embedded module under 30W and with a very small size, being ideal for robots or other autonomous projects.

3.2 Software

3.2.1 ROS - Robot Operating System

Robot Operating System is an open source framework that enables the development of collaborative software for robots [49]. In the ROS environment there are different modules, called nodes, that communicate with each other through messages and for communication it is necessary that the messages are published in topics. In addition, two nodes can communicate synchronously via ROS services, where there is a request message from one node to which a reply message is sent by the other.

ROS presents the possibility of recording topics in rosbags during data acquisition,

Figure 3.3: NVIDIA Jetson AGX Xavier.

and the user can define which topics are recorded. With this tool it is possible to re-run the topics acquired during the recording as if data was being collected in real time.

3.2.2 JupyterLab

JupyterLab is an interactive interface that allows working with different types of documents such as Jupyter notebooks or a simple text file [50]. JupyterLab works in a remote machine, in this case in the LAR Workstation, and is accessed through a web browser, having been used to create and train the selected models. In figure 3.4 its interface is shown.

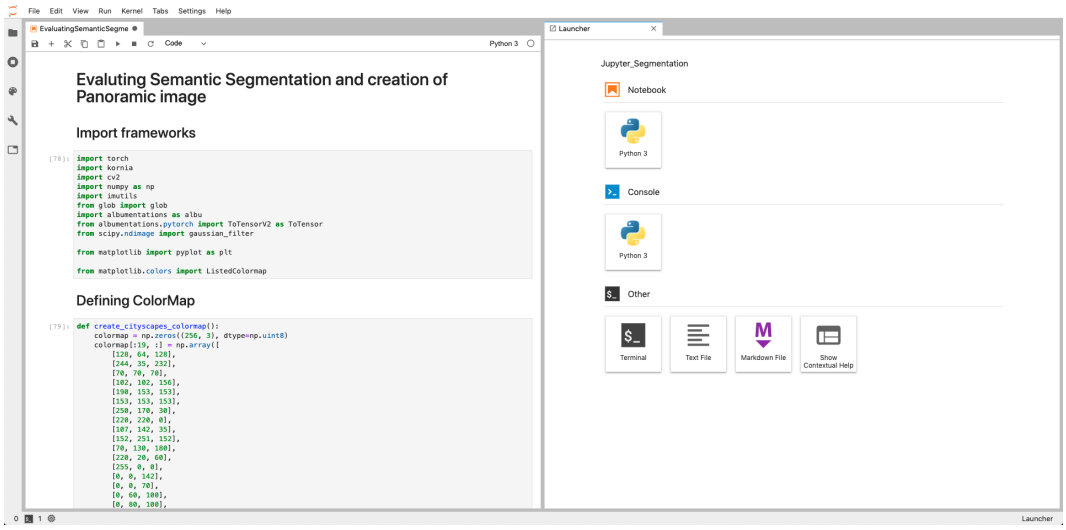


Figure 3.4: JupyterLab user interface.

3.2.3 Pytorch

PyTorch is an open source library based on Python that facilitates the creation of deep learning models. PyTorch takes advantage from Tensors, a multidimensional array similar to Numpy arrays, thus allowing them and their related operations to take advantage of GPU processing speed [51].

Other major advantages of PyTorch is the automatic differentiation for building or training neural networks and no need to pre-define the computational graph of the model.

3.2.4 OpenCV

OpenCV is an open source computer vision library that has different modules for image processing [52]. These modules contain different algorithms for image processing, either classic or machine learning algorithms, so that a programmer can implement them in real-time vision applications. Although OpenCV is written in C++, it has also Python, Java and MATLAB interfaces and is capable of running on different operating systems. In this work, this library was initially used for the creation of the panoramic image.

3.2.5 Kornia

Kornia is a computer vision library for PyTorch inspired by OpenCv. This library is composed of a series of packages with different types of operators for generic computer vision problems that can operate directly on tensors [53].

This library has been used since it offers a very easy integration with Pytorch, and its main use in the dissertation was in the creation of the panoramic image, that is explained in the following chapter.

3.2.6 Solidworks

SolidWorks is a Computer-Aided Design (CAD) software that allows the creation and modeling of 3D parts. This program was used to develop the camera supports, since they need to be well fixed to achieve a high-quality panoramic image.

Chapter 4

Proposed solution

This chapter discusses the proposed solution for the segmentation of road and obstacles on board ATLASCAR2. Initially, an overview of the proposed solution is presented, detailing the link between each element and the solution workflow. Next, the positioning of the cameras is explained and the supports that have been created are presented. The third section discusses the way the images are acquired and the last two sections explain the process of segmentation and the combination of the results obtained through the application of the networks.

4.1 Solution Overview

In order to have an easy integration in ATLASCAR2, the proposed solution is based on a ROS-based architecture, where initially the images are acquired through the cameras and are published in ROS topics. The previously published raw images topics are then subscribed and the panoramic image creation algorithm or the segmentation network is applied, depending on how the segmented panoramic image is to be created.

As represented in figure 4.1, the two branches of the proposed solution are quite similar. If the panoramic image has to be segmented, the algorithm to create the panoramic image runs first, and then the panoramic image is segmented by the segmentation network. Figure 4.2 shows the computation graph for this way of generating the segmented panoramic image. As shown in this graph, the nodes corresponding to the three cameras publish the raw images on three different topics. These topics are then subscribed to by the panoramic image creation node which applies the transformations to the images and publishes the panoramic image in the panoramic image node. Subsequently, the node for the segmentation network subscribes this image and publishes the result of the segmentation in a new topic.

In the second case, the segmentation network is applied to each image individually and then the segmented images are provided to the panoramic image creation algorithm and for this case, the computation graph is represented in figure 4.3. Similarly to the first case, the nodes corresponding to the three cameras also publish the raw images on the three different topics. However, in this second method these images are subscribed by the segmentation network node as can be seen in the computational graph. After the segmentation of the images from each camera, this node publishes the segmented images in new topics, which are then subscribed by the node that creates the panoramic image.

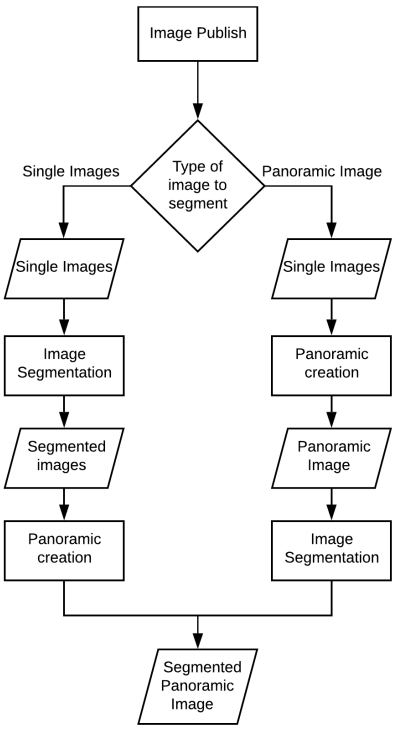


Figure 4.1: Flowchart of the proposed solution. In the left branch the raw images of each camera are segmented and published in ROS topics. Later, these topics are subscribed by the panoramic creation algorithm, which applies the transformations to the segmented images, originating the segmented panoramic image. In the other branch the process is done in reverse, where initially the panoramic image is created through the camera images and then the panoramic image is given as input to the segmentation network.

This node also subscribes the images from the cameras to calculate the transformation matrices and only then creates the panoramic image with the segmented images.

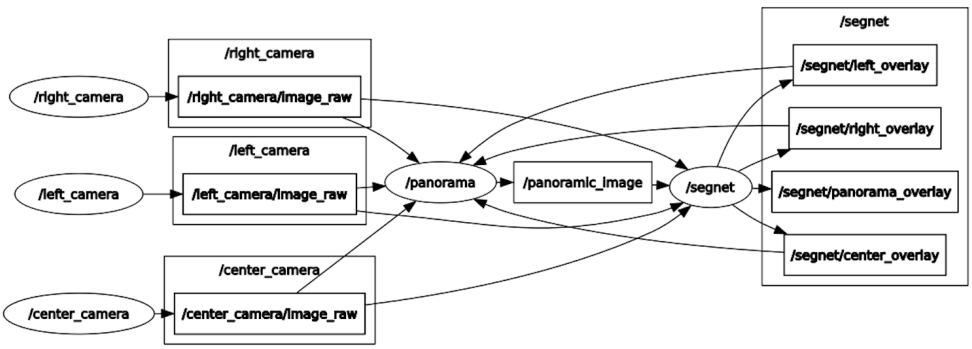


Figure 4.2: Computation graph from panoramic image segmentation.

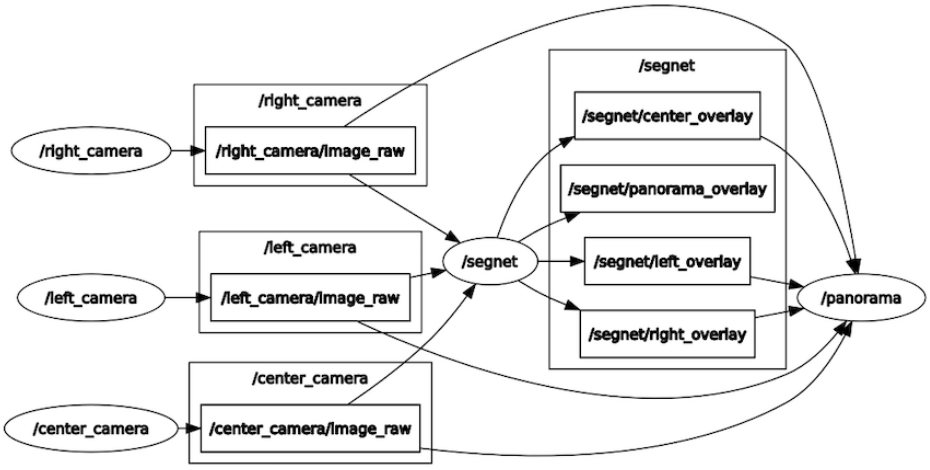
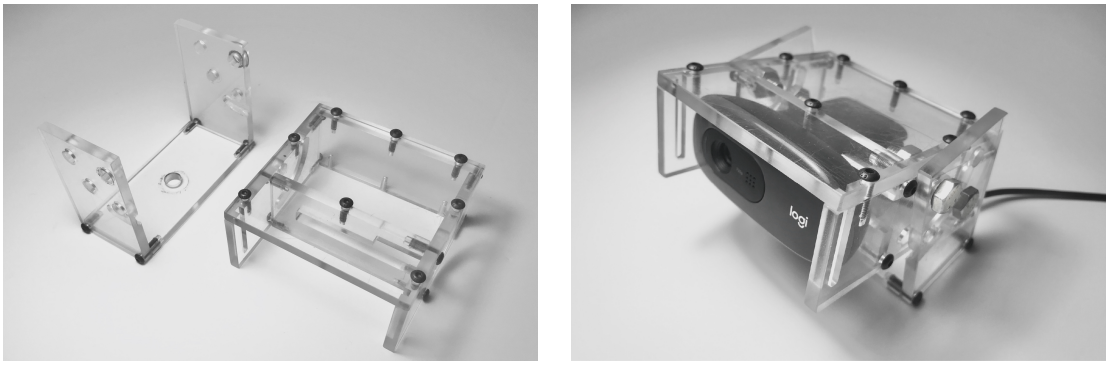


Figure 4.3: Computation graph of the creation of the panoramic image through segmented individual images.

4.2 Camera stands and setup

In order to create the panoramic image and to segment it, it is necessary to have the cameras fixed and properly positioned. Initially, the positioning of the cameras would be done through a camera ring that is being developed for ATLASCAR2, however, and as it was not ready to be used for this dissertation, it was necessary to create supports for the selected cameras.

During the development of the supports presented in figure 4.4 there was a focus on the importance of creating a way to easily adjust the position and orientation, while at the same time a robust design was necessary.



(a) Developed parts (b) Assembly with the camera

Figure 4.4: Camera mounts on the left and assembly with the camera on the right.

For this, and taking into account that the cameras have to be positioned on the structures positioned on the roof of the car, two parts were developed which, together with the connection piece suitable for these profiles, ensures the possibility of adjustment with three degrees of freedom. In addition, the necessity of making a stand that is simple to assemble and that allows an adjustment of the position and direction of the cameras easily has been also considered. In the first image of figure 4.4 are represented the two components that have been developed and in the second the assembly of this set with the camera.

In addition to these stands, it was necessary to check the positioning and number of cameras needed to create the panoramic image on board the car, and having initially selected Logitech C270 cameras, the field of view of these cameras had to be taken into account. To be able to create a panoramic image, as explained later in this chapter, the cameras had to overlap with each other. Therefore, to create a 360 degrees panoramic image more than 6 cameras are needed to have this overlap, and a study of their positioning has been done as shown in figure 4.5. As can be seen, with the variation of the positioning of cameras there is a change in the field of view. For a more central positioning, there is a decrease in dead zones and an increase in the overlap of the cameras while for a positioning of the cameras more at the edges of the vehicle the opposite happens. Besides this study, during the testing and data gathering only three cameras were used to create the panoramic images.

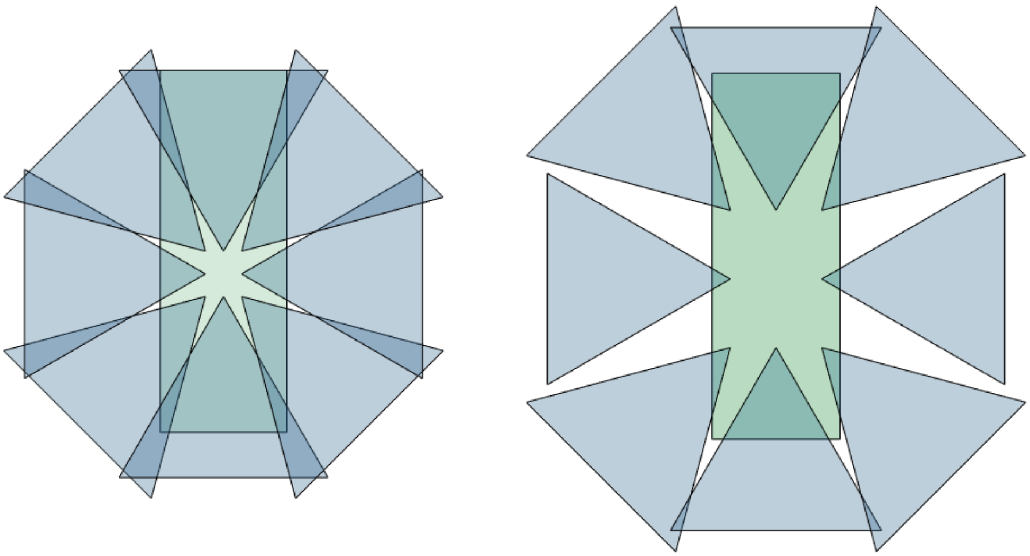
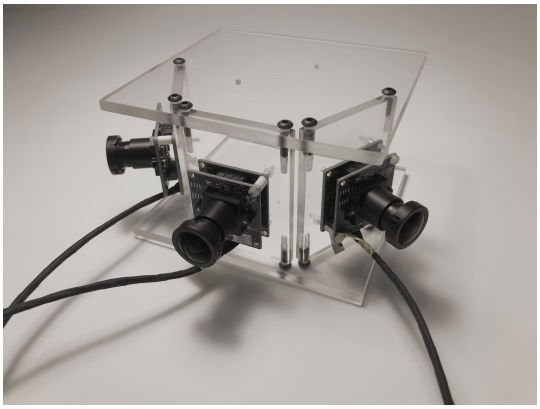
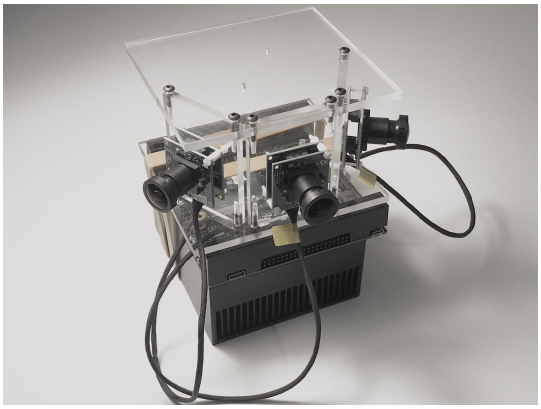


Figure 4.5: Study on the positioning of cameras on the ATLASCAR2 roof. The green rectangle represents the ATLASCAR2 and the triangles represent the field of view of each camera, using the 60 degrees of Logitech C270. The left image represents a more central positioning of the cameras, while the right image represents the positioning of the cameras at the edges of the support positioned on the ATLASCAR2 roof.

A second setup was used to perform the tests with a Jetson Xavier unit on board the ATLASCAR2. This setup has three 4K cameras with a field of view of 132 degrees each, and the positioning of the cameras between them was designed for another project [37] and is represented in the first image of figure 4.6.



(a) Camera positioning support



(b) Jetson AGX Xavier with cameras mounted

Figure 4.6: Cameras support on the left and assembly with Jetson AGX Xavier on the right.

In order to properly position the cameras and the Jetson Xavier system in the ATLASCAR2, both were grouped as shown in the right image of figure 4.6 and the setup was assembled inside the car as shown in figure 4.7. The choice of the assembly inside the vehicle was due to the restriction on the length of the connecting cables of the cameras that only allow a distance of up to 20 centimetres between the Jetson and the cameras.

Figure 4.7: Jetson AGX Xavier and cameras setup in ATLASCAR2. A suction stand was used to hold the Jetson AGX Xavier setup with the cameras. This was placed on the dashboard of the car and a small sponge was used on the top to prevent movement during the images acquisition.

4.3 Image Acquisition

In the first step of the proposed solution, the acquisition of individual images and their publication on ROS topics is done. Initially, it was intended to create the panoramic image after this acquisition, but as mentioned in the first section of this chapter, the creation of this can be done through the already segmented images.

This option was presented since in the different tests performed it was only possible to obtain a panoramic image at a frame rate of 6 FPS. Despite the alternatives presented, in all of them the images of each camera are necessary to create the segmented panoramic image.

To start publishing the images in the topics it is necessary to run the launch file presented in isting 4.1 and to subscription of the image topics is according to the listing 4.2.

```
<launch>

    <node pkg="cv_camera" type="cv_camera_node" name="FL_camera">
        <param name="device_id" value="4"/>
        <param name="frame_id" value="FL_camera"/>
        <param name="set_camera_fps" value="30"/>
        <param name="camera_info_url" value="file://$(env HOME)/.ros/
    camera_info/FL_camera.yaml" />
    </node>

    <node pkg="cv_camera" type="cv_camera_node" name="FM_camera">
        <param name="device_id" value="2"/>
        <param name="frame_id" value="FM_camera"/>
        <param name="set_camera_fps" value="30"/>
        <param name="camera_info_url" value="file://$(env HOME)/.ros/
    camera_info/FM_camera.yaml" />
    </node>

    <node pkg="cv_camera" type="cv_camera_node" name="FR_camera">
        <param name="device_id" value="0"/>
        <param name="frame_id" value="FR_camera"/>
        <param name="set_camera_fps" value="30"/>
        <param name="camera_info_url" value="file://$(env HOME)/.ros/
    camera_info/FR_camera.yaml" />
    </node>
</launch>
```

Listing 4.1: Launch file for the three cameras

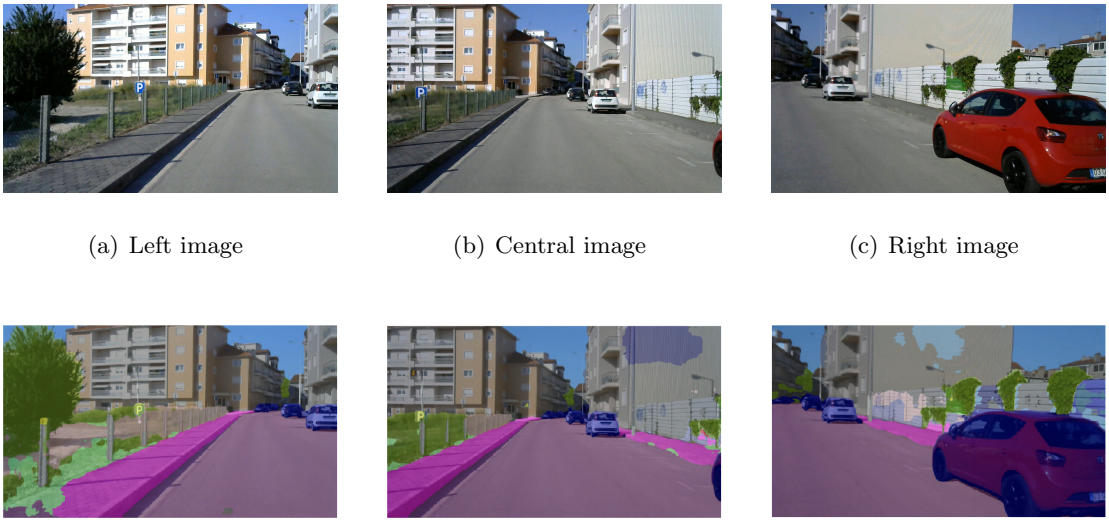
```
image_sub_FL = rospy.Subscriber("/FL_camera/image_raw",Image,self.
    callback_FL_image)

def callback_FL_image(self,data):
        try:
            cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
            self.FL_image = cv_image
        except CvBridgeError as e:
            print(e)
```

Listing 4.2: Image subscription example

4.4 Image Segmentation

One of the key points of the proposed solution is the segmentation of images. In order to integrate image segmentation models to make inference in Jetson AGX Xavier, a library from NVIDIA was used. In addition to this library, a ROS package called Panoramic Segmentation was created based on a repository for ROS deep learning [54] to perform image segmentation.

In this package the images are subscribed and the segmentation node uses the model that is specified to segment them, publishing in different topics the channel mask, the colour mask and the overlay of the individual images and the panoramic if applicable. In figure 4.8 three images are shown before and after their segmentation, and in figure 4.9 are represented the normal overview image, its segmentation and its road mask.



(a) Left image (b) Central image (c) Right image

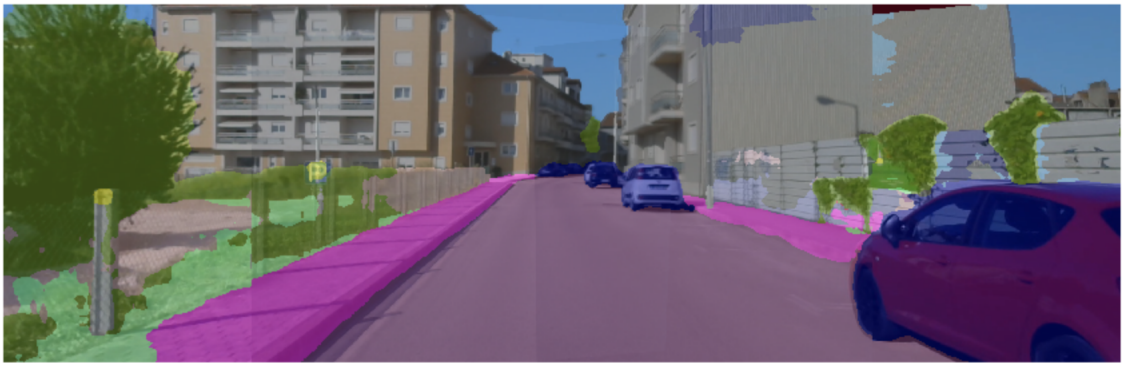
(d) Left image segmented (e) Center image segmented (f) Right image segmented

Figure 4.8: Individual images before and after segmentation.

Apart from the use of the segmentation models already existing in the Jetson Inference library [55], some models were trained in the LAR workstation. Before training the models, we defined the architectures to train and which datasets to use. Regarding the architectures, some of the selected were ENet, ContextNet and Fast S-CNN. Most of the selected architectures were chosen due to the low number of parameters which allows a faster inference process.

For the training process of these networks, two datasets were chosen, CityScapes and BDD100K. The first dataset is often used for image segmentation and most of the chosen architectures usually present their performance in its validation set. The choice of the second dataset was due to the fact that it has a larger amount of images that end up having the same resolution as the images acquired by our cameras. The process of training these architectures is discussed in more detail in the next chapter.

(a)



(b)

(c)

Figure 4.9: The first image shows the panoramic created with the images from the three cameras, the second image is the panoramic segmentation and the last one is the mask for the road class.

4.5 Data Combination

After the segmentation of the images it is necessary to combine them in order to obtain a panoramic image, and this process can be done in two ways, as mentioned previously. In the first, method can combine the segmented images and only then proceed to the extraction of the binary masks of the classes that are intended to be analyzed. Another option is to extract the binary masks from each image individually and then conjugate them in order to create the panorama. Although these two approaches are distinct, the creation of the panorama is done in the same way, and figure 4.10 shows the flowchart of this process.

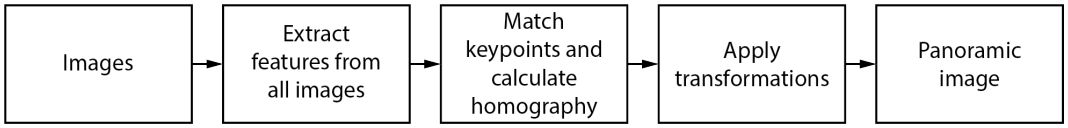


Figure 4.10: Flowchart of panoramic image creation. Despite the alternatives for creating the panoramic image, the stage of extracting the features of the images needs to have raw images as input, whether to create only the panoramic image or the road mask present in it.

To create the panoramic image it is necessary to know the transformations between each image. To do this, a first function was created to calculate the necessary transformations to apply, and this function (Listing 4.3) is only used each time the camera positions are changed. To calculate these transformations, the images to be calibrated are provided, applying a detector feature to each one, and then the homogeneous matrix between each pair of images is obtained.

```
def transformationsCalculator(self,images,ratio=0.8, reprojThresh=4.0):
    (image_left, image_center, image_right) = images

    (kpsLeft, featuresLeft) = self.detectAndDescribe(image_left)
    (kpsCenter, featuresCenter) = self.detectAndDescribe(image_center)
    (kpsRight, featuresRight) = self.detectAndDescribe(image_right)

    if kpsLeft is None or kpsCenter is None or kpsRight is None:
        print("It was not possible to extract the keypoints")
        return None

    M_left_center = self.matchKeypoints(kpsLeft, kpsCenter,featuresLeft,
    featuresCenter, ratio, reprojThresh)
    M_right_center = self.matchKeypoints(kpsRight, kpsCenter,
    featuresRight, featuresCenter, ratio, reprojThresh)

    if M_left_center is None or M_right_center is None:
        print("At least one of the matrices was not calculated!")
        return None

    return (M_left_center[1],M_right_center[1])
```

Listing 4.3: Function to calculate the transformation matrices

Two sub-functions are used are used in this function, the first one is responsible for extracting the features from each image by applying a feature detector (Listing 4.4) as shown in figure 4.11.

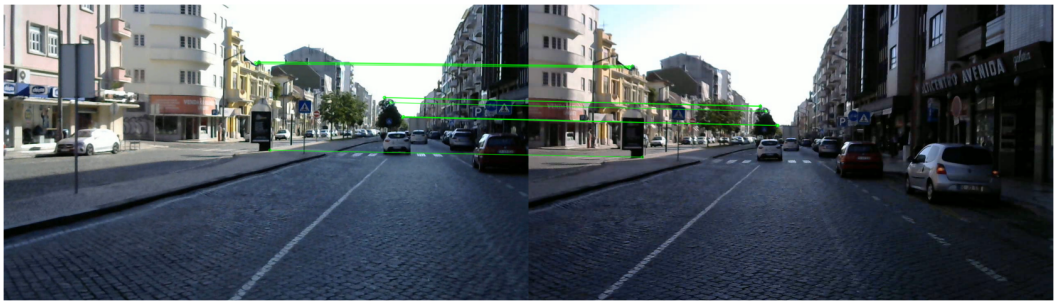


Figure 4.11: Feature matching between images.

The most commonly used feature detectors in this kind of tasks are the SIFT and SURF, however the ORB, Oriented FAST and Rotated Brief was tested, since to use the other two it is necessary to install the OpenCV Contrib package. This detector computes less key points than the two mentioned detectors, but it is an efficient alternative to the previous two.

```
def detectAndDescribe(self, image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    detector = cv2.ORB_create()
    kps = detector.detect(gray, None)
    (kps,features) = detector.compute(gray, kps)
    kps = np.float32([kp.pt for kp in kps])
    return (kps,features)
```

Listing 4.4: Feature detector function

Once all the key points in the image have been detected, the second function (Listing 4.5) is used to identify which points are common to each pair of images. A Brute Force Matcher [56] is used to combine an image with all the other features of another image and returns the match based on distance. Although this method is slow, there won't be much of a problem since this task is performed only in the calibration task. If the number of matches found is greater than 15, the homography is calculated and returned.

```
def matchKeypoints(self, kpsA, kpsB, featuresA, featuresB, ratio,
    reprojThresh):
    matcher = cv2.DescriptorMatcher_create("BruteForce")
    rawMatches = matcher.knnMatch(featuresA, featuresB, 2)
    matches = []
    for m in rawMatches:
        if len(m) == 2 and m[0].distance < m[1].distance * ratio:
            matches.append((m[0].trainIdx, m[0].queryIdx))
    if len(matches) > 15:
        ptsA = np.float32([kpsA[i] for (_, i) in matches])
        ptsB = np.float32([kpsB[i] for (i, _) in matches])
        (H, status) = cv2.findHomography(ptsA, ptsB, cv2.RANSAC,
    reprojThresh)
        return (matches, H, status)
    return None
```

Listing 4.5: Feauture matching function

After this calculation is done for all pairs of images, it is then possible to make the panoramic image and for this the transformations are applied to each mask or segmented

image. The left image or mask suffers the transformation relative to the homogeneous matrix between this image and the central one, while the right one suffers the transformation relative to the homogeneous matrix between the right image and the central one. In this part of the code (Listing 4.6), unlike the previous one, the Kornia library [53] was used to replace the OpenCV since this task will be constantly repeated, and so we can take full advantage of the GPU used in the inference of our model.

```
def stitch(self,Masks,result_width,M_left_center,M_center_right):
    (Maskleft_image, Maskcenter_image, Maskright_image) = Masks

    self.cachedHlc = M_left_center
    self.cachedHrc = M_center_right

    T = np.array([[1.0, 0.0, (result_width/2)-(Masks[0].shape[3]/2)],
                  [0.0, 1.0, 0.0],
                  [0.0, 0.0, 1.0]]).astype(dtype=np.float32)


    transformations = [self.cachedHlc, np.identity(3, dtype=np.float32),
                       self.cachedHrc]
    result = np.zeros((Masks[0].shape[0],Masks[0].shape[1],result_width))
              .astype(np.float32)
    weights = np.zeros_like(result)

    for i in range(len(Masks)):
        warp = kornia.warp_perspective(Masks[i],torch.tensor(np.dot(T,
               transformations[i]),dtype=torch.float32),(Masks[i].shape
               [2],result_width))
        weight = kornia.warp_perspective(torch.tensor(np.ones_like(
                 Masks[i])),torch.tensor(np.dot(T,transformations[i]),
                 dtype=torch.float32),(Masks[i].shape[2],result_width))

        result = kornia.color.add_weighted(torch.tensor(result),1.0,
                 warp,1.0,0.0)
        weights = kornia.color.add_weighted(torch.tensor(weights),1.0,
                  weight,1.0,0.0)

    return np.uint8(kornia.tensor_to_image(result) / kornia.
             tensor_to_image(weights))
```

Listing 4.6: Stitching function

As an alternative to the two panoramic image creation techniques presented, other techniques using polygons or points present in a segmented mask of a given class were explored. For this, two subfunctions were created, the first responsible for extracting the points of a class and the second for creating the panoramic image.

To extract the points relative to a given class, the function 4.7 takes as input the representative binary image of the class and calculates the coordinates of the points belonging to the mask. In this operation, it is possible to calculate the coordinates of all the points of the mask or only the coordinates of its outline(border).

Once this calculation has been made, the function 4.7 is responsible for applying the transformations to the previously calculated points and subsequently representing them as defined. If the desired representation is in the form of a polygon, this can be done either by the delimiting line of the polygon or by the polygon filled in as shown in the first and second images of figure 4.12, respectively.

```
def polygonGenerator(image,hullMode):

    if hullMode ==False:
        #Get coordinates where road was detected
        points = np.column_stack(np.where((image)==1))
    else:
        #Convert the image to grayscale & find edges
        img = np.uint8(np.float32(image))
        edges = cv2.Canny(img,3,3)
        coords = np.column_stack(np.where(edges==255))
        points = np.zeros((len(coords)+1, 2))
        points[:len(coords),:]= coords
        points[len(coords),0]= 736

    return points
```

Listing 4.7: Function to determine the coordinates of points in a particular class.

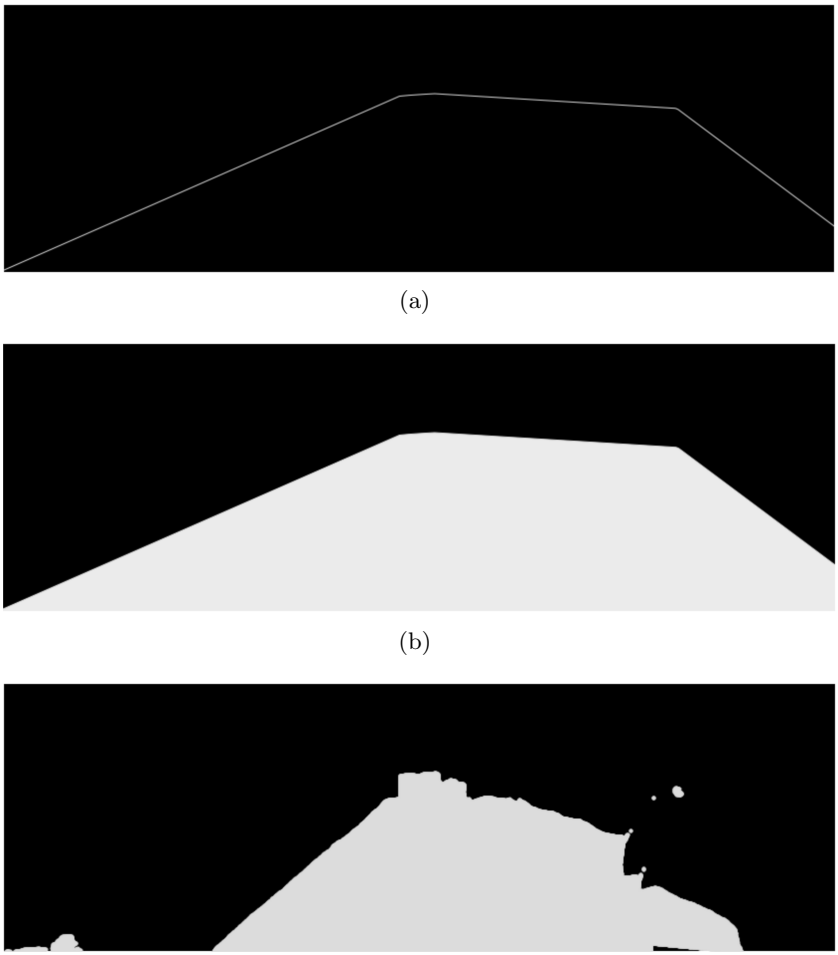
```
def polygonStitcher(Points,img_height,img_width,M_left_center,
    M_center_right,result_width,hullMode,Lines=False):
    (PointsE, PointsM, PointsD) = Points

    d = np.array([pointsD.astype('float32')])
    m = np.array([pointsM.astype('float32')])
    e = np.array([pointsE.astype('float32')])

    T = np.array([[1.0, 0.0, (result_width/2)-(img_width/2)],
                          [0.0, 1.0, 0.0],
                          [0.0, 0.0, 1.0]]).astype(dtype=np.float32)


    pointsOutD = cv2.perspectiveTransform(d, np.dot(T,M_center_right))
    pointsOutM = cv2.perspectiveTransform(m, T)
    pointsOutE = cv2.perspectiveTransform(e, np.dot(T,M_left_center))


    imagemPoints = np.zeros([img_height,result_width])

    pointList=np.concatenate((pointsOutE,pointsOutM,pointsOutD))

    return pointList
```

Listing 4.8: Part of the function that applies the transformations to the points of the polygons or of the masks.

As an option to these two shapes, the function can also represent the road mask as illustrated in the last image of figure 4.12. The latter generates a result similar to that shown in the third image of figure 4.9, however only the transformations are applied to the coordinates of the points representing the class, contrary to what is done in figure 4.9 where the transformations are applied to all points of the individual images.

4.6 Summary

Throughout this chapter, the proposed solution for the segmentation of road and objects on board ATLASCAR2 was introduced. Initially an overview of the solution was



(a)

(b)

(c)

Figure 4.12: The first two images concern the generation of the polygon that represents the road class, where the first one represents its contours and the second one its filling. The last image is the mask created with the coordinates of the road points extracted from the individual images.

presented, mentioning the two approaches used to create the segmented panoramic image. After that, the camera supports developed and the study of their positioning on the roof of the car were explained. The image segmentation process was also presented, specifying which libraries were used for the inference process in the Jetson Xavier unit. Finally the process of creating the panoramic image was detailed and three solutions in addition to those mentioned in the first section were proposed.

Intentionally blank page.

Chapter 5

Experiments and Results

This chapter describes all the tests performed throughout this dissertation, as well as the results obtained from them. Initially, some tests are detailed regarding panoramic images and data gathering. The process of training the models and the results obtained, both in terms of performance and accuracy, are presented next.

5.1 Panoramic tests and setup positioning

One of the first steps of this dissertation was to explore the field of panoramic images. As a first step, and after having a program to create the panoramic images similar to the one presented in the previous chapter, it was decided to perform some road tests to understand what difficulties might exist during navigation.

Initially, several tests[1] were performed on camera positioning, using what had already been explored in a more theoretical way in section 4.2. With these tests it was confirmed that for a setup with cameras closer and in a more central position of the car, the panoramic images generated were better, since there was less variation of brightness between the cameras and a greater overlap between the images that made the initial calibration easier. Nevertheless, and as can be seen on image 5.1, the overlap between the images during these tests was too big, which meant that the gain in terms of field of view was not significative enough.

Figure 5.1: Frame of the camera positioning test video. The overlap of the cameras is too large since the side images are slightly overlapped.

[1] Camera positioning test video in Aveiro `https://www.youtube.com/watch?v=69jH8Cza7gg`.

This is due to the fact that in the space where the initial calibration of the cameras was made, i.e., the identification of the key points and the respective calculation of the transformation matrices for the lateral images, the calibration program could not find enough common points in each pair of images and for this the overlap between the images had to be increased in order to obtain the panoramic image. One solution to this problem is to use an environment with a greater diversity of objects or by positioning some objects in the overlapping zones of the images in order to facilitate the identification of key-points.

Another adversity found during the calibration process was the difference in light intensity in each image, which sometimes led to the panoramic image not being obtained since, as shown in figure 5.2, the feature detector has difficulty in identifying key points in a pair of images.

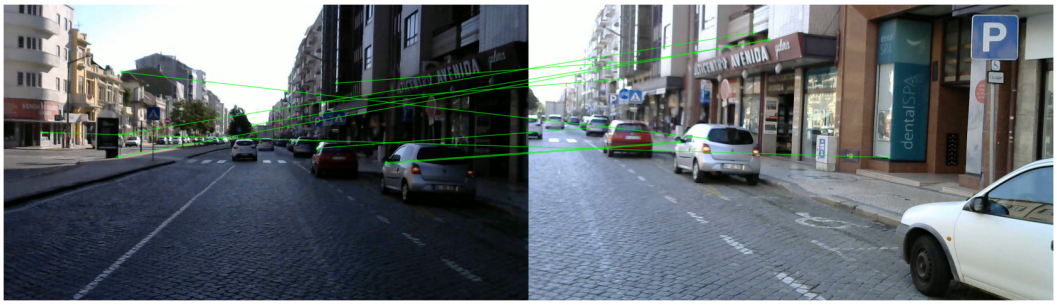


Figure 5.2: Poor detection of key points due to discrepancy in brightness intensity between the two images.

After defining a positioning for the cameras, some tests were carried out on board in order to check whether the frame rate with which the panoramic image was published was sufficient for navigation. During this stage, a laptop was used to perform the tests, and a frame rate of 6 FPS was obtained in the ROS topic regarding the panoramic image. With this, we tried to identify which operation took the longest time to be performed, having identified that the task that took the longest time to be performed was the application of the transformations to the images with *WarpPerspective*. One of the possible solutions for this problem is the creation of the panoramic image by combining the individual segmented images or through the representation of the polygons that best represent a specific class. These solutions will be evaluated and compared in section 1.5 of this chapter.

5.2 Data Gathering

After performing the tests mentioned in the previous section, a collection of images was made for the city of Aveiro. The main reason for this collection was the creation of a small validation set that served to verify which model had the best results in the streets of Aveiro, as demonstrated later in this chapter.

As before, the Logitech C270 cameras were positioned on a grid placed on the roof of a car, as shown in figure 5.3, and some streets of Aveiro were covered, being presented two of the routes traveled in figure 5.4, collecting more than 26 minutes of videos with a resolution of 1280x720 on each camera.

Figure 5.3: Positioning of cameras for data gathering.

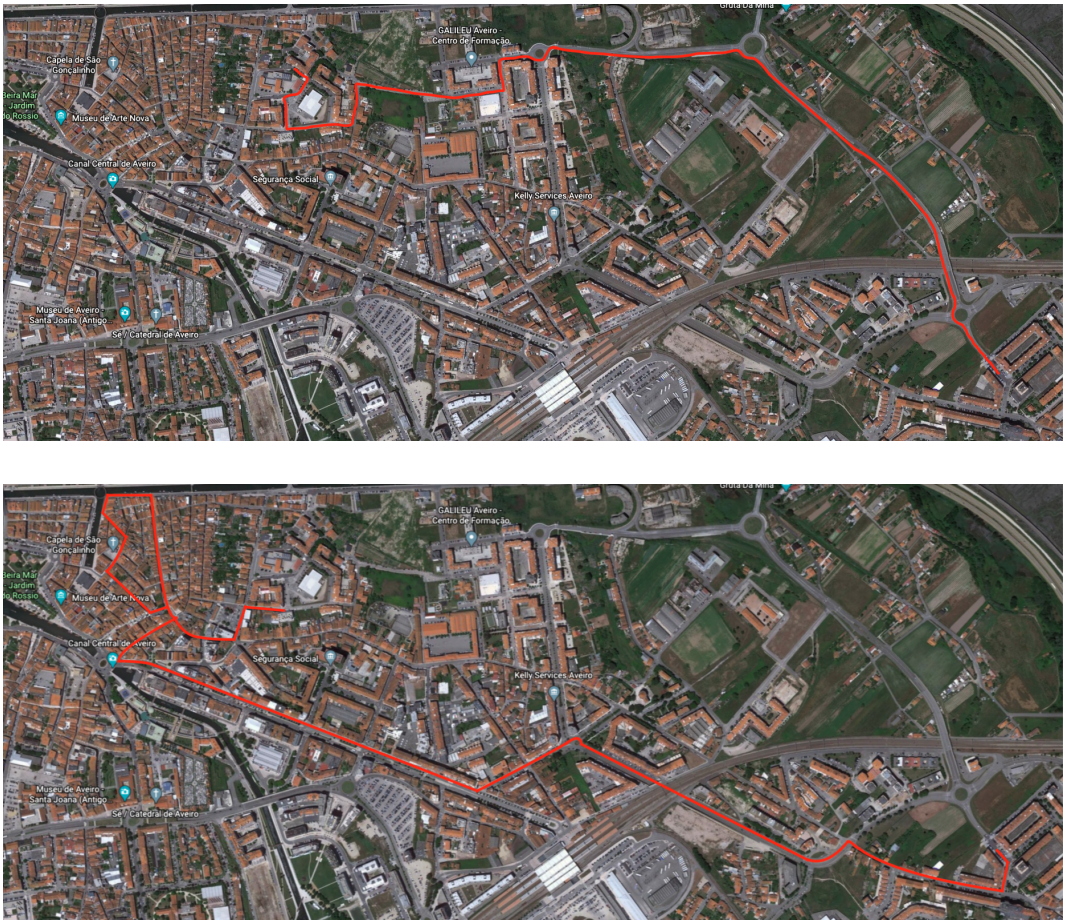


Figure 5.4: Routes taken during image acquisition for the creation of the validation dataset.

To create the validation set, it was necessary to choose the most representative images of Aveiro since the labelling process takes some time. Initially, an automatic selection of the images was idealized, extracting the embeddings of each image and making its

representation in order to understand which set of images is more distinct. Through this representation it would be possible to define the best set of images of Aveiro to make the labelling. However, taking into account the time this selection would take to be implemented, a manual selection was chosen, based on a first selection of images with 10 seconds between them, later suffering a refinement, where a total of 120 images were selected, 40 from each camera. In this selection an attempt to select the images that covered more areas of the city was made, while also selecting images of characteristic points of the city.

After selecting the images to create the validation set, LabelBox [57] was used as the platform to label the images. Before starting the labelling process, a small introduction was made to this platform in order to get to know the tools available for the labelling and to understand how long it would take to create the validation set. With this process of setting up LabelBox and after making an estimate of the time it would take to do the complete labelling of the images, it was defined that only the labelling of road, sidewalks and cars present in the images would be done. The choice of these specific classes is due to the fact that they are important for navigation, since the road is the main element to be detected in order to carry out a route planning, cars are usually the main obstacles in traffic and the sidewalk is due to the fact that it is one of the classes that can be mistakenly detected as road.

In figure 5.5 are represented some images with the respective labels of the selected classes, and this process took about 18 hours of manual work to be completed for the 120 images. Regarding the presence of the classes in the images, all images have the class relative to the road and about 92% and 88% contain cars or sidewalks, respectively.

5.3 Training Process

As mentioned earlier, it is necessary to have a model capable of segmenting the images in real time. For this purpose, some of the architectures mentioned in section 2.3.2 were selected, and this selection was based on the implementation feasibility of the architectures and the training of the models.

For the training process, models were trained with CityScapes dataset, and ContextNet was also trained with Berkeley Deep Drive dataset to prove the benefits to be expected from training with a larger dataset. The training process was similar in almost all models. The most used loss function for the training process was OHEM Loss, although cross entropy loss was also used for ENet training. Regarding optimization, AdamW was chosen for ContextNet and ENet training and SGD for Fast S CNN and SegNet. For the learning rate schedulers, the Cossine Anealing Scheduler was used in all architectures.

For the training, all models were trained with 19 classes, were applied random scale factors between 0.5 and 2.0 to the training images and random crop of 768 pixels in each sizes. Horizontal flip transformations or saturation changes have also been applied. In this way, the models are less susceptible to variations of image sizes, making the model more robust. Apart from using different datasets for ContextNet training, a model with twice the parameters was trained in the CityScapes dataset and the training was also explored with the knowledge distillation method. In this last method, a smaller model is trained to replicate a larger model, often making the analogy of this method to a teacher

Figure 5.5: Some examples of images selected for the validation set on the left and the respective labels created in LabelBox on the right.

teaching a student.

5.4 Performance and Accuracy Evaluation

In this section the trained models are evaluated both in terms of performance and accuracy. For the accuracy evaluation, the results are compared with the values obtained by the authors of the architectures in the respective dataset, the models are evaluated in the dataset created with images of the city of Aveiro and a qualitative analysis is made taking into account the results obtained in images of the city of Aveiro. Regarding the performance, all models are evaluated in one GPU of the computer of LAR and compared with each other the processing speed.

5.4.1 Accuracy Datasets

Table 5.1 shows the results of all trained models, as well as the mIoU values obtained in the architectures' papers for the CityScapes validation set for the 19 trained classes.

Table 5.1: Results obtained by the models trained in their validation sets. ContextNet 2x architecture contains two times the parameters of the standard.

Architecture	**Training Dataset**	**Validation Dataset**	**mIoU** [%]	**IoU Road** [%]	**IoU Sidewalk** [%]	**IoU Car** [%]
ContextNet	CityScapes	CityScapes	62.42	95.96	74.27	89.32
ContextNet Knowledge Destillation	CityScapes	CityScapes	65.80	96.81	76.63	91.11
ContextNet 2x	CityScapes	CityScapes	68.51	97.02	77.96	92.15
ContextNet 2x	BDD100K	BDD100K	51.80	93.32	57.87	87.64
Fast S-CNN	CityScapes	CityScapes	65.53	97.05	77.22	91.13
SegNet	CityScapes	CityScapes	63.15	94.99	79.62	92.95
ENet	CityScapes	CityScapes	56.40	95.06	69.73	92.95

The architectures in which the value was closer to what the authors reached were ContextNet and Fast S-CNN, and for the former, when trained with knowledge distillation, the result was only 0.3% below the value of the paper. Still for the first architecture, when trained with the double of parameters it obtained the best result in the CityScapes validation set, surpassing the previous one by 2.7%.

Regarding ENet, the results obtained with the training were slightly lower than the value of the authors, and it was with this architecture that the lowest value of mIoU was obtained. The largest architecture tested was SegNet, however the trained architecture had a small variation in its format. This variation occurred in the decoder part of the network and led to a reduction in the number of parameters used by the network. In spite of that, it ended up obtaining a result superior to the one presented by the authors, surpassing this one by about 7%.

Taking into account the results obtained after training all these models, it was decided to train the ContextNet architecture which has double the parameters of the regular architecture in the Berkeley Deep Drive dataset, since it was the model that had achieved the best result among the previous ones, having obtained a value of 51.80% of mIoU. Although in the ContextNet article there is no evaluation in the BDD100K, the value achieved in the validation defined in the article of this dataset was 56.9% for the semantic segmentation task, so it is noted that there is still room for improvement.

5.4.2 Accuracy Aveiro Dataset

To make a better comparison between the trained models, and to understand with which training dataset the best results were obtained, the result of the evaluation of the models in the Aveiro dataset is presented in table 5.2. The values of mIoU were obtained through validation in the 3 classes for which the labelling was done.

Table 5.2: Aveiro Dataset comparision result.

Architecture	**Training Dataset**	**Validation Dataset**	**mIoU** [%]	**IoU Road** [%]	**IoU Sidewalk** [%]	**IoU Car** [%]
ContextNet	CityScapes	Aveiro Dataset	77.91	89.57	55.13	89.20
ContextNet Knowledge Destillation	CityScapes	Aveiro Dataset	79.20	90.84	54.47	92.30
ContextNet 2x	CityScapes	Aveiro Dataset	79.98	91.20	56.88	91.87
ContextNet 2x	BDD100K	Aveiro Dataset	83.05	92.22	62.75	94.18
Fast S-CNN	CityScapes	Aveiro Dataset	77.52	89.89	54.67	88.01
SegNet	CityScapes	Aveiro Dataset	72.03	82.62	48.73	84.74
ENet	CityScapes	Aveiro Dataset	71.17	85.59	48.05	79.87

Comparing the mIoU values obtained in this dataset, and making a comparison with the previous results, we noticed that SegNet, despite being the network with the largest number of parameters obtained only the second worst result in Aveiro's dataset, achieving only 72%, only ahead of ENet which reached 71.17%. The best values were obtained by the Fast and ContextNet models, with the models trained in CityScapes achieving very similar results, all of them obtaining more than 77% of mIoU. The ContextNet model trained on the BDD100K was the one that obtained the best result, having obtained 83% of mIoU, which was expected since this dataset has a higher number of images even being from another continent.

When analyzing in more detail some images segmented by the models trained in the two datasets, it can be perceived that the gain that was added when training with a larger dataset. In the first pair of images in figure 5.6, the model trained with the BDD100K detects the sidewalk with more precision, while the detection of the other model has much more noise. The middle images, where the image was captured while driving in a very characteristic area of the city, the network trained on CityScapes shows again a lot of noise compared to the other model. The gain obtained with the training on the Berkeley Deep Drive dataset is notorious once again when analyzing the last two images, where the ContextNet model trained on the BDD100K showed again better and smoother results.

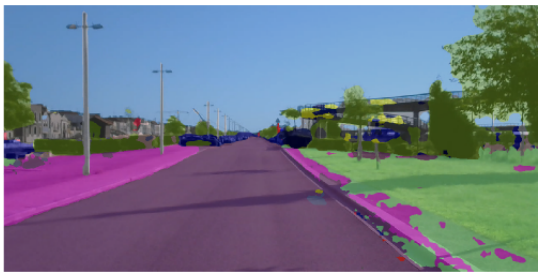
Regarding the results obtained in the classes in these last two architectures, it was

(a) Segmented image with ContextNet 2x model trained on CityScapes

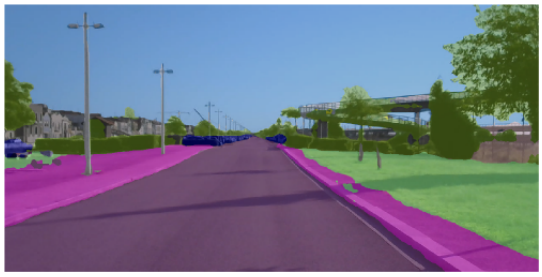
(b) Segmented image with ContextNet 2x model trained on BDD100K

(c) Segmented image with ContextNet 2x model trained on CityScapes

(d) Segmented image with ContextNet 2x model trained on BDD100K



(e) Segmented image with ContextNet 2x model trained on CityScapes



(f) Segmented image with ContextNet 2x model trained on BDD100K

Figure 5.6: Comparison of segmented images with models trained in CityScapes dataset and Berkeley Deep Drive dataset. In general, the architecture trained in the larger dataset achieved better results.

the class of cars that maintained the most similar results, which is understandable since it is the element that has less variation globally. As for the road, this class dropped about 6% in each model and the sidewalk ended up being the class with the biggest drop, about 18%. These results are due to the fact that some roads and pedestrian

areas in the city of Aveiro are not so common or even because of the presence of small vegetation in some of them, which makes it difficult to detect for models.

Despite the accuracy gain when the model was trained with the BDD100K, there are certain situations where it presents some difficulties in identifying the analyzed classes. One of them is identified in figure 5.7, where the model can not correctly identify the sidewalks and identifies this as vegetation or road. Furthermore, in this image the network mistakenly assumes that the wall at the junction is road.

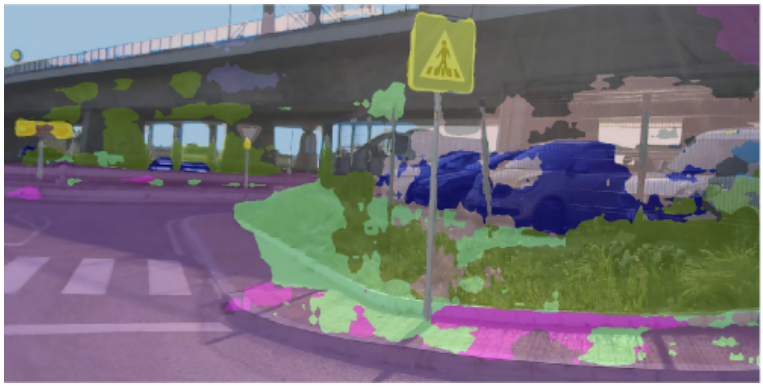
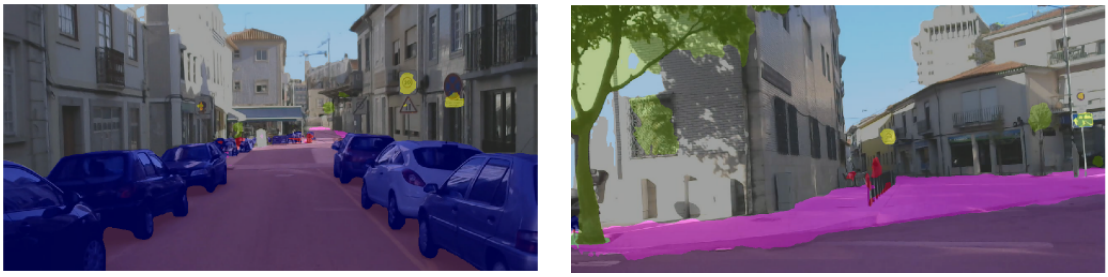


Figure 5.7: Example of poor model segmentation. In this case the model detects part of the sidewalk and the wall at the intersection as road or vegetation. In addition, it is noticeable that the model also fails to do the segmentation of the fence and the cars correctly.

Something that allows to understand that there is still a need for progression of these models is the situation presented in figure 5.8. In the first image of this figure the model correctly identifies a circulation area similar to a sidewalk as a road however, in the second image, the model assumes an area of the same type as a sidewalk.



(a) Well segmented road (b) Partially well segmented road

Figure 5.8: This figure presentes two images of roads that are similar to a sidewalk. In the left image the model correctly identifies a large part of the road, however in the second case it assumes a driving zone as a ride, although it is similar to the one in the previous image.

Another situation that may become critical is presented in figure 5.9, where the model does not detect the sidewalk, and in a planning situation the car could assume that should change lanes, causing an accident.

Figure 5.9: Example of poor detection of the sidewalk. In this case, the model detects a part of the sidewalk correctly, however it does not detect the continuation of that sidewalk after the pole. This could lead to a collision, since the car could assume that it should switch to the lane further to the right.

5.4.3 Performance Evaluation

To evaluate the performance of the trained architectures one of the GPU's of the LAR workstation was used. In table 5.3 the results are presented for different sizes of images in all architectures.

Table 5.3: Performance comparison between models. The values of the frames per second obtained in three image sizes by the models are presented. Models with the same architecture were omitted since their performance is similar.

Architecture	**Training Dataset**	**FPS for 2048x1024px**	**FPS for 1024x512px**	**FPS for 640x480px**
ContextNet	CityScapes	124.00	221.75	223.39
ContextNet 2x	CityScapes	59.30	205.88	218.42
Fast S-CNN	CityScapes	140.60	245.23	248.68
SegNet	CityScapes	6.87	27.22	43.53
ENet	CityScapes	28.11	71.73	78.67

The architecture that got the worst frame rate was SegNet with 6.87 frames-per-second for a 2048x1024 pixels image, 20 times slower than Fast S-CNN. Although ENet is an architecture with few parameters, it presented the second worst inference result in all tests, achieving up to 78 FPS in images with a size of 640x480. The architectures

that showed the most identical results were ContextNet and Fast S-CNN that reached more than 120 FPS for all image sizes, the latter achieving a maximum of 248 FPS for the smallest image size. The ContexNet model with twice the parameters reached a minimum of 59 FPS for the largest image size, about half of what was achieved by the normal architecture, and approximated for a smaller image size.

By making an overall analysis of what was the performance and accuracy of the architectures used, we can conclude that ENet and SegNet achieved the least satisfactory results on both sides, while ContextNet and Fast S-CNN achieved quite interesting results.

5.5 Panoramic Segmentation Evaluation

The way of creating the segmented panoramic image was also evaluated in order to understand which form has the best balance between performance and quality. As mentioned in the previous chapter, some solutions were explored that would allow the creation of the segmented panoramic image to be more efficient, including a representation through polygons or the points present in the masks of a given class.

Table 5.4 shows the results of the times obtained during the creation of the panoramic images in the different ways, and having this test been performed in a Jupyter Notebook, the time of creation of the panoramic image is presented in order to have a time for comparison.

Table 5.4: Performance comparison in the creation of the segmented panoramic image.

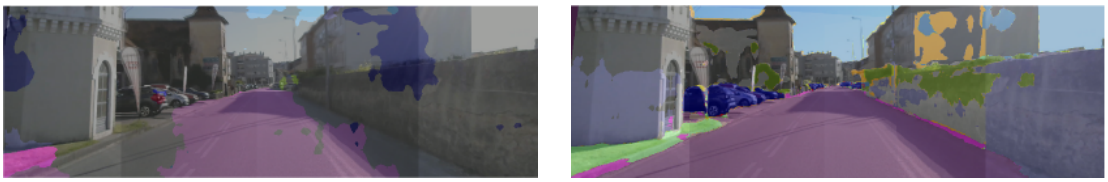
Panoramic image method	**Time per image [s]**
Panoramic image	1.80
Segmentation with argmax	31.05
Panoramic segmentation	2.23
Panoramic from segmented images	2.74
Panoramic mask with points	1.13
Panoramic mask with polygon lines	1.14
Panoramic mask with polygon	0.96

Before comparing the results of the several ways of creating the segmented panorama, and looking at the time values of image creation presented, it can be seen that these values are not of the same magnitude as those obtained in some of the tests performed previously. This disparity is due to the fact that these tests were executed on the LAR workstation and PyTorch was used in the task of creating the panoramic image, however this will not be the operational environment in a solution on board ATLASCAR2. This library is slower than Numpy to perform simpler operations but, having performed the tests under the same conditions, the values obtained allow some conclusions.

Of all forms presented, the slowest is the creation of the segmented panoramic image with agrmax, in which only the class of each pixel is calculated at the end of the junction of the images, thus having to apply the transformations to all classes. This means that the program for creating the panoramic image takes an image with the dimension of $C \times H \times W$, where H and W represent the height and width of the image respectively, and C represents the number of classes, and the transformations are applied C times. In this case, the transformations were applied 19 times and then the maximum value of each pixel in the C layers was calculated, determining to which class each pixel belongs. Although the processing time is too high, qualitatively, this way of creating the segmentation of the panoramic image presents good results as seen in figure 5.10.

Figure 5.10: Segmented panoramic image with argmax.

The times achieved by the segmentation of the panoramic image or by the creation of the panoramic through the segmented individual images were similar, and although the former has a better processing time, the segmentation is quite weak as shown in the first image of figure 5.11. This is due to the fact that the colours of the panoramic image are not rectified. The segmented panorama created through the individual images presents a better segmentation than the previous one as can be seen in the second image of figure 5.11, and although when compared with the image 5.10 the results are not so good, the ratio between quality and time of segmentation is better.



(a) Panoramic image segmentation (b) Panoramic image from segmented single images

Figure 5.11: Segmented panoramic image. On the left is the segmented image generated from the panoramic image segmentation and on the right is the image created by combining the individual segmented images.

The alternatives presented to these methods of panoramic image segmentation proved to be much more efficient, with the representation of a given class by polygons reaching almost half of the processing time of the panoramic image. Among these options, the one that presents the best result in terms of representation of a given class is the creation of the panoramic mask using all the points of the individual images belonging to that class and applies the transformations to them. The other two ways create a polygon through the set of points that allow it to contain the class mask inside and with this introduce small image segments that do not belong to the evaluated class, which can create some kind of navigation problems. In figure 5.12 are represented two of the alternatives proposed for the creation of the panoramic mask of a class.

(a) Road mask through points (b) Road mask through polygon

Figure 5.12: The image on the left shows the creation of the panoramic road mask through the points of this class, and on the right the polygon that best represents it.

5.6 Real time experiments

As mentioned in the first chapter of this dissertation, one of the objectives was to perform inference tests in a real case. For that, and having chosen Jetson Xavier as hardware to implement the model in ATLASCAR2, we started by exporting the trained models and inserting them into the Jetson Inference library so that our ROS package could locate and use them.

When starting the inference tests of our models on the selected hardware it was noticed that there was an incompatibility between the version of TensorRT needed to run the trained models and the one installed on the hardware, so there was a need to install a new version of TensorRT on Jetson AGX Xavier. Since it is necessary to flash the device in order to do this update and the cameras that are currently connected to it require some time for configuration, it was decided to use the segmentation models that are available in the Jetson Inference library in order to perform the road tests.

During the circulation through the streets of Aveiro two videos were recorded[2], where the first is related to the creation of the panoramic image through the segmented single images and the second is the segmentation of the panoramic image. In these tests, the images were acquired by the cameras with a size of 640x480x or 1280x720 pixels and the library model used for the tests was the FCN trained in CityScapes. It was found that

[2]Panoramic image segmentation on board the ATLASCAR2. `https://www.youtube.com/watch?v=ggD8k_pGEh4&t`.

this model was able to segment the 3 images at a frame rate of 30 FPS to the smallest image size and 20 FPS when the images were larger.

Regarding the process of creating the panoramic image, it was only possible to achieve a publication in the ROS 10 FPS topics either in the creation of the panoramic image through the individual segmented images or in the segmentation of the panoramic image. The other segmentation options of the panoramic image presented in this dissertation were not implemented due to the lack of some necessary libraries. However, and although the images pick up some part of the car interior and there is some reflection on the windscreen, we can see in figure 5.13 that the creation of the panoramic image through the segmentation of the individual images, excluding the noise caused by the car interior, shows again better results than the segmentation of the panoramic image. This result is in line with the previous section where the segmentation of the panoramic image also showed the poorer results.

Figure 5.13: Tests performed on board ATLASCAR2. The first image is relative to the panoramic created by merging the segmented images and the second image is the result of the segmentation of the panoramic image.

5.7 Summary

In this chapter the tests that were performed throughout the dissertation were presented. Firstly, a small evaluation of the creation of the panoramic image was made, which defined the need to explore new approaches to its creation. In addition, the work carried out in the development of the validation dataset of the city of Aveiro was presented, which was very important for the evaluation process of the trained models. The process of training the architectures was briefly explained and the models were evaluated taking

into account their accuracy and performance. Finally some solutions for the creation of the segmented panoramic image were presented and the tests performed on board the ATLASCAR2 were evaluated.

Intentionally blank page.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

For an autonomous vehicle to drive safely, it is essential that the vehicle has perception of its surroundings. The main objective of this dissertation was the creation of a model capable of identifying road or other objects in this environment. For this, panoramic images were used and to segment them, models trained with Deep Learning were used.

To acquire the panoramic images on board the ATLASCAR2, and since the camera ring for the car was not yet developed, it was necessary to study the creation of this type of images. Therefore, the positioning of the cameras was studied and a function was developed for the creation of this type of image, enabling a panoramic image to be obtained at 10 FPS on board the ATLASCAR2.

The biggest challenge of this dissertation was the creation of segmentation models using Deep Learning, and so there was the need to explore this area in order to be able to propose datasets and architectures that were suitable for the segmentation of images in real time. The training of the proposed models was performed in two different datasets and four different architectures were explored, with a validation in a dataset created with images of the city of Aveiro, where positive results were obtained in terms of road and car segmentation. The performance of the proposed models was also evaluated, obtaining up to 140 FPS in a 2048x1024 pixels image, during inference.

For the inference process, a ROS package was developed which allowed the segmentation of the panoramic image or its creation through the individual segmented images. Considering the testing process on board the ATLASCAR2, it can be concluded that the segmented panoramic image has higher quality when choosing to create it through the segmented individual images. Three distinct ways of representing the panoramic mask of a given class were also presented with the objective of reducing the processing time in the creation of the panoramic image.

Finally, the best solution found in this dissertation combines ContextNet with twice the size trained on the BDD100K and the formation of the panoramic image through the individual segmented images, if all classes have to be visualized, or through the use of polygons, if only one class representation is required.

The whole process of the work developed during this dissertation is available on the dissertation blog, `https://rubendfcosta.github.io/MasterThesisBlog/`, and the code is documented at `https://github.com/lardemua/AtlasCar2PanoramicDetection`.

6.2 Future Work

The work developed during this dissertation combined different contents that can be further developed. The field of creating the panoramic image can be explored in order to decrease the time needed for this task and consequently increase the frame rate with which the image is published, which would make the ATLASCAR2 motion planning process more secure. Also related to the panoramic image, the calibration of the camera ring that is under development will be fundamental for the creation of a panoramic image with higher quality than those used in this dissertation.

Regarding the use of Deep Learning, it is necessary to train architectures with larger datasets than the selected ones, as observed with the model trained with Berkeley DeepDrive dataset, which presented better results in this dissertation. The task of panoptic segmentation presented in the second chapter of the dissertation can also be explored.

The three proposed solutions for viewing the panoramic masks can be improved, especially in the representation of the masks through polygons, as they showed a reduction in the time needed to create the panoramic image and can be useful for the task of route planning.

References

[1] "Road Mobility and Transport." `https://ec.europa.eu/transport/themes/its/road_it` Acessed: 2020-05-11.

[2] "Hours spent in road congestion annually," Oct. 2016. `https://ec.europa.eu/transport/facts-fundings/scoreboard/compare/energy-union-innovation/road-congestion_en` Acessed: 2020-05-11.

[3] "ATLAS project." `http://atlas.web.ua.pt/index.html` Acessed:2020-05-07.

[4] T. Birdal and A. Erçil, "Real-time automated road, lane and car detection for autonomous driving," in *DSPincars, Istanbul*, 2007.

[5] T.-Y. Sun, S.-J. Tsai, and V. Chan, "HSI color model based lane-marking detection," in *2006 IEEE Intelligent Transportation Systems Conference*, pp. 1168–1172, Sept. 2006. ISSN: 2153-0017.

[6] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886–893 vol. 1, June 2005. ISSN: 1063-6919.

[7] D. Lowe, "Object Recognition from Local Scale-Invariant Features," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2, Jan. 2001.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[9] N. O. Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. Velasco-Hernandez, L. Krpalkova, D. Riordan, and J. Walsh, "Deep Learning vs. Traditional Computer Vision," *arXiv:1910.13796 [cs]*, vol. 943, 2020. arXiv: 1910.13796.

[10] A. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech Recognition Using Deep Neural Networks: A Systematic Review," *IEEE Access*, vol. PP, pp. 1–1, Feb. 2019.

[11] D. W. Otter, J. R. Medina, and J. K. Kalita, "A Survey of the Usages of Deep Learning in Natural Language Processing," *arXiv:1807.10854 [cs]*, Dec. 2019. arXiv: 1807.10854.

[12] J. Patterson and A. Gibson, *Deep Learning: A Practitioner's Approach*. Beijing: O'Reilly, 2017.

[13] "Complete Guide to Artificial Neural Network Concepts & Models." https://missinglink.ai/guides/neural-network-concepts/complete-guide-artificial-neural-networks/ Acessed: 2020-06-12.

[14] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning," *arXiv:1811.03378 [cs]*, Nov. 2018. arXiv: 1811.03378.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *arXiv:1502.01852 [cs]*, Feb. 2015. arXiv: 1502.01852.

[16] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for Activation Functions," *arXiv:1710.05941 [cs]*, Oct. 2017. arXiv: 1710.05941 version: 1.

[17] Y. Ho and S. Wookey, "The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling," *IEEE Access*, vol. 8, pp. 4806–4813, 2020. arXiv: 2001.00570.

[18] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation," *arXiv:1606.04797 [cs]*, June 2016. arXiv: 1606.04797.

[19] B. Lourenço, V. Santos, M. Oliveira, and T. Almeida, "Performance Analysis on Deep Learning Semantic Segmentation with multivariate Training Procedures," pp. 89–95, Apr. 2020.

[20] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár, "Panoptic Segmentation," *arXiv:1801.00868 [cs]*, Apr. 2019. arXiv: 1801.00868.

[21] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *arXiv:1411.4038 [cs]*, Mar. 2015. arXiv: 1411.4038.

[22] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *arXiv:1511.00561 [cs]*, Oct. 2016. arXiv: 1511.00561.

[23] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation," *arXiv:1606.02147 [cs]*, June 2016. arXiv: 1606.02147.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015. arXiv: 1512.03385.

[25] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, "BiSeNet: Bilateral Segmentation Network for Real-time Semantic Segmentation," *arXiv:1808.00897 [cs]*, Aug. 2018. arXiv: 1808.00897.

[26] R. P. K. Poudel, U. Bonde, S. Liwicki, and C. Zach, "ContextNet: Exploring Context and Detail for Semantic Segmentation in Real-time," *arXiv:1805.04554 [cs]*, Nov. 2018. arXiv: 1805.04554.

[27] R. P. K. Poudel, S. Liwicki, and R. Cipolla, "Fast-SCNN: Fast Semantic Segmentation Network," *arXiv:1902.04502 [cs]*, Feb. 2019. arXiv: 1902.04502.

[28] Y. Wang, Q. Zhou, J. Liu, J. Xiong, G. Gao, X. Wu, and L. J. Latecki, "LEDNet: A Lightweight Encoder-Decoder Network for Real-Time Semantic Segmentation," *arXiv:1905.02423 [cs]*, May 2019. arXiv: 1905.02423.

[29] Y. Wang, Q. Zhou, and X. Wu, "ESNet: An Efficient Symmetric Network for Real-time Semantic Segmentation," *arXiv:1906.09826 [cs]*, June 2019. arXiv: 1906.09826.

[30] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognition Letters*, vol. 30, pp. 88–97, Jan. 2009.

[31] "Cityscapes Dataset." Library Catalog: `www.cityscapes-dataset.com` Acessed: 2020-05-26.

[32] X. Huang, P. Wang, X. Cheng, D. Zhou, Q. Geng, and R. Yang, "The ApolloScape Open Dataset for Autonomous Driving and its Application," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020. arXiv: 1803.06184.

[33] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning," *arXiv:1805.04687 [cs]*, Apr. 2020. arXiv: 1805.04687.

[34] G. Neuhold, T. Ollmann, S. R. Bulo, and P. Kontschieder, "The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes," in *2017 IEEE International Conference on Computer Vision (ICCV)*, (Venice), pp. 5000–5009, IEEE, Oct. 2017.

[35] J. Geyer, Y. Kassahun, M. Mahmudi, X. Ricou, R. Durgesh, A. S. Chung, L. Hauswald, V. H. Pham, M. Mühlegg, S. Dorn, T. Fernandez, M. Jänicke, S. Mirashi, C. Savani, M. Sturm, O. Vorobiov, M. Oelker, S. Garreis, and P. Schuberth, "A2D2: Audi Autonomous Driving Dataset," *arXiv:2004.06320 [cs, eess]*, Apr. 2020. arXiv: 2004.06320.

[36] T. Osório, "Deteção de Objectos para Carros e Pedestres Através de Deep Learning," Master's thesis, Universidade de Aveiro, Aveiro, 2018.

[37] T. Almeida, "Arquitetura Multi-Câmara e Multi-Algoritmo para Perceção Visual a Bordo do ATLASCAR2," Master's thesis, Universidade de Aveiro, Aveiro, July 2019.

[38] "Waypoint - The official Waymo blog: Introducing the 5th-generation Waymo Driver: Informed by experience, designed for scale, engineered to tackle more environments." `https://blog.waymo.com/2020/03/introducing-5th-generation-waymo-driver.html` Acessed: 2020-05-26.

[39] S. Hecker, D. Dai, and L. Van Gool, "End-to-End Learning of Driving Models with Surround-View Cameras and Route Planners," *arXiv:1803.10158 [cs]*, Aug. 2018. arXiv: 1803.10158.

[40] M. Brown and D. G. Lowe, "Automatic Panoramic Image Stitching using Invariant Features," *International Journal of Computer Vision*, vol. 74, pp. 59–73, Apr. 2007.

[41] H. Chau and R. Karol, "Robust Panoramic Image Stitching," tech. rep., Department of Aeronautics and Astronautics, Stanford University, 2015. CS231A Course Final Report.

[42] P. Kale and K. R. Singh, "A Technical Analysis of Image Stitching Algorithm," *International Journal of Computer Science and Information Technologies*, vol. 6, p. 5, 2015.

[43] N. Appiah and N. Bandaru, "Obstacle detection using stereo vision for self-driving cars," tech. rep., Department of Mechanical Engineering, Stanford University, 2016. CS232 Course Final Report.

[44] Y. Xu, K. Wang, K. Yang, D. Sun, and J. Fu, "Semantic segmentation of panoramic images using a synthetic dataset," in *Artificial Intelligence and Machine Learning in Defense Applications* (J. Dijk, ed.), (Strasbourg, France), p. 9, SPIE, Sept. 2019.

[45] K. Yang, X. Hu, L. M. Bergasa, E. Romera, X. Huang, D. Sun, and K. Wang, "Can we PASS beyond the Field of View? Panoramic Annular Semantic Segmentation for Real-World Surrounding Perception," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, (Paris, France), pp. 446–453, IEEE, June 2019.

[46] W. Zhou, A. Zyner, S. Worrall, and E. Nebot, "Adapting Semantic Segmentation Models for Changes in Illumination and Camera Perspective," *IEEE Robotics and Automation Letters*, vol. 4, pp. 461–468, Apr. 2019. arXiv: 1809.04730.

[47] Y. Li, G. Tong, H. Gao, Y. Wang, L. Zhang, and H. Chen, "Pano-RSOD: A Dataset and Benchmark for Panoramic Road Scene Object Detection," *Electronics*, vol. 8, p. 329, Mar. 2019. Number: 3 Publisher: Multidisciplinary Digital Publishing Institute.

[48] T. Almeida, V. Santos, B. Lourenço, and P. Fonseca, "Detection of data matrix encoded landmarks in unstructured environments using deep learning," in *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 74–80, 2020.

[49] "ROS.org | About ROS." `https://www.ros.org/about-ros/` Acessed: 2020-06-03.

[50] "Overview — JupyterLab 2.1.0 documentation." `https://jupyterlab.readthedocs.io/en/latest/getting_started/overview.html` Acessed: 2020-06-03.

[51] E. Stevens and L. Antiga, "Deep Learning with PyTorch," p. 141.

[52] "OpenCV." `https://opencv.org/about/` Acessed: 2020-06-03.

[53] "Introduction — Kornia documentation." `https://kornia.readthedocs.io/en/latest/introduction.html` Acessed: 2020-06-03.

[54] D. Franklin, "dusty-nv/ros_deep_learning." https://github.com/dusty-nv/ros_deep_learning Acessed: 2020-07-05.

[55] "dusty-nv/jetson-inference: Hello AI World guide to deploying deep-learning inference networks and deep vision primitives with TensorRT and NVIDIA Jetson.." https://github.com/dusty-nv/jetson-inference Acessed: 2020-07-05.

[56] "Feature Matching — OpenCV-Python Tutorials 1 documentation." https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html Acessed: 2020-06-04.

[57] "Image segmentation with Labelbox." https://labelbox.com/product/image-segmentation Acessed: 2020-05-06.

Intentionally blank page.

Appendix A

Inference Instructions on Jetson AGX Xavier

A.1 Packages and preparation

To make the inference on Jetson AGX Xavier the following packages are required:

- Panoramic_segmentation package
- Jetson_Inference package

To run the panoramic_segmentation package it is necessary to install ROS Melodic. It can be done by running the following command on terminal:

```
sudo apt install ros-melodic-desktop-full
```

After this the installation steps present in `http://wiki.ros.org/melodic/Installation/Ubuntu` must be followed to configure the catkin workspace. The first package can be found in `https://github.com/lardemua/AtlasCar2PanoramicDetection` and must be under the catkin_ws folder. The latter package can be found in `https://github.com/dusty-nv/jetson-inference` and must be place in the home folder.

If the multi-camera board for the Jetson AGX Xavier is used, the calibration shown in the next sub-section must be made, otherwise this process is not necessary.

A.1.1 Camera profile calibration

To proceed with the calibration of the cameras, a terminal should be opened and the following commands should be followed:

- `cd /`
- `cd usr/local/ecam_tk1/bin`
- `./ecam_tk1_gucview -d /dev/video*`, where * is the number of the camera to be calibrated

After running these commands in the terminal, simply configure the camera with one of the profiles already created or create a new one with the help of the user interface. This process should be performed for all cameras.

A.2 Inference

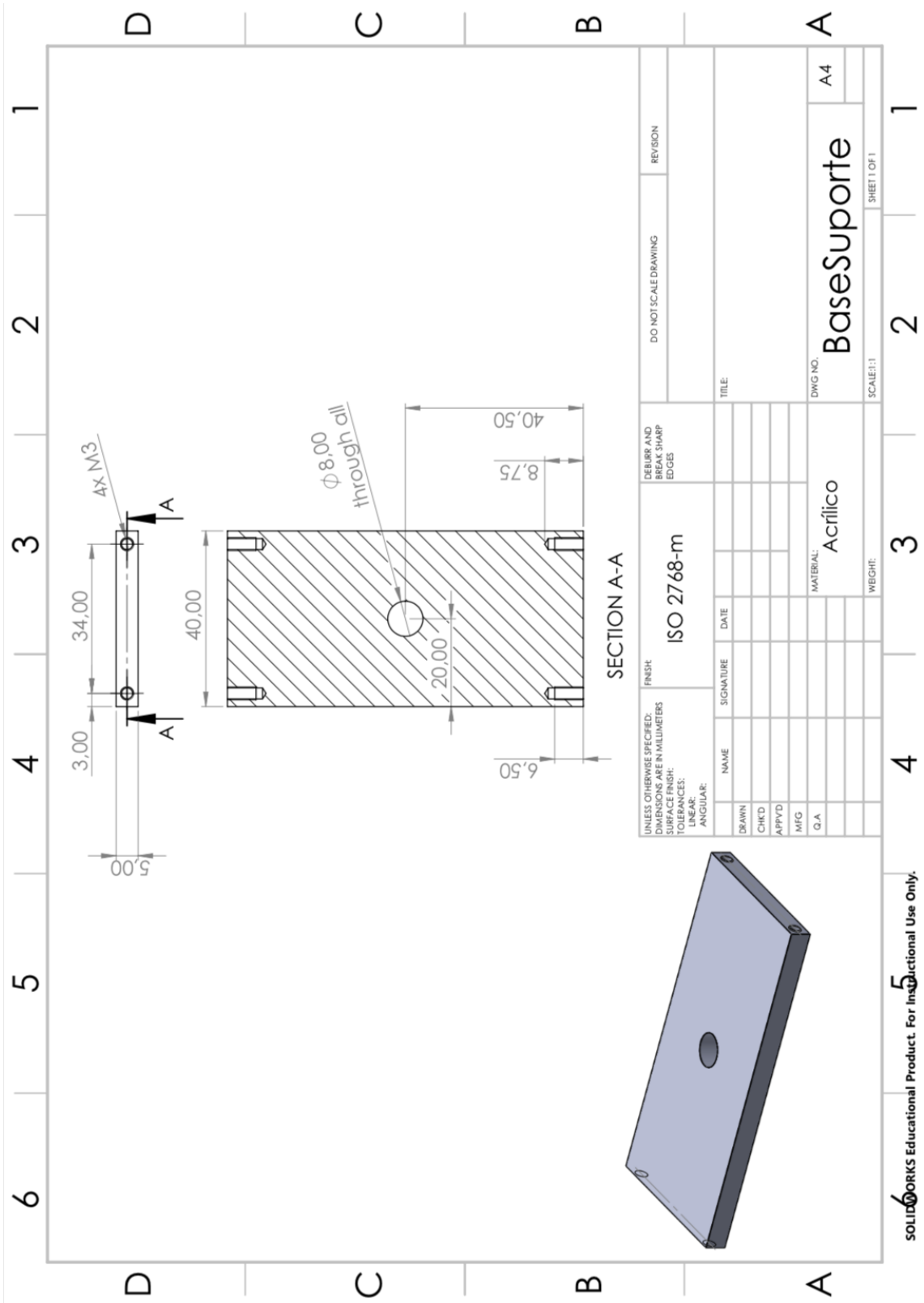
To launch the packages the following commands must be run on different terminals:

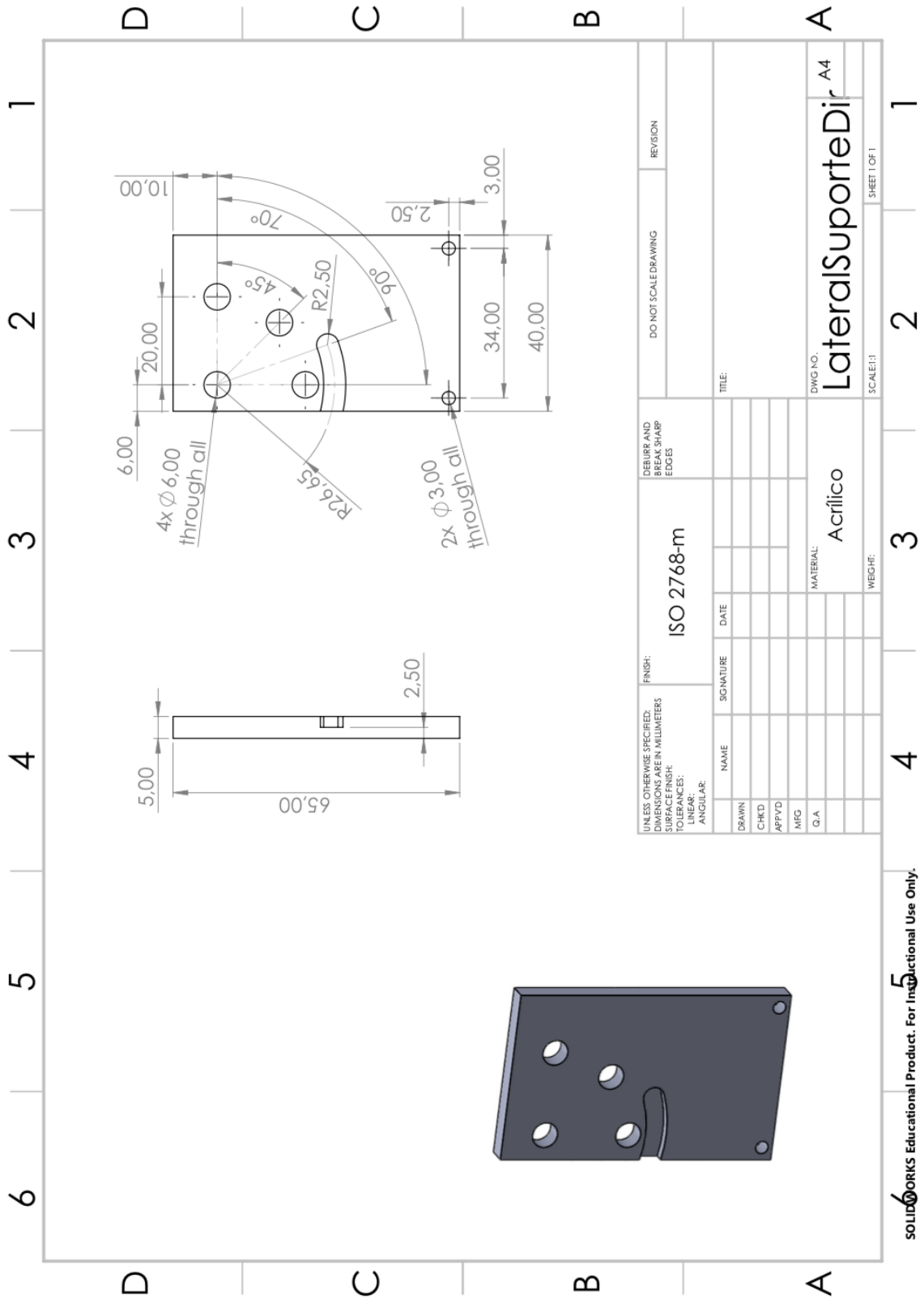
- `roscore` so that the ROS nodes can communicate
- `roslaunch panoramic_segmentation run.launch` to start acquiring images with the cameras
- `rosrun panoramic_segmentation segnet _model_name:=FCN-ResNet18-Cityscapes-1024x512` to launch the program that will create the segmented panoramic image

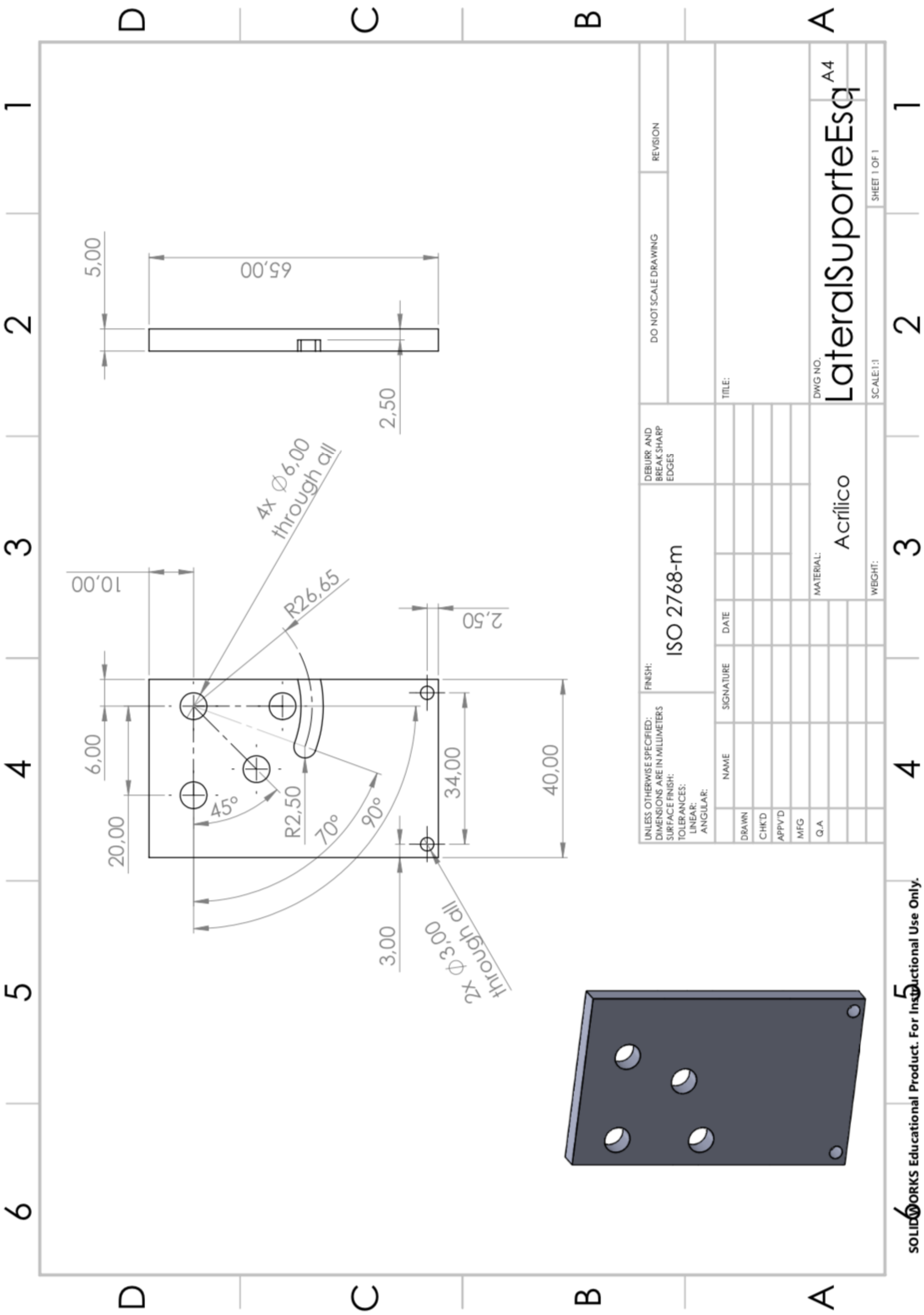
In this last command, the `_model_name` chosen can be another one as long as it is present inside the *jetson-inference/data/networks* folder. The images topics can be viewed by launching `rqt` in the terminal.

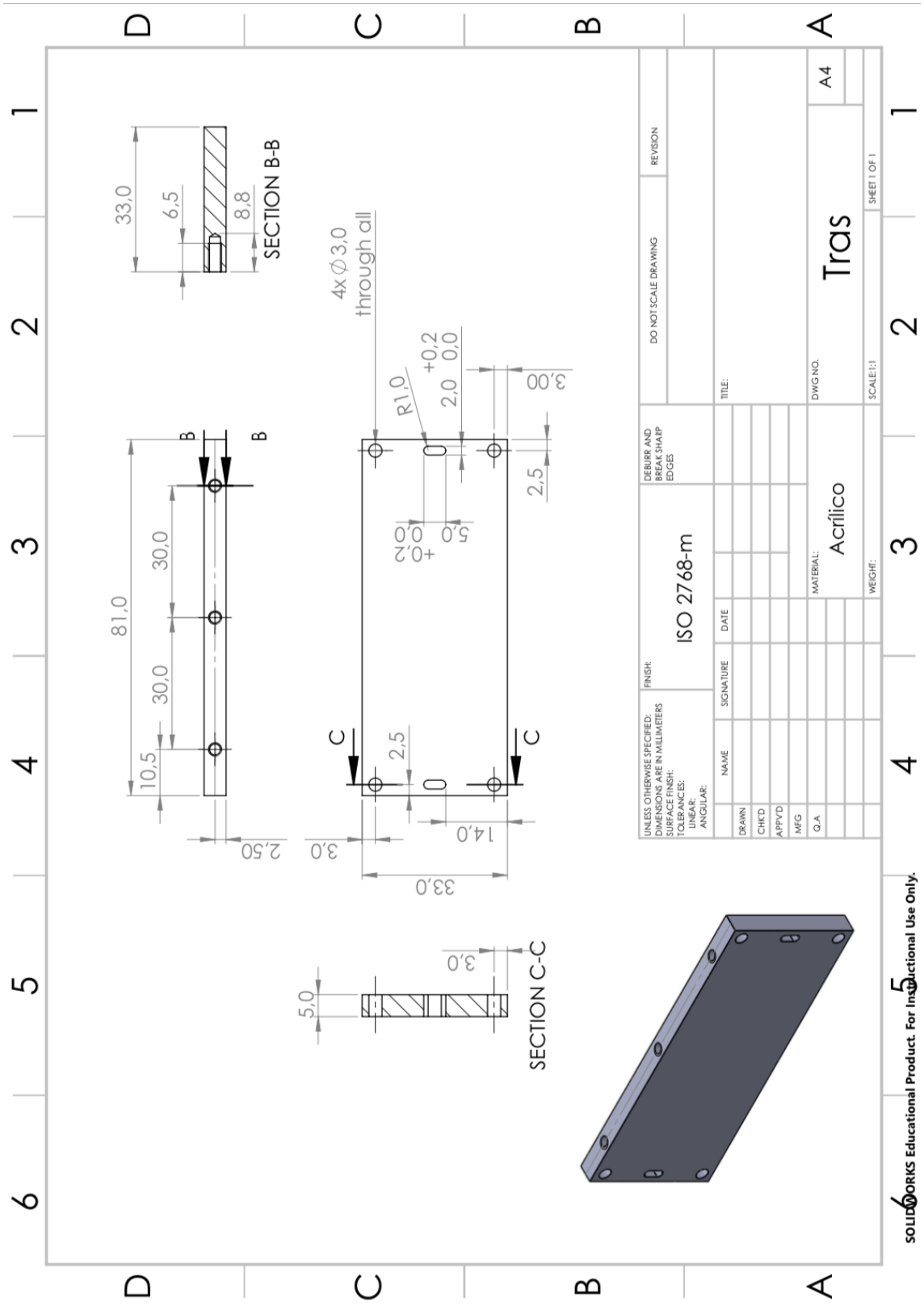
Appendix B

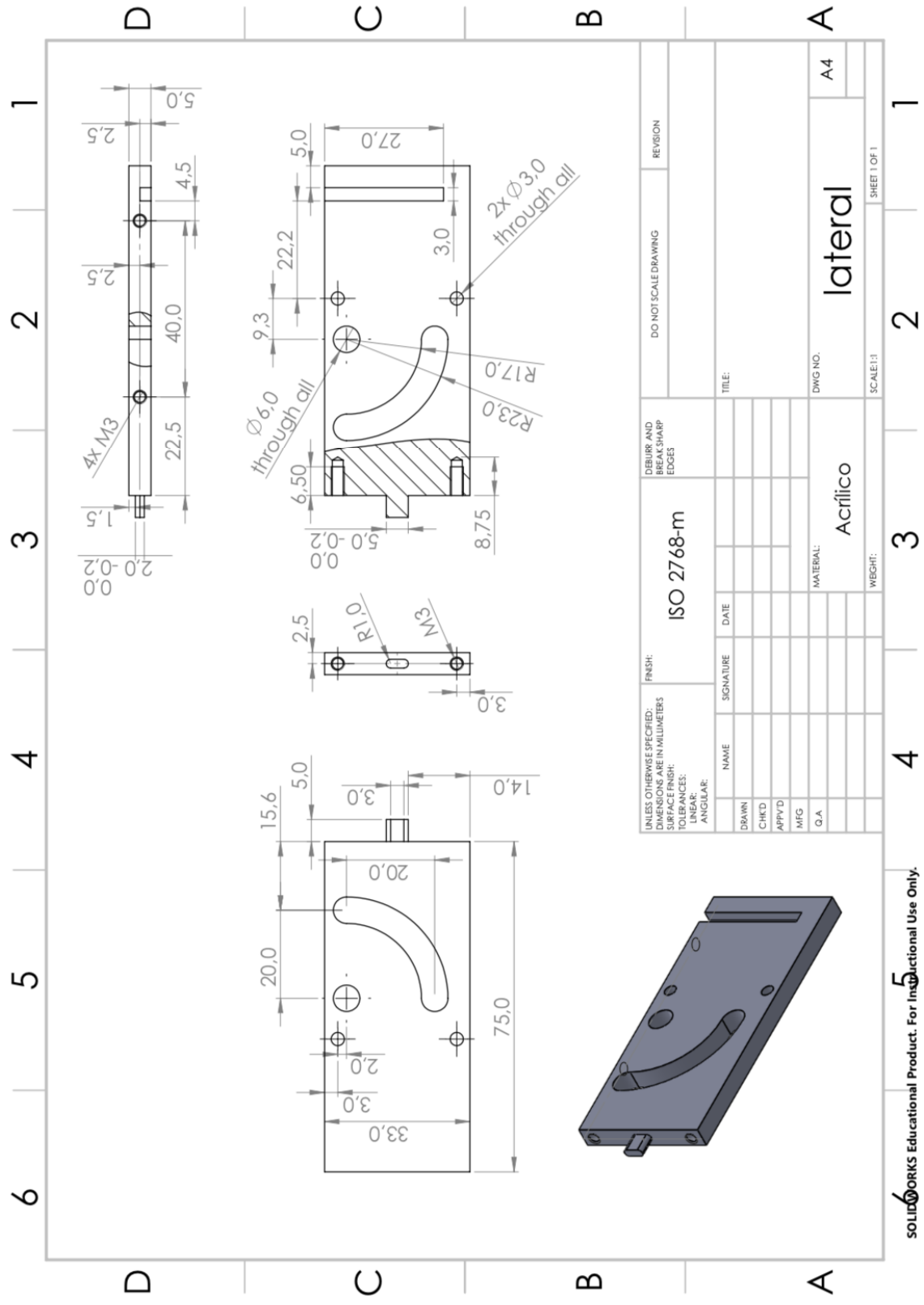
Camera Support

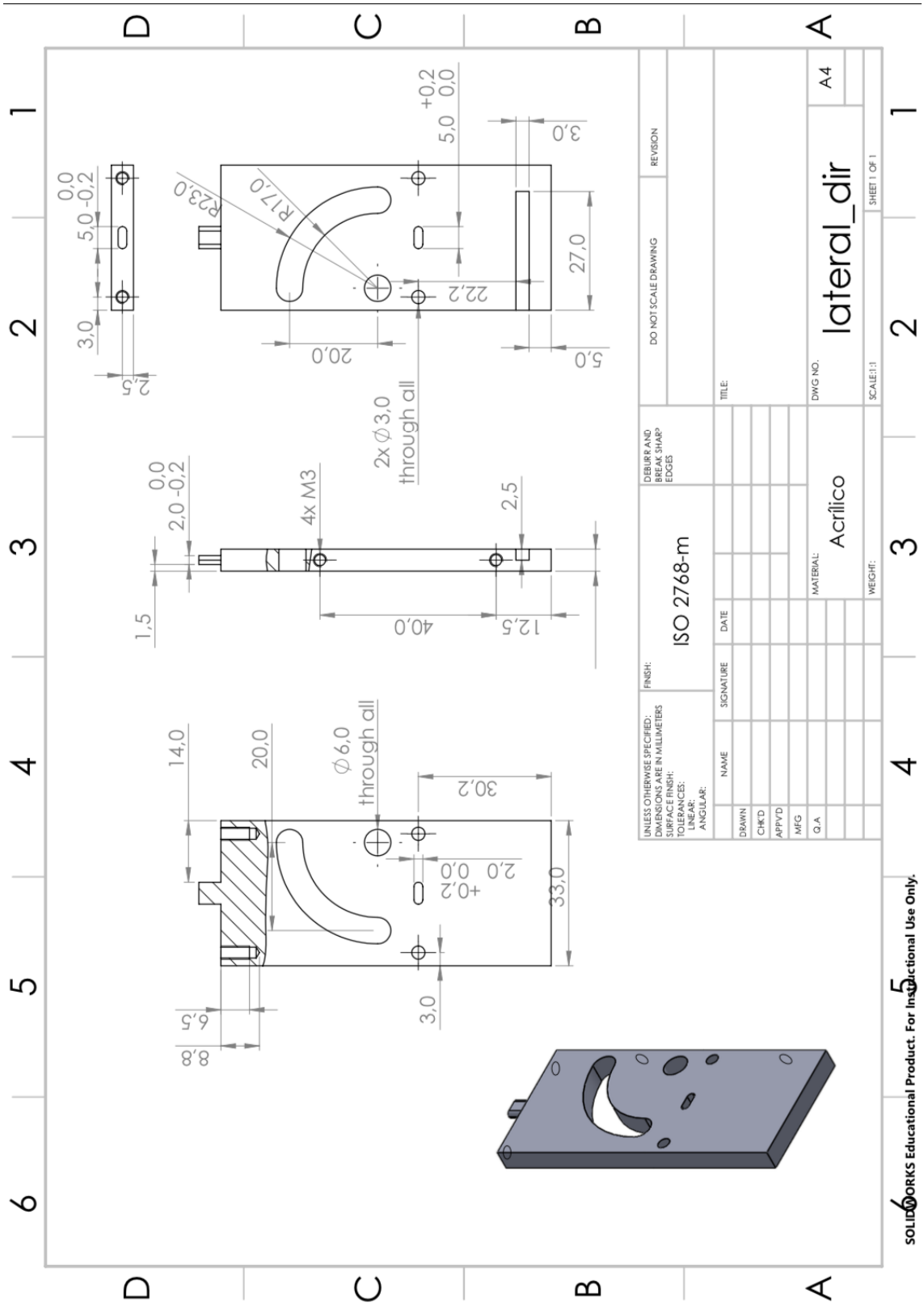

SECTION A-A
4x M3
ϕ8,00
through all
40,50
8,75
20,00
6,50
34,00
40,00
3,00
5,00
A
A
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
ISO 2768-m
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
TITLE:
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
Acrílico
WEIGHT:
DWG NO.
BaseSuporte
A4
SCALE:1:1
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

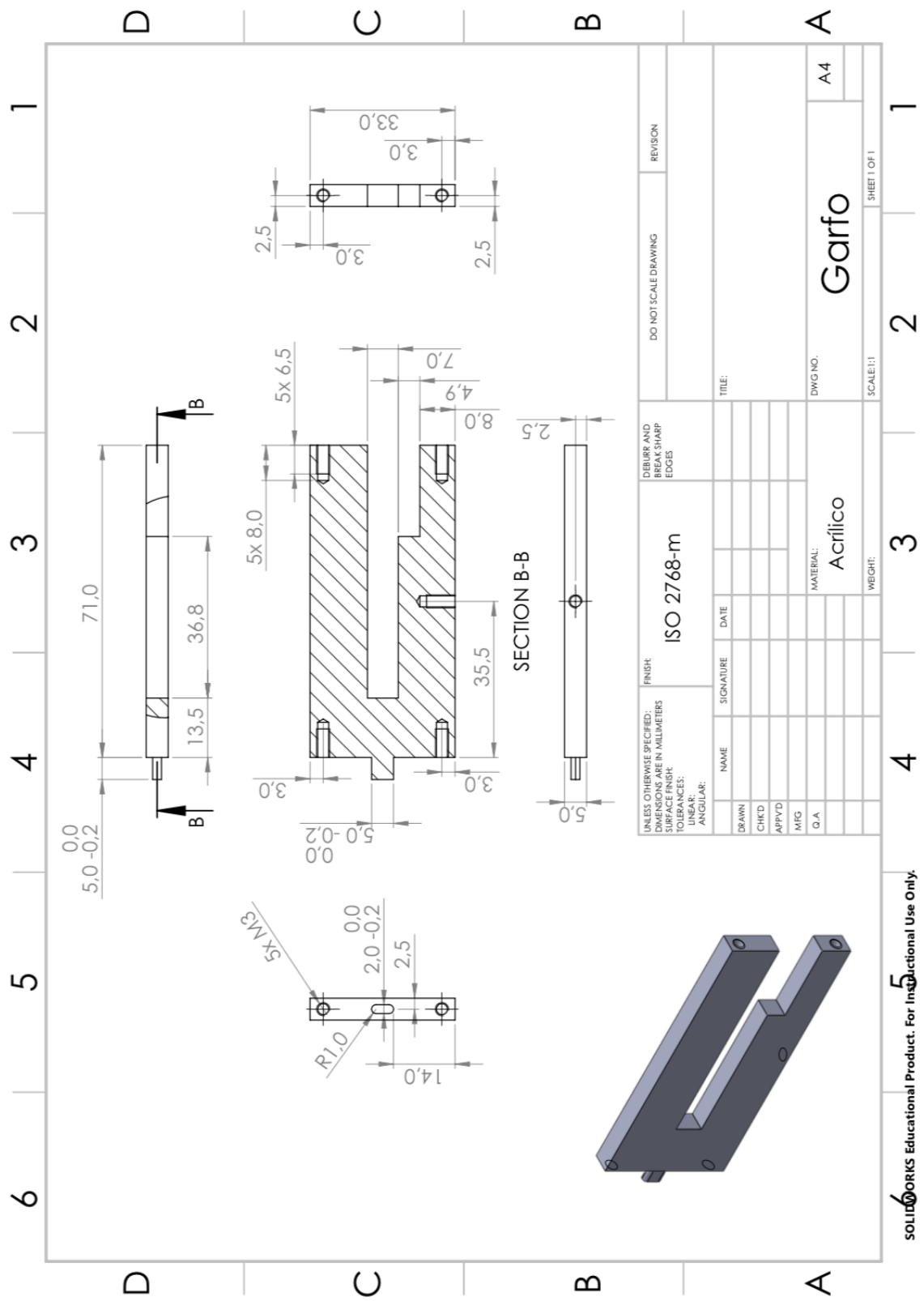

10,00
70°
2,50
3,00
45°
R2,50
90°
20,00
34,00
40,00
6,00
4x ⌀6,00 through all
R26,65
2x ⌀3,00 through all
2,50
5,00
65,00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
ISO 2768-m
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APP'VD
MFG
Q.A
MATERIAL:
Acrílico
WEIGHT:
TITLE:
DWG NO.
LateralSuporteDi
A4
SCALE:1:1
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

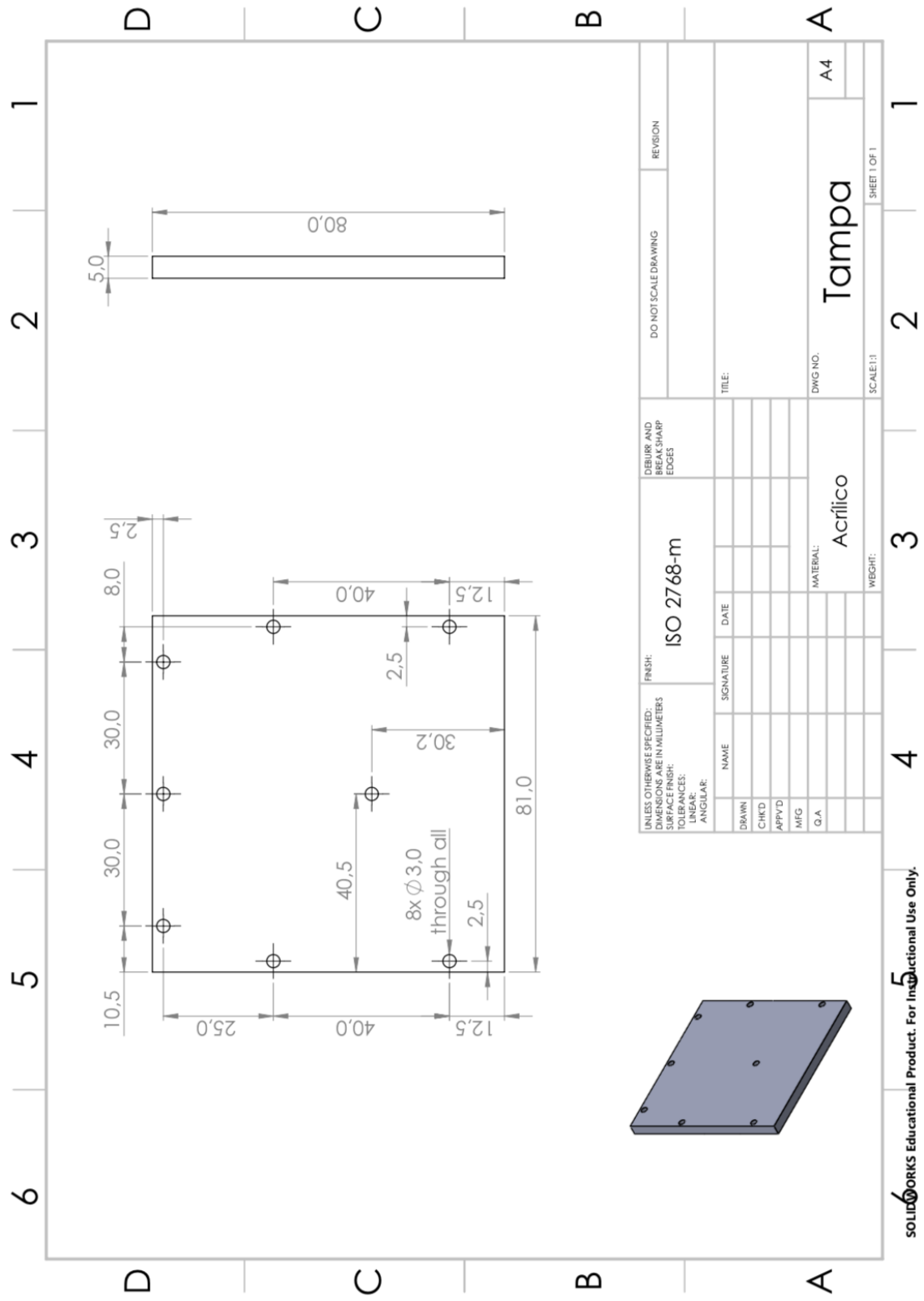

LateralSuporteEsq
Acrílico
ISO 2768-m
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
SCALE:1:1
SHEET 1 OF 1
A4
4x Ø6,00 through all
2x Ø3,00 through all
R26,65
R2,50
65,00
40,00
34,00
20,00
10,00
6,00
5,00
3,00
2,50
45°
70°
90°
SOLIDWORKS Educational Product. For Instructional Use Only.


SECTION B-B
33,0
6,5
8,8
4x Ø3,0
through all
R1,0
+0,2
2,0 0,0
3,00
2,5
5,0 +0,2 0,0
81,0
30,0
30,0
10,5
2,50
3,0
14,0
33,0
SECTION C-C
5,0
3,0
DO NOT SCALE DRAWING
REVISION
DEBURR AND BREAK SHARP EDGES
ISO 2768-m
FINISH:
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
DWG NO.
Tras
A4
MATERIAL:
Acrílico
WEIGHT:
SCALE:1:1
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.


lateral
Acrílico
ISO 2768-m
A4
SOLIDWORKS Educational Product. For Instructional Use Only.


lateral_dir
Acrílico
ISO 2768-m
A4
DO NOT SCALE DRAWING
DEBUR AND BREAK SHARP EDGES
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
MATERIAL:
DWG NO.
TITLE:
SCALE:1:1
SHEET 1 OF 1
WEIGHT:
REVISION
DRAWN
CHK'D
APPV'D
MFG
Q.A
NAME
SIGNATURE
DATE
2x ⌀3,0 through all
4x M3
⌀6,0 through all
SOLIDWORKS Educational Product. For Instructional Use Only.


Garfo
Acrílico
ISO 2768-m
A4
SECTION B-B
SOLIDWORKS Educational Product. For Instructional Use Only.


80,0
5,0
2,5
8,0
30,0
30,0
10,5
40,0
12,5
2,5
30,2
40,5
8x ⌀3,0
through all
2,5
81,0
25,0
40,0
12,5
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
ISO 2768-m
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
Acrílico
WEIGHT:
TITLE:
DWG NO.
Tampa
A4
SCALE:1:1
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.