3D Segmentation and Localization Using Visual Cues in Uncontrolled Environments

Hanz Cuevas Velasquez

Doctor of Philosophy
Institute of Perception, Action and Behaviour
School of Informatics
University of Edinburgh
2022

Abstract

3D scene understanding is an important area in robotics, autonomous vehicles, and virtual reality. The goal of scene understanding is to recognize and localize all the objects around the agent. This is done through semantic segmentation and depth estimation. Current approaches focus on improving the robustness to solve each task but fail in making them efficient for real-time usage. This thesis presents four efficient methods for scene understanding that work in real environments. The methods also aim to provide a solution for 2D and 3D data.

The first approach presents a pipeline that combines the block matching algorithm for disparity estimation, an encoder-decoder neural network for semantic segmentation, and a refinement step that uses both outputs to complete the regions that were not labeled or did not have any disparity assigned to them. This method provides accurate results in 3D reconstruction and morphology estimation of complex structures like rose bushes. Due to the lack of datasets of rose bushes and their segmentation, we also made three large datasets. Two of them have real roses that were manually labeled, and the third one was created using a scene modeler and 3D rendering software. The last dataset aims to capture diversity, realism and obtain different types of labeling.

The second contribution provides a strategy for real-time rose pruning using visual servoing of a robotic arm and our previous approach. Current methods obtain the structure of the plant and plan the cutting trajectory using only a global planner and assume a constant background. Our method works in real environments and uses visual feedback to refine the location of the cutting targets and modify the planned trajectory. The proposed visual servoing allows the robot to reach the cutting points 94% of the time. This is an improvement compared to only using a global planner without visual feedback, which reaches the targets 50% of the time. To the best of our knowledge, this is the first robot able to prune a complete rose bush in a natural environment.

Recent deep learning image segmentation and disparity estimation networks provide accurate results. However, most of these methods are computationally expensive, which makes them impractical for real-time tasks. Our third contribution uses multi-task learning to learn the image segmentation and disparity estimation together end-to-end. The experiments show that our network has at most $1/3$ of the parameters of the state-of-the-art of each individual task and still provides competitive results.

The last contribution explores the area of scene understanding using 3D data. Recent approaches use point-based networks to do point cloud segmentation and find local relations between points using only the latent features provided by the network, omitting the geometric information from the point clouds. Our approach aggregates the geometric information into the network. Given that the geometric and latent features are different, our network also uses a two-headed attention mechanism to do local aggregation at the latent and geometric level. This additional information helps the network to obtain a more accurate semantic segmentation, in real point cloud data, using fewer parameters than current methods. Overall, the method obtains the state-of-the-art segmentation in the real datasets S3DIS with 69.2% and competitive results in the ModelNet40 and ShapeNetPart datasets.

Lay Summary

The goal of the thesis is to help machines to see and understand their environment better. The proposed methods focus on recognizing the objects around the robot and knowing how far they are. Because the robot can have different sensors, the thesis proposes approaches for color images and 3D data. The latter is the representation of an object in 3 dimensions (x,y,z).

First, a method that finds the number of branches of a plant and their diameter is proposed. This approach uses color images, from a stereo camera (two cameras in parallel), as input to locate the position of rose bushes through a refinement process. Then, the number and width of the branches, and the skeleton of the plant are recovered. This technique works with a variety of rose bushes in real environments.

The second contribution is a pipeline for a robotic arm that finds the location of the rose stems to cut them based on gardening rules. The cutting process is obtained by finding key locations on the plant. Our approach is fast enough that the robot can update the cutting locations while the robot moves towards them. This feedback on the cutting locations modifies the trajectory of the robot, which improves the cutting success rate.

Lately, deep neural networks have achieved significant improvement in segmentation and localization tasks. However, they use a lot of memory from the computer, which makes them impractical for real-time use. Our third contribution is a light neural network that learns both tasks jointly and uses the descriptors of one task to improve the other, and vice versa.

The last contribution focuses on locating objects in a 3D scene. Our method extracts local information using two types of data: 3D positions and features learned by the network. This information is then combined and processed further to obtain better descriptions that can differentiate objects in a real scene, with better accuracy than the current methods.

Acknowledgements

I would like to thank my supervisor Professor Bob Fisher for his help, support, guidance throughout these years, and for giving me the opportunity to do the PhD. I am also grateful to Dr. Javier Gallego for his interest in my work and support since the first year of my PhD.
I would like to express my gratitude to my partner ZhiWei Guo, for supporting me and being my number one fan.
Finally, I am profoundly thankful to my parents and brother, who supported me, encourage me, and made all this journey possible.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified. Some of the material presented in this thesis was published in the following papers:

- Hanz Cuevas-Velasquez, Antonio-Javier Gallego, and Robert B Fisher. Segmentation and 3d reconstruction of rose plants from stereoscopic images. *Computers and electronics in agriculture*, 2020

  Contributions of co-authors: Antonio-Javier Gallego helped with the development, experiments, and writing of the semantic segmentation algorithm in Section 3.3.1 and Section 3.5, which is 15% of the contribution of the paper. Robert B. Fisher supervised the writing and theory of the paper.

- Hanz Cuevas-Velasquez, Antonio-Javier Gallego, Radim Tylecek, Jochen Hemming, Bart Van Tuijl, Angelo Mencarelli, and Robert B Fisher. Real-time stereo visual servoing for rose pruning with robotic arm. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7050–7056. IEEE, 2020

  Contributions of co-authors: We reused the semantic segmentation network developed by Antonio-Javier Gallego. Jochen Hemming, Bart Van Tuijl, and Angelo Mencarelli designed the end-effector of the robot and the housing for the stereo camera (Section 4.2). Radim Tylecek and Robert B. Fisher supervised the experiments and writing of the paper.

- Hanz Cuevas-Velasquez, Antonio-Javier Gallego, and Robert B Fisher. Two heads are better than one: Geometric-latent attention for point cloud classification and segmentation. In *32th British Machine Vision Conference, BMVC 2021*, 2021

  Contributions of co-authors: Antonio-Javier Gallego and Robert B. Fisher supervised the writing and theory of the paper.

(*Hanz Cuevas Velasquez*)

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Finding the surrounding objects and their locations are key elements in many applications, such as scene understanding, robot navigation, autonomous driving, and augmented reality. One of the recurrent problems of these tasks is to locate the objects in a complex environment using 2D or 3D sensors (e.g. 2D images or 3D point clouds) while keeping the robustness, efficiency, and being able to work in an uncontrolled environment. These topics are the research motivation of the proposed thesis.

This chapter provides an overview of the thesis and is structured as follows. First, the problems related to 3D scene understanding are described. Then, the original contributions to solve each problem are stated. Finally, an overview of the thesis is presented.

1.1 Problem statement

Humans are extremely good at perceiving natural scenes and understanding high-level structures which helps them to navigate and interact with their environment. Robots or autonomous agents emulate this behavior by capturing their environment with 2D, 2.5D, or 3D sensors, such as cameras, stereo cameras, RGB-D sensors, and LIDARs. Then, the agents use scene understanding to extract useful information about the environment to interact with it. Two key elements to understand the environment are knowing what the objects are and where they are located. However, processing this data requires robust methods to work in a real environment and efficient implementations to help the robot interact with its surroundings in real-time.

Scene understanding includes a wide variety of tasks. Some of them involve image and pixel-level semantic labeling such as scene classification, object detection, instance segmentation, and semantic segmentation. Other tasks focus more on localization like pose estimation, depth estimation, disparity estimation, and trajectory prediction [4]. From the previous sentence, it can be observed that these tasks can be grouped into two groups. The first group is used to identify the objects in the scene or the scene itself, and the second group is used to find the position of the objects either in the 2D image or in a 3D environment. Finding the identity and the location of the objects from real-world data is a highly challenging task. An object class might span a wide range of possibilities, from different shapes, colors, sizes, and poses. For example, an object "chair" might have a different number of legs, and shape. This is called intra-class variation, as a result, object segmentation approaches have to learn a variety of object appearances to maximize the correct number of segmented objects. In the case where the input data is a 2D image, the complexity increases, because the objects can be captured from different viewpoints, and the projection from the 3D scene to the 2D image increases the number of appearances that an object might have. If the data is a point cloud, depending on the type of sensor, it might be noisy or sparse (LIDARs with few laser beams) which also adds a level of complexity. Moreover, external conditions such as illumination or low image resolution add further difficulty to the task, by limiting the useful information. Once the objects have been segmented, their position with respect to the agent can be found by using 3D sensors like LIDARs. However, these sensors are expensive and the data they capture are sparse. A cheaper and denser solution is the use of stereo cameras, where the depth is found by estimating the disparity between the left and right images, and triangulation based on the camera parameters.

A robot uses scene understanding to identify and locate the objects in the scene at the same time. Therefore, the second problem is how the agent (robot) can use both methods efficiently and robustly to interact with its environment in real-time. The problem can be approached using 2D or 3D data. If only 2D data is used, one solution is to segment the scene and estimate the depth map using the state-of-the-art methods in each area like [5] for semantic segmentation and [6] for depth and disparity estimation. This approach is robust but inefficient because these methods usually require a lot of computational power. Another approach to the problem is through muti-task learning, where the depth and segmentation tasks are learned jointly like in the works [7, 8, 9]. A third approach is to obtain the location of the objects first, and perform the semantic segmentation in the 3D domain, like in [10, 11].

An area where robotics relies on efficient scene understanding is agricultural robotics, more specifically the area of plant phenotyping. Here, scene understanding is used to extract information about the structure of plants, find specific parts and perform fruit harvesting or pruning. These are not trivial tasks, because plants present thin parts and a self-overlapping structure. For fruit harvesting, most of the methods use the color of the fruit to find its location. However, for plant pruning, all the branches have a similar appearance and might have a similar color to the background (green grass and green branches). Current methods use constant background [12] to segment the branches, and a global planner to cut a branch [13, 14]. The downside of these methods is that in case that the plant moves or the reconstruction presents some errors, the cutting might fail.

1.2 Original contributions

The goal of the thesis is to propose segmentation and localization methods that are robust and efficient which can be used in robotics applications. As mentioned in Section 1.1, there are several approaches to find and locate objects in a scene. The solutions in this thesis covers only the area of semantic segmentation to identify the objects in the scene, and disparity estimation to find their location. Also, these methods are applied to data obtained by a stereo camera. The first reason is that stereo cameras are cheaper and obtain RGB information and denser depth values than LiDAR. Also, the proposed algorithms are expected to work in outdoor environments. This favors stereo cameras over structured light sensors because outdoor illumination usually degrades the performance of structured light-based techniques [15].

This thesis has four original contributions, three methods that advance the problem of scene segmentation using 2D data, and a fourth one that focuses on 3D segmentation. The first contribution approaches the problem by using a robust but efficient image segmentation network to segment 2D images. The segmentation mask is then used to improve the disparity map obtained by a fast block-matching algorithm. The improved disparity is used to enhance the segmented mask. The second contribution applies the previous method in a visual servoing pipeline. Here, a robotic arm has to find branches of rose bushes, in a real garden, and navigate towards them to cut them based on gardening rules. The third contribution uses multi-task learning to perform semantic segmentation and disparity estimation together. The final contribution works in the 3D domain to semantically segment point clouds. More concretely, the contributions of this thesis are:

- Efficient 2D scene understanding pipeline, to reconstruct the 3D morphology and structure of rose bushes. This method presents a rose stem segmentation algorithm through Selectional Autoencoders adjusted to work on real environments with variable light conditions. The binary segmentation is combined with a disparity image to iteratively refine each other. This process is then used to obtain the 3D morphology and structure of the plant through skeletonization. This contribution was published in the journal Computers and Electronics in Agriculture 2020 [1].

- A novel rose pruning pipeline based on stereo visual servoing and real-time target update. The proposed method is a novel 2.5D servoing algorithm that combines point clouds and 2D images to find stems and cutting locations on them. This work was accepted in the International Conference on Robotics and Automation 2020 [2].

- An end-to-end multi-task learning method that successfully learns a semantic segmentation and disparity map together from a stereo pair. To achieve this, it learns task-specific features that are shared between the two tasks progressively. The proposed network solves both tasks jointly using less than $1/3$ of the parameters that the previous works use to solve only one task. This reduced number of parameters is advantageous for systems that have limited resources.

- An innovative two-headed attention layer that combines geometric and latent features to segment a 3D scene into semantically meaningful subsets. Each head combines local and global information, using either the geometric or latent features, of a neighborhood of points and uses this information to learn better local relationships. This work has been accepted in the British Machine Vision Conference 2021 [3].

1.3 Minor contributions

Part of the work in this thesis contributed to agricultural robotics. While working in this area, we observed that there is a lack of large datasets for plant and rose segmentation and disparity estimation. This motivated us to created a large dataset of real and synthetic roses called ROSeS (*Roses for Object Segmentation and Skeletonization*), which is divided into three categories: S-ROSeS, a synthetic dataset of rose bushes, H-ROSeS, a hybrid dataset with 200 real indoor images but with realistic plastic plants, and R-ROSeS, a completely real dataset with 100 photos of rose bushes taken in different botanic gardens.

H-ROSeS and R-ROSeS were captured using a stereo camera, with a resolution of 720×480 px, and a baseline of 0.03 m. The segmented ground truth was obtained by labelling manually each image at the pixel level. Because labelling roses is a hard and arduous labor, it is easy to label wrong certain parts of the image or confuse a branch with other objects in the image. Therefore, the quality of the labelling was evaluated twice to check for parts that were wrongly labelled or missing.

S-ROSeS consists of 5760 stereo images with a resolution of 720×480 px obtained from 160 different views of 36 synthetic rose bushes generated with Blender[1]. The images were generated at different distances and perspectives by rotating the camera around each plant and moving the camera closer and farther from it. Each of these bushes is unique. They were created following the morphology of real rose bushes with a mean height of 0.6m $\pm$ 0.20m. The 36 synthetic bushes are distributed in 4 different outdoor environments (9 bushes per environment). The environments differ in background, light conditions and position of the sun. Each stereo pair has its corresponding ground truth with the segmented image, the disparity map, and the skeleton for both left and right views (see Figure 3.10). These images were also generated using Blender's properties. To obtain the depth map of the synthetic dataset, we simulated a stereo camera with a parallel stereoscopic configuration, a sensor size of 32 mm, a focal length of 0.035m, and a baseline of 0.03m.

1.4 Thesis outline

The central problem of this thesis is the study of 3D object segmentation and localization. Chapter 2 provides a background and recent works in this area. Chapter 3 approaches the problem from the 2D perspective by combining stereo and disparity images for object reconstruction applied to plant phenotyping. If we want to put these 3D techniques into a real-time application for visual feedback, these methods have to be light and robust to changes, so the robot or agent can track the desired key points; this is discussed in Chapter 4. With the recent improvements of deep learning. Methods that use this approach obtain state-of-the-art results in semantic segmentation and disparity estimation. However, they are not computationally efficient. Chapter 5 de-

[1]Blender (`https://www.blender.org/`) is a free and open source 3D creation suite.

scribes our solution using multi-task learning. Previous chapters approach the scene understanding problem using 2D stereo pairs. On the contrary, Chapter 6 provides a solution for unordered 3D point clouds.

Chapter 2

Background

This chapter explores the relevant background of scene understanding addressed in this thesis, such as semantic segmentation, multi-task learning for scene understanding, and 3D segmentation. A brief history of the tasks along with the most recent techniques is also presented, as well as an overview of the applications of these techniques in real-world problems.

2.1 Semantic segmentation

Semantic segmentation is the task of labeling each pixel in an image with a corresponding class. It is one of the oldest, yet unresolved tasks of modern computer vision and, as such, has been extensively researched. Therefore, an overwhelmingly large number of approaches have been proposed, and the summary provided here is far from exhaustive. We present an overview of the most relevant works that have kept the field moving forward, with a special focus on those that are used in this thesis.

2.1.1 Early approaches

The early approaches, also known as classical methods, are algorithms that rely on domain knowledge and handcrafted features rather than neural networks. The typical segmentation process starts with image acquisition, then, the features of the image are extracted, usually at different image scales. The next step uses these features to learn patterns that can describe an object in the image, either using supervised or unsupervised learning. Finally, the result is refined through different post-processing steps [16, 17].

The most basic form of segmentation is based on the intensity value of the pixels, where all the pixels with similar intensity values are grouped into one class using clustering techniques [18, 19, 20]. This pixel intensity can be represented by different color spaces, such as gray-scale, RGB, and HSV. To reduce the color cast caused by illumination, normalized color spaces are preferred [17]. Later on, features with geometric invariant properties started to be used, some of these features are: Histogram of Oriented Gradients [21], Scale-invariant Feature Transform (SIFT) [22], and Speeded Up Robust Features (SURF) [23]. The features can also be grouped in a bag of words to describe complex objects [24]. These features are then used to train a classifier. These classifiers can be supervised: support vector machines (SVM), random forests, and conditional random fields (CRF). Supervised methods are trained using a labeled dataset to find the most likely output y given input features X $P(y|X)$. On the other hand, unsupervised methods cluster the data based on similar patterns. Therefore, these methods label with the same class the regions that share similar features. Some methods used for semantic segmentation are: k-means, mean-shift [19], graph-based image segmentation [25], random walks [26], active contour [27], and watershed segmentation [28].

2.1.2 Deep learning

It is almost a decade since the state-of-the-art in semantic segmentation has been dominated by approaches based on deep neural networks, mainly, convolutional neural networks (CNNs). A CNN is capable of learning interesting features, not only low-level features like edge and blob detectors but also high-level features which detect more complex structures in the image like in Figure 2.1.

One of the most important breakthroughs for the CNNs, in the area of computer vision, was done by Krizhevsky *et al.* [30]through an efficient GPU implementation. Even though the GPU implementation was done for a classification task, it became a solid tool for other areas in the field. [31] was one of the first works that used a CNN for image segmentation, more specifically, they used fully convolutional neural networks (FCN). This type of network uses locally connected convolutional layers, pooling layers, and up-sampling, instead of dense layers. Such properties allow the network to work with variable image sizes and make it faster to train [31]. An example of these networks can be seen in Figure 2.2. An FCN is composed of a backbone and an up-sampling layer at the end of the network. The backbone is usually another

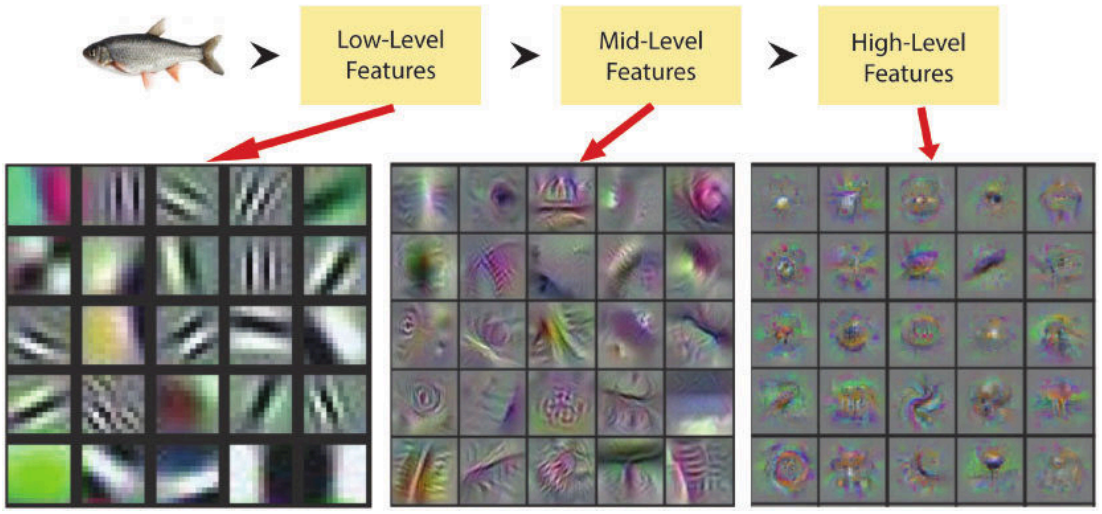


Figure 2.1: Example extracted from [29] showing the different features learned at different levels of the network.

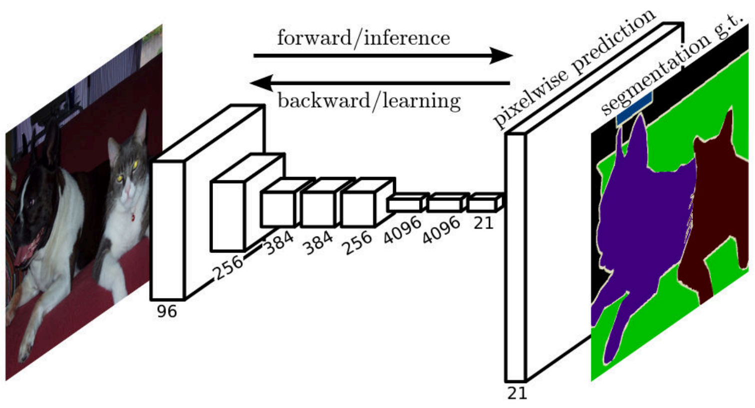


Figure 2.2: FCN architecture proposed by Long obtained from *et al.* [31].

network, trained previously for a classification problem, like VGG [32], ResNet [33], and DenseNet [34].

The next improvement in the area was proposed by [35] with their implementation of dilated convolutional layers, also known as Atrous convolutions in their network called Deeplab. This CNN replaced the last layer with a dilated version of convolutional weights. DeepLab v2 [35] added a dilated spatial pyramid pooling layer at the end of the backbone to capture features at different resolutions. DeepLab v3 [36] improved the previous model by replacing some of the normal convolutions in the backbone with their dilated layers to incorporate more context. Pyramid Scene Parsing Network (PSPNet) [37] had a similar approach to DeepLab v2 but instead of using

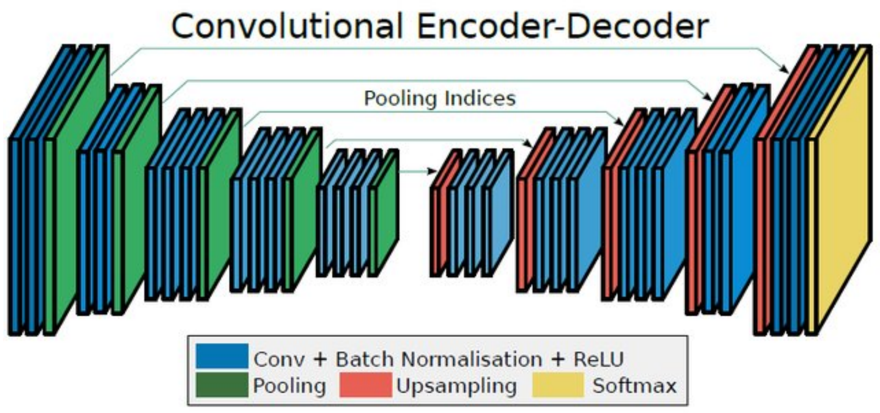

Convolutional Encoder-Decoder
Pooling Indices
Conv + Batch Normalisation + ReLU
Pooling
Upsampling
Softmax

Figure 2.3: SegNet encoder-decoder architecture taken from [39].

dilated convolutions, they used pooling layers to extract the context at different scales. Another architecture also became dominant in the area, where, instead of only up-sampling the last layer of the backbone to predict the segmented classes, a decoder network was added. The decoder allows the network to take low-resolution features and map them to features at full input resolution. DeconvNet [38] was one of the first works to use this type of architecture. For the decoder part, they proposed a de-convolution layer that associates a single input activation with multiple outputs. in SegNet [39] the decoder network has the same number of layers as the encoder part, and up-samples the features using the pooling indices from the encoder layers (Figure 2.3). A similar approach is used by U-Net [40], but instead of using the pooling indices, they copy the encoder features to their respective decoder layer. In this way, they add low-level feature information from the encoder to the decoder. DeepLab v3+ [41] combined this architecture with their Atrous layers to improve their segmentation. Other types of networks focus on improving the segmentation by including extra information or modules. For example, Global Convolution Network (GCN) [42] adds a convolutional layer with a bigger field of view (larger kernel size) between the encoder and decoder connections. On the other hand, Gated-SCNN [5] adds edge information as input. We can also consider, as extra information, the use of disparity or depth images, either as input or as a multi-task optimization problem, to improve the semantic segmentation. These types of networks are described in Section 2.2.

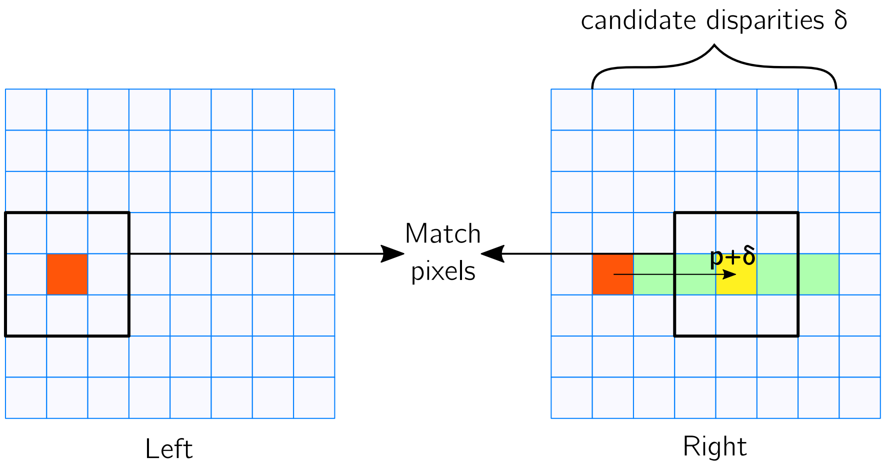


Figure 2.4: Block matching method, where a window from the left image is compared with window from the right image to find the best match.

2.2 Joint segmentation and disparity learning

This section describes the methods that use either disparity maps to improve the semantic segmentation or learn both using multi-task learning. However, before describing such methods, a brief introduction to disparity estimation will be given.

2.2.1 Disparity estimation

Disparity estimation, in stereo vision, is the process of finding the shift (disparity) of the pixels from one image (left) to another (right), as seen in Figure 2.4. Assuming that the left and right images are rectified[1], the disparity between the two images can be found by matching image features along the same image rows.

Similar to semantic segmentation, disparity methods can be separated into classical and deep learning methods. Usually, the latter perform better, but at the cost of processing time. The classical methods use block matching algorithms to find the disparities, these types of methods [43, 44, 45, 46, 47] take a window around each pixel in the left image and look for the best match in the right image (See Figure 2.4).

On the other hand, deep learning methods consider stereo matching as a learning problem. The most common architecture used for this task is an encoder-decoder network, where a backbone network is used to extract the features from the left and right images, a cost volume function calculates the similarities between feature maps, and a final network processes the resultant features to obtain the disparity. One of the first

[1]Transformation process used to project images onto a common image plane.

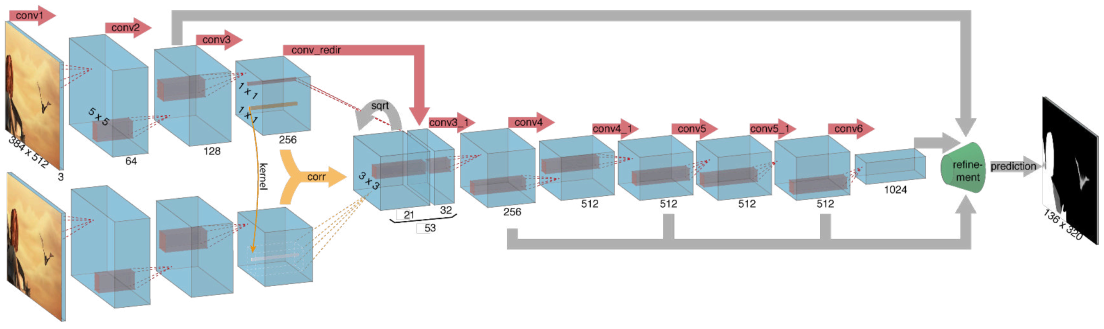


Figure 2.5: DispNet architecture taken from [48].

successful approaches was obtained by Mayer *et al.* [48], where they introduced a 1D correlation layer along the epipolar lines to calculate the similarity, or cost, between two similar feature patches, see Figure 2.5. Other approaches added context information to the cost volumes and use 3D convolutions to output the final prediction [49, 50]. Chang *et al.* [50] used pooling layers to extract the context at different scales similar to the work [37] for image segmentation.

Other approaches focus on adding new modules that improve the quality of the disparity map. [51] considers the cost aggregation as a learning process. Therefore, they add a module that generates and selects cost aggregation proposals. [52] adds a differentiable semi-global aggregation as a cost aggregation module. Other proposals focus on a progressive refinement of the disparity map, either using recurrent neural networks (RNN) like [53], checking feature consistency [54], or performing multi-scale refinement [55, 56]. These methods usually perform poorly on real datasets because they need a large number of images to learn the disparity map correctly. Because of the reduced number of data in real datasets, the networks are first trained on synthetic images and then fine-tuned with the real dataset to improve their performance. [57] proposes a domain normalization layer, which regularizes the distribution of learned representations, making them domain invariant. Thus, allowing the network to obtain good performance on real data, even when it is trained only with synthetic images and not fine-tuned. Although these end-to-end neural networks demonstrated good performance for stereo matching, they usually suffer from huge memory usage and low-speed [6].

One problem of training with stereo images is the lack of large datasets. As a result, another way to estimate the disparity between two stereo images is using unsupervised learning. This approach relies on minimizing photometric warping error. Xie *et al.* [58] predicts the right image from the left image using pixel-wise loss. Their output is a probability map, for a given range of disparity d, for each pixel. Then, the left image is shifted using the disparity probabilities to obtain the right image. [59] uses a similar approach for flow-estimation, where they add a smooth term to model the difference among the neighboring flow predictions.

2.2.2 Multi-task learning

Previous sections described the history and evolution of semantic segmentation and disparity estimation. Also, it was seen that the current state-of-the-art uses CNNs to learn each of these tasks separately. However, Zamir *et al.* [60] and Standley *et al.* [61] demonstrated that the tasks of segmentation and depth estimation (inverse disparity) have a good learning affinity, and that learning both tasks jointly can improve their performance. Because of this, some works use multi-task learning to output the semantic segmentation and disparity estimation together, using the same network[2]. Figure 2.6 shows a graphic example of the similarities among the semantic segmentation, disparity, and multi-task networks.

Multi-task learning focuses on improving the learning efficiency and prediction accuracy of multiple complementary tasks by sharing information between them [62]. If one task is similar to another, learning one will improve the performance of the other due to their similarity [60, 61]. However, if two tasks are different they will worsen their performance [61].

In the literature, there are methods that use a disparity map as extra input of a network to segment the objects in an image [63, 64, 65, 66, 67], as well as methods that use segmented images to improve the disparity estimation [68, 69, 70, 8, 71]. One of these approaches is SegStereo [8], where a pre-trained semantic segmentation network is used to obtain semantic features from the left and right images and compute the matching cost between them. The cost volume and left features are further processed by convolutional layers to predict the disparity. They added an auxiliary loss by warping the right image with the predicted disparity and predict the semantic class for each shifted pixel. [72] Uses 3 networks: a pre-trained disparity network, similar to dispNet [48], to estimate the disparity, a pretrained DeepLab network [35] to segment the left image, and a third network that uses intermediate features of the two previous networks to refine the disparity. Zhang *et al.* [73], instead of using two networks separately to predict the segmentation and disparity, train a semantic segmentation network with

[2]Multi-task CNNs usually output the segmentation and disparity of the left image, unless stated otherwise.

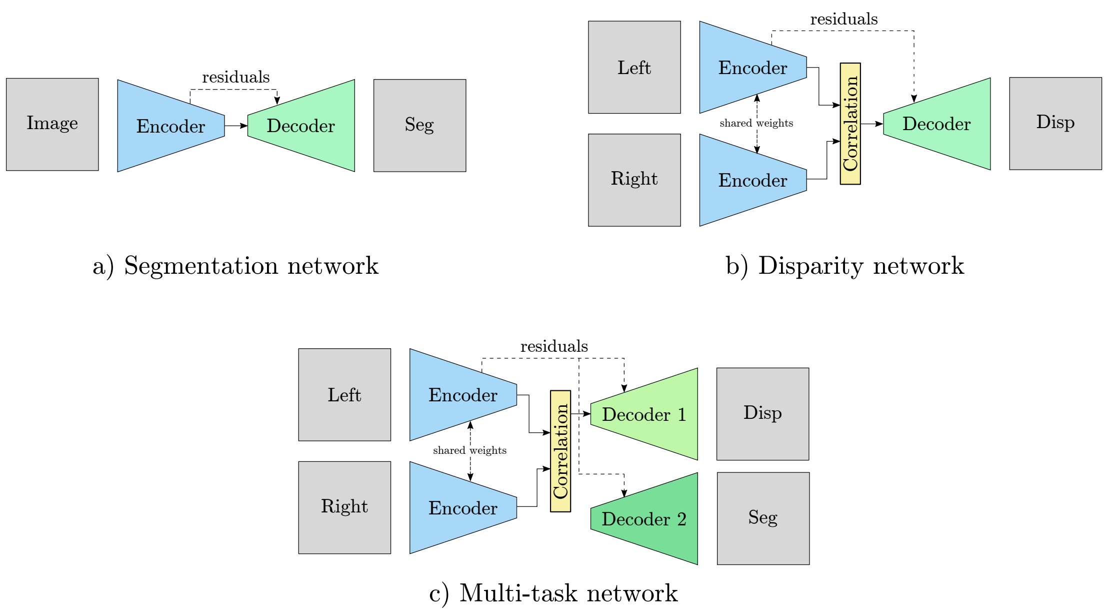


Figure 2.6: Generic semantic segmentation, disparity, and multi-task learning networks. The semantic segmentation network consists of an encoder and a decoder to process the input image and output the segmented probabilities. The disparity network has two encoders, one for each input image. These encoders share weights, so the features from left and right images are processed by the same learned filters. Then, the last encoder features are used to obtain a correlation cost. The multi-task network is similar to the disparity network, but it uses two decoders, one for each task.

multiple scale outputs, freeze the network and use the learned intermediate features to learn the disparity map. As a final step, they refine the network by learning both tasks but optimizing the segmentation of the warped right image instead of the left. Different from the previous approaches, the work by [7] trains an end-to-end network to obtain the disparity and segmentation maps jointly. The network consists of one encoder and two decoders with multi-scale outputs (see Figure 2.7). One decoder segments the input image and the other estimates the disparity. Finally, the intermediate features of both decoders are used to output a refined disparity.

The works above show the most common approaches to disparity estimation and semantic segmentation joint learning. They are robust but computationally expensive, either because they require to train state-of-the-art networks of one or both tasks to be combined later, or because they have sub-networks, one for each task. This leaves space for more efficient methods (or networks) to be researched, which are capable of combining both tasks and reduce the number of extra networks and sub-networks.

In the literature of multi-task learning, it can be seen that a more explored area is the estimation of monocular depth maps instead of disparities. It might be due to

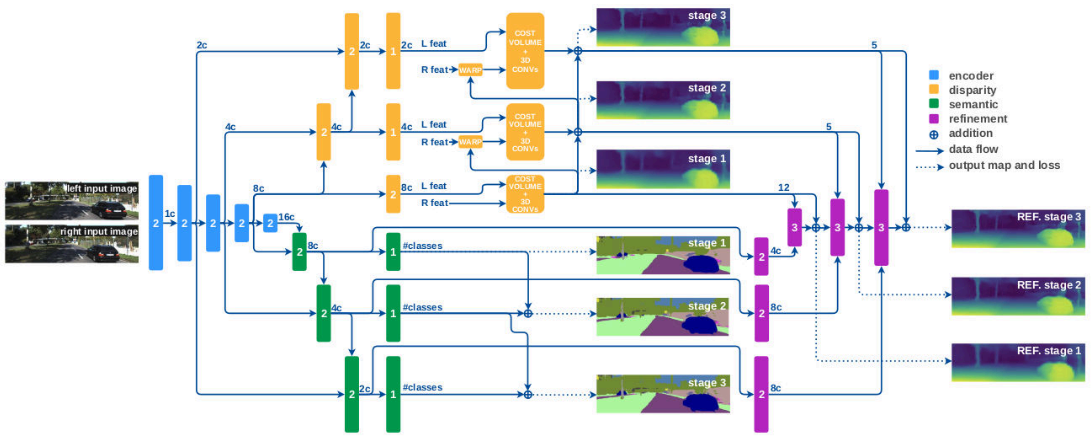


Figure 2.7: Multi-task network proposed by Dovesi extracted from *et al.* [7].

two reasons: 1) Lack of disparity datasets. This is because it is difficult to obtain proper disparity ground truths for real data (images). One way to get "real" disparity ground truths is by creating synthetic datasets. However, rendering images is time-consuming, in particular, when aiming for high-quality images with visual effects such as motion blur or depth of field to simulate real conditions [74]. 2) Depth estimation using monocular inputs removes the necessity of computing cost volumes between features of image pairs. These methods will be briefly reviewed in the following paragraph because they use multi-task learning, and depth and disparity are related. One of the simplest approaches is given by [75]. Their work proposes an architecture that can be used to learn semantic segmentation, depth and normals by changing only the last layer. Ubernet [76] improves this by proposing an asynchronous backpropagation, which allows the network to update only the weights of chosen tasks during the training. As a result, their network can learn seven tasks even when the ground truth is not available. The work by [77] proposes an encoder-decoder (backbone) where the last layer is separated into task-specific branches, one for segmentation and the other for depth estimation. [9] improves the previous method by using DeepLabV3+ [41] with Atrous Spatial Pyramid Pooling (ASPP) as backbone. Instead of outputting both predictions at the end of the network, Zhang *et al.* [78] work predicts the depth and segmentation after each decoder layer, similar to Dovesi *et al.* [7]. These approaches are generally lighter than the ones that use stereo-pair inputs because they do not need to process two input images at the same time. However, monocular depth estimation is an ill-posed problem [79] unless more context is given to the image.

2.3 3D semantic segmentation and classification

Previous sections describe how to find what an object is (semantic segmentation) and where they are located (disparity estimation with calibrated camera) in 2D images. However, the agent (robot) can also receive the information from its surroundings using other types of sensors, which means that the input data would be different. A common type of data is a point cloud. A point cloud is a set of points in 3D, usually unordered and sparse. Given that the point clouds already provide the location of the objects, the goal of scene understanding is to find the semantic classes of each point. This task is called 3D point cloud semantic segmentation, and this section will review the most relevant works in the field.

2.3.1 Early approaches

Similar to early approaches for 2D image segmentation, the first approaches for point cloud segmentation used handcrafted features designed for specific tasks. These features encode statistical or geometric properties of the points and usually are invariant to certain transformations such as translation, rotation, and scale. [11] classifies these features as intrinsic or local [80, 81, 82], and extrinsic or global [83, 84, 85]. The biggest downside of handcrafted features is that they are task-specific and is not trivial to find the optimal feature combination.

2.3.2 Volumetric-based methods

These methods quantize an unordered point cloud in a uniform structure like voxels (see Figure 2.8). Some approaches use 3D convolutions to find local relationships between closer groups of points [86, 87]. However, the amount of memory required to compute these convolutions makes them unfeasible to process a large number of points. Methods like OctNet [88] and O-CNN [89] save computation time by using octrees to avoid processing empty spaces. [90] and [91] use Kd-tree and Hash structures instead. Although these implementations reduce the computation required to train a 3D CNN, quantizing the points comes with the cost of losing important fine-grained information.

2.3.3 Point-based networks

These are networks capable of using irregular point clouds without projecting or quantizing them into regular grids. The main characteristic of this type of network is the

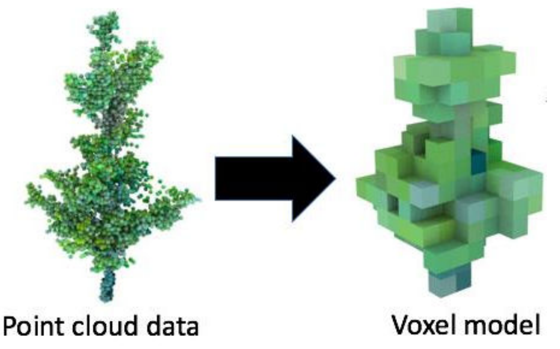


Figure 2.8: Point cloud to voxels example from [92].

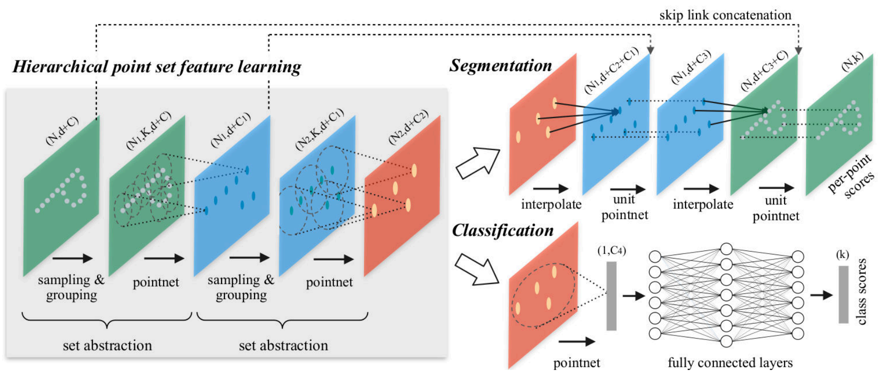


Figure 2.9: PointNet++ neighborhood clustering and sampling process from [93].

use of shared MLP layers, also known as point-wise or 1D convolutional layers. PointNet [11] is a milestone of this kind of network. This approach uses MLP layers as permutation-invariant functions to process each point of a point cloud individually, and a max-pooling layer to aggregate them. The performance of the network is limited because they do not consider local spatial relationships in the data. PointNet++ [93] addresses this issue by sampling the points, grouping them in clusters, and applying PointNet on the clusters; an example can be found in Figure 2.9. SO-Net [94] uses a similar hierarchical structure adding self-organizing maps (SOMs) to capture better local structures. Other approaches like PointConv [95], PointCNN [96] and KPConv [10] construct kernels based on the input coordinates to be used as convolution weights. Some works project local neighborhoods into tangent planes and process them with 2D convolutions. The tangent plane parameters can be found using point tangent estimation [97], or approximated [98, 99, 100]. The downside of these approaches is that they lose the information of 1 dimension given that they project the points to a local 2D plane.

2.3.4 Self-attention and transformers

Self-attention and transformers have revolutionized the area of NLP [101, 102]. This has lead to the 3D segmentation field to investigate these techniques [103, 104, 105]. In the point cloud domain, self-attention networks can be seen as an improvement of the MLP networks, where instead of using pooling layers to capture local relationships, they learn these relationships through an attention layer which estimates a score function to weight the contribution of each neighbor. The self-attention architecture resembles the encoder part of the transformers.

One of the attempts to apply transformers to point clouds is PCT [103]. They replace the MLP layers from PointNet++ with transformer layers and the output feature of each layer is enriched with a discrete Laplacian operator. There are two differences between their method and ours. The first one is that they aggregate the points inside a neighborhood by applying maxpooling. We will use a self-attention instead, which allows the information of all the neighbors to be passed, rather than only using the neighbor with the highest feature value. The second difference is that they use global attention and we will use local attention. This means that their method attends to all points, while ours attends to a local cluster of points. In terms of computation, their memory requirements are quadratic $[N \times N]$ while ours is $[K \times N]$ where N is the number of points and K is the number of neighbors in the local cluster. [104] uses geometric features to add extra information to semantic features, and self-attention and maxpooling are used to do the local aggregation. The same operation is performed at different scales and the results are combined using another self-attention layer. The way they perform local aggregation is the closest to our work presented in Chapter 6. However, this design loses information from the geometric features by combining them indiscriminately with the semantic features. Our network approaches this problem by using a two-head self-attention mechanism, where one head focuses on the geometric features and the other on the semantic features. In addition, they use scalar attention while we use vector attention, the latter brings more flexibility to the attention layer. This is because scalar attention uses the same learned score for all the feature channels of a neighbor point, whereas vector attention obtains a score for each channel individually [105]. Our method can also be considered as the closest implementation to a transformer for point cloud segmentation, because, unlike previous works, we adapted one of the key properties of transformers, which is the multi-head attention structure.

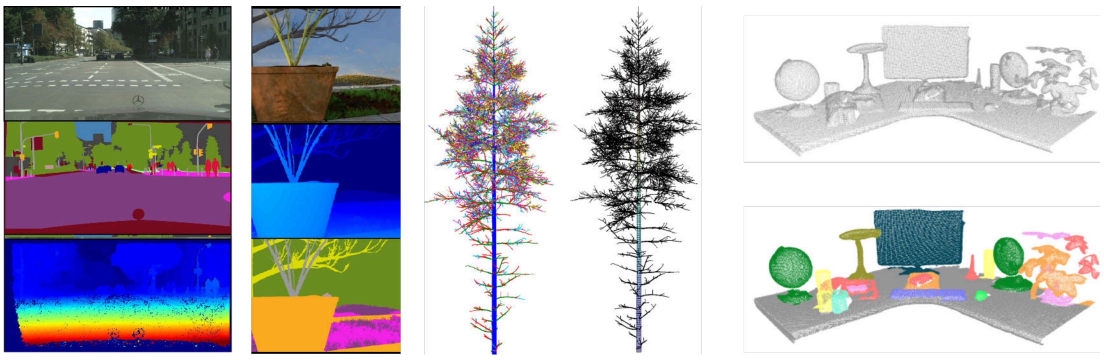


Figure 2.10: Scene understanding applications using 2D and 3D data. From left to right. The first two columns show examples applied to urban and rural areas to segment and localize objects. The last two columns show examples of 3D point cloud segmentation. The third column is part segmentation (segment branches in plants) taken from [106] and the fourth is object segmentation taken from [107].

2.4 Scene understanding in the real world

After exploring the history and background of the main parts of scene understanding, this section gives an overview of the areas where it can be applied, focusing more on the applications in agriculture, where the objects in the environment have similar characteristics, making it an interesting and challenging problem. Some examples can be found in Figure 2.10. Finally, as some of our methods are applied to visual servoing (in agricultural robotics), a brief overview about this area and their challenges will be given.

2.4.1 Urban areas

This thesis considers data captured from urban street and indoor areas. The most common application for urban areas is street scene understanding for autonomous driving. The approaches vary from segmenting the whole data [41, 10] (2D images or point cloud) to only one part of it, like roads [108], cracks on the road [109], people [110], and cars [111]. Indoor scene understanding is usually used for robot navigation, obstacle avoidance [112, 113], grasping [114], and visual servoing [13]. As part of the perception module, it is in charge of finding the objects that the robot will interact with. Therefore, if the robot needs to operate in real-time and in a real environment, this module can be a bottleneck for the system.

2.4.2 Agriculture

Similar to the applications in urban areas, scene understanding is often used in agriculture to find the location of the objects that surrounds an agent (robot). It is used to navigate around gardens and farms [115], or to do more specialized work like fruit grasping [116], crop picking [117], plant segmentation [118], branch cutting [13], plant phenotyping [106]. The latter is a growing area where scene understanding is used to recover the physical form and external structure of plants, which allows the robot to interact with them.

2.4.3 Visual servoing in agricultural robotics

Visual servoing methods, iteratively and in real-time, control robots using visual information as input data. To acquire information around the robot, cameras can be placed on the manipulator (eye-in-hand) or in the environment (eye-to-hand). These terms have been defined as: *"the camera is said eye-in-hand (EinH) when rigidly mounted on the robot end-effector and it is said eye-to-hand (EtoH) when it observes the robot within its work space"* [119].

The benefit of the EinH configuration is that the robot can have a closer and more precise interaction with the target. However, it is limited to the working space of the arm. Therefore, the information that can be received is constrained to the field of view of the camera [120] and its current position. On the other hand, EtoH methods use a camera located outside the work-cell, which allows them to have a broader, but less detailed, view of the working space. This helps to overcome the problem of partial view that eye-in-hand cameras have.

In agricultural robotics, more specifically in the tasks of grasping and harvesting, EinH configuration is preferred for visual servoing because the vision module can capture more detailed information of the target when the robot gets closer to it. EtoH is generally used to model the plant in 3D or extract richer information of the object. Then, the robot plans the trajectory based on this information. However, there is no visual feedback [121, 13, 122]. Thus, the rest of the section will describe previous EinH approaches.

Gurel *et al.* in [12] and [14] present a simple approach to segment rose stems by thresholding the green color space. They use this segmentation to find the skeleton of the stem and trace it with a manipulator and a stereo camera. This process is done using a single rose stem with a constant background. On the other hand, [123] trims bushes in real gardens using 3D shape fitting and an eye-in-hand stereo camera. Based on the shape, they calculate the optimal positions and trajectory for the robot to trim the bush to the desired shape. After each trim, the robot evaluates the quality of the cut to repeat it or continue with the planned trajectory. In [124], they harvest tomatoes using an image-based visual servoing approach. They find the position of the fruit based on their round shape and characteristic color. Similar to the previous method, Barth *et al.* [125] uses a CNN to segment the fruit of sweet pepper plants and SLAM to locate their position in 3D. However, unlike [124], they use position-based visual servoing to find the velocities of the robot. Similar to the previous approach, [126] segments the fruit to find its position in the image but uses a stereo camera to obtain the 3D location instead.

While exploring the literature, many applications of visual servoing on fruit harvesting were found [127, 128, 129, 130], but none for rose pruning. This might be caused by the complexity of the task. In fruit harvesting, the color of the fruit highlights its position. However, to prune a rose, the robot has to interact with branches, whose shape and structure are similar along with the whole bush, making it a more complicated task. The closest approach to rose pruning is the work developed by Botterill *et al.* [13] where they create a system to prune branches of grapevines. However, they do not use visual feedback to help with navigation. First, they capture the plant from different perspectives, segment the branches, and then reconstruct the plant in 3D. Then, a brute force searching algorithm is used to find the branches that should be pruned. Once the branches are found, their position is sent to the robot to cut them.

2.5 Discussion

Scene understanding can be divided into two sub-tasks, semantic segmentation to detect the objects in the scene, and disparity (or depth) estimation, to locate them. Individually, each area has an extensive literature with good performance using 2D images. However, their state-of-the-art CNNs are prohibitively expensive in terms of processing time and memory, which makes them unsuitable for their application in real-time scenarios. One solution is to use a classical method to compute the disparity, which usually provides good results in real-time, and use deep learning to segment the images, which is a more difficult task when the environment is not controlled. This strategy is used in Chapter 3. Chapter 4 validates the previous method by applying it as the perception module of a robot arm to do real-time visual servoing. These two first approaches are tested in the agricultural robotics field. More specifically, to segment rose bushes, obtain their morphology and find their 3D position in the environment, as well as key locations based on some constraint rules (gardening rules). This area is chosen because robots in agriculture need light online methods for navigation. Also, agriculture itself is an area that benefits from robust vision methods to extract information about plants to know their location [115], collect fruit [131], or get their morphology [132]. To obtain this information from such complex objects, it is necessary to have a large and diverse dataset, especially in the age of deep learning. Thus, we created a large dataset of real and synthetic roses which capture diversity, realism and have different types of labeling.

Unlike Chapters 3 and 4 where deep learning and classical methods are combined in a pipeline, Chapter 5 explores the area of multitask learning to learn jointly the semantic segmentation and disparity maps using a single CNN. Apart from cameras, robots can use other types of sensors to gather data. These sensors can capture 3D information like point clouds. Here, the goal of scene understanding is to label each point with a class. Chapter 6 uses geometric and latent information from 3D point clouds and a neural network to tackle this problem.

Scene understanding is more than just segmenting an image. It allows the agent to detect and localize the objects in the scene and interact with them. This interaction can be as simple as finding what are the objects around the agent or as difficult as recovering specific information of an object so the agent can adapt their behavior when interacting with it, such as grasping different surfaces and obtaining quantitative information from an object.

Chapter 3

2D Segmentation and Disparity Refinement

This chapter explores the area of 2D scene understanding applied to morphology estimation and 3D reconstruction using the classical pipeline approach of image acquisition, segmentation, disparity estimation and refinement. Because the segmentation in real environments is a hard task, it is done using a convolutional neural network. To test our method, segmentation and disparity outputs are used to reconstruct and find the branching structure of plants. The research reported in this chapter was previously published in [1].

3.1 Introduction

Computer vision and robotics have made significant advances in detection and automation of indoor and outdoor tasks. The vision part is generally used to segment, localize or track objects so the robot can navigate to an area of interest and manipulate the object [133]. Outdoor tasks usually deal with a more uncontrolled environment than indoor tasks, mainly due to the variations of light, wind, shadows, as well as variations in the type of terrain. An example of a challenging outdoor task is that of a robot that can move through a garden, detect key elements and recover their structure in order to work with them.

The method proposed in this Chapter belongs to the vision module of a garden robot capable of navigating towards rose bushes and clip them according to a set of pruning rules[1]. An example of the pruning rules can be seen in Figure 3.1. This robot

[1]Project TrimBot2020: `http://trimbot2020.webhosting.rug.nl`

(see Figure 3.2) consists of a mobile platform with a camera rig for navigation and a 6-DOF robotic arm with a cutting tool and a stereo camera mounted on the end-effector using an eye-in-hand configuration [134]. A detailed description of the robot can be found in [135]. A demonstration of the rose pruning process can be also seen in the video: `https://youtu.be/r9IHy5lH8YM`.

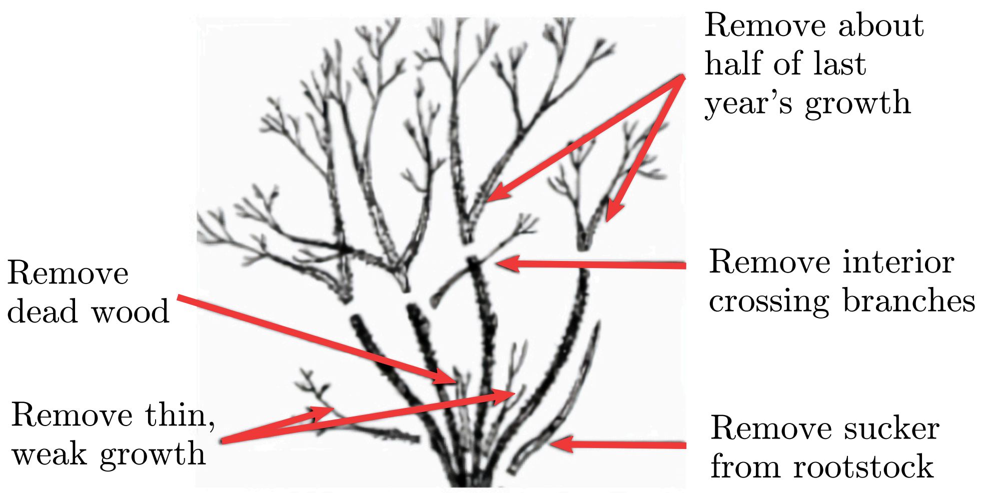

Remove about half of last year's growth
Remove interior crossing branches
Remove dead wood
Remove thin, weak growth
Remove sucker from rootstock

Figure 3.1: The image[2]shows the rules a human should follow to prune a rose bush. In the case of the robot, it must recognize parts of the plant where these rules should be applied. Our method obtains the morphology of the rose bush. In other words, it finds the branches of the bush as well as their radius. These outputs are essential for the perception module, as they can be used to find parts in the plant that are needed to do rose pruning, such as dead branches, crossing branches, thin growths, and the height of the plant.

The vision module is divided in two main parts, one for robot navigation and other for visual servoing. The robot navigation uses a depth fusion system which combines multiple disparity images obtained from a 10 camera rig [136] and SLAM for robot localization [137]. This part allows the robot to navigate in the garden towards a rose bush. The second part of the vision module implements visual servoing for the manipulator. This is in charge of processing the images captured by a stereo camera mounted on the robot arm to detect cutting locations on a rose branch and move the cutter towards those cutting locations.

The first module allows the robot to navigate towards a rose bush and, once a rose bush is located and the robot is facing towards it, the system switches to the second module. This second module assumes it will receive images where only one rose bush

[2]Image modified from `https://www.allaboutannapolis.com/preparing-your-yard-for-winter.html`

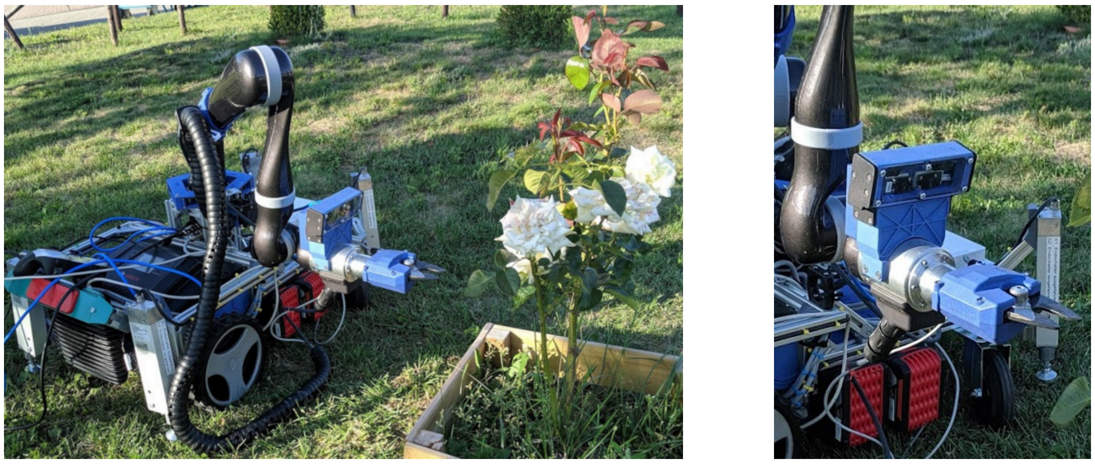


Figure 3.2: General overview of the robot (left image) including the 6-DOF robotic arm and the cutting tool. A stereo camera using an eye-in-hand configuration (right image) is mounted on the same cutting tool.

appears and all the segmented branches belong to the same plant.

The pruning process is: First, the branches are localized by segmenting them from the rest of the scene. Then, the 3D morphology of the plant is recovered. Finally, based on the morphology of the plant, the robot finds the branches that should be cut and sends the manipulator to clip them. This Chapter focuses on the vision module, which segments the stems of a rose bush and obtains its morphology. To the best of our knowledge, there are no previous methods devoted to segment roses and obtain their branching structure for pruning. The closest approaches are the segmentation of trees and plants, which are described in detail in the following section.

Our approach tries to solve this problem without making assumptions about the type of environment or lighting conditions, working with rectified stereo input images and recovering the morphology of the plant to determine the branches to prune. First, a Selectional Autoencoder architecture [138] is trained to select the pixels that belong to the branches of a rose bush. It then calculates the disparity map and combines it with segmentation to improve the accuracy of the result. Finally, the skeleton of the bush, the branches and the 3D morphology of the plant are obtained.

Also, we have seen that previous methods use simple datasets with constant backgrounds and few images to segment the plants from the rest of the scene. This is mostly because pixel-level labeling of stems and branches is time-consuming and requires more precision than other objects, like cars or people, due to their thin structure. The reduced number of datasets and images might be also one of the causes why plant segmentation is a less explored area compared to other semantic segmentation tasks. In addition, the availability of a large dataset has become even more important in the era of convolutional networks. For these reasons, this work presents a collection of three such datasets with real and synthetic plants in different environments, light conditions, and with intra-class variation, as well as their segmentation ground truth, and disparity map for the synthetic plants.

Therefore, our approach is validated using five datasets. Our three large datasets of roses and two datasets of plants. More specifically, we have: (1) A synthetic dataset of rose bushes in 4 different environments, (2) a dataset of realistic plastic rose bushes with real backgrounds, (3) a dataset of real roses captured in two different botanic gardens, (4) a dataset from the state of the art of *Arabidopsis* genus plants captured indoors, and (5) another dataset from the state of the art of real roses captured in a real garden.

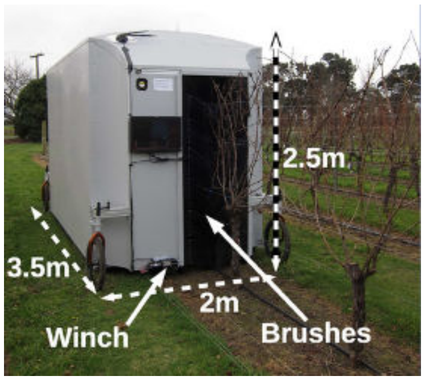
In summary, **the Chapter makes the following contributions**: (1) Rose stem segmentation through Selectional Autoencoders adjusted to work on real environments with variable lighting conditions, (2) novel combination of binary segmentation with disparity to obtain the 3D morphology of the plant through skeletonization, (3) a complete pipeline to recover the morphology of rose stems from stereo images that allows to determine the 3D structure of branches and choose the ones that should be pruned, and (4) a collection of datasets for rose bush segmentation. The rest of the Chapter is organized as follows: the next section makes a brief review of the state of the art, Section 3.3 presents the proposed approach, Section 3.4 describes the datasets used in the evaluation, Section 3.5 reports the evaluation results, and finally, conclusions and future work are addressed in Section 3.6.

3.2 Related work

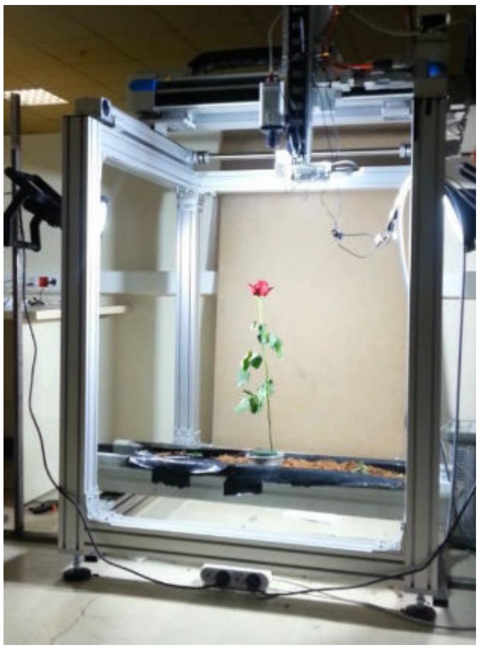
As stated before, there is a limited literature on the specific task of rose bush segmentation and morphology extraction using 2D stereo images. Therefore, we will focus on works that are closely related, mainly in the areas of tree and plant modeling, either using 2D images, multiple 2D images or point clouds. There is an extensive literature on these topics, so we will review it by grouping them according to the type of data used.

Among the methods that use point clouds we can find works that only reconstruct the branches (without leaves, called off-leaf) and others that do include them (on-leaf). For example, [106] model off-leaves trees by fitting cylinders to the point cloud. Then, they find the tree components by considering small regions of the tree and searching for neighbors and bifurcations. Hackenberg *et al.* [139] propose a similar approach but using spheres to look for sections of the branch instead of splitting the tree in regions. These two approaches obtain the plant information using point clouds of off-leaf trees. However, they need a clean point cloud as input while our approach uses a raw stereo image as a starting point. There are also works on getting the morphology of trees with leaves. Livny *et al.* [140] propose an on-leaf model to reconstruct the skeletal structures of trees from a point cloud. They perform global optimization and clustering on a directed acyclic graph that represents the points of a tree. Belton *et al.* [141] obtain geometric features and use Gaussian Mixture Models (GMM) to split the tree into its principal parts (trunk, branches and leaves). Huang *et al.* [142] get the skeleton of a plant by applying *L1 - medial skeleton* on a point clouds. While this method works well to obtain the 3D skeleton of any 3D shape, it does not capture any morphological information of the plant such as branching structure and radius of the branches like our method. Tabb *et al.* [143] go a step further and not only compute the 3D skeleton of a tree, but also obtain its graph structure and radius of the branches. However (and unlike us), they segment the tree using a clean point cloud with a constant background. In general, all these methods work only with the point clouds of isolated trees which were previously pre-processed or cleaned.

One can also find some works that use 2D images to recover the morphology of plants and trees. For example, Zheng *et al.* [144] segment plants using mean-shift, color features and a shallow neural network to judge whether a cluster belongs to the plant or not. In Zheng *et al.* [145], they also use mean-shift to segment plants in a field but they employ a Fisher linear discriminant (FLD) instead of a neural network to perform the segmentation. However, none of these methods is evaluated with plants in a real garden. They only focus on green vegetation planted in soil. So they would not be applicable in our case, where it is necessary to deal with a variety of rose bushes, with different branch colors, with or without leaves, and in multiple scenarios. Barth *et al.* [146] segment the stems of sweet pepper plants along with other 6 classes (buds, leaf stems, cuts, etc.) using a Fully Convolutional Network (FCN) with a Conditional Random Field (CRF) layer. They also generate synthetic images [147] to train the FCN and fine-tune it with a small dataset of real images. The stem segmentation had a low true positive rate because of the similarity among the different classes considered. In addition, the synthetic dataset only includes one environment and focuses on recreating a specific garden. In our case, five different datasets are evaluated, also including a synthetic datasets but with multiple rose bushes viewed from different perspectives, scenarios, and lighting conditions. Gurel *et al.* [12, 14] segment rose stems to find their center lines and trace them with a manipulator using a stereo camera. This process is done using a single rose stem under a controlled environment where the background is different from the stems (see Figure 3.3). Botterill *et al.* [13] obtain the 3D model of grapevines using 3 monocular cameras. Their approach takes the color information of the cameras, segments them and finds their correspondences to reconstruct the 3D plant. Finally, they move the cameras parallel to the vines to add more information to the 3D reconstruction. The method is robust in constrained environments with constant light conditions (Figure 3.3).



a) Controlled environment



b) Constant background

Figure 3.3: The figure shows examples of how current methods constrain the surroundings of plants to better segment and locate the stem. The images were taken from [13] and [14] respectively.

Another common approach to recover the morphology of a plant is to reconstruct it in 3D using multiple 2D images captured from different perspectives. For example, Isokane *et al.* [148] segment the plant from 2D views and get the probability of the existence of a branch by using a variation of a Pix2Pix GAN [149], then they reconstruct the plant in 3D combining multiple views. The algorithm works well with synthetic data, however, the method does not generalize well for real data. Another work that combines 2D and 3D images is [150], which captures 64 views of a plant in

2D and obtain the mesh for each view. Unlike the previous methods, they segment the meshes of the plant rather than segmenting the 2D image. For this, they apply a region growing algorithm and fit geometric primitives to the segmented meshes. Simek *et al.* [151] proposed an approach based on temporal information and Gaussian processes to model the branches from multiple 2D images. Similarly, Gelard *et al.* [152] build the 3D model of a plant using structure from motion. Then, they segment the plant by fitting a cylinder from the base of the stem until it reaches the top. After segmenting the stem, they remove it from the point cloud and process the leaves by clustering the remaining points. Santos *et al.* [132] uses structure from motion to obtain the point cloud from multiple 2D images. They segment each part of the plant by using spectral clustering on the 3D points. Alenya *et al.* [153] combines color and depth images to segment the leaves of a plant. They perform a rough segmentation of the leaves using a graph-based method [25], then they choose the best leaf segments by fitting quadratic surface models to the segmented depth images and evaluating which segments best fit the depth data. These segments are then post-processed using a nearest-neighbor graph to reduce over segmentation. All these methods extract the point cloud by capturing multiple images of the plant from different perspectives. Also, they operate in the 3D space to separate the branches from the rest of the plant. In our case, we use 2D information combined with a calibrated stereo camera to segment and obtain the 3D morphology of the plant.

In summary, while all these methods segment the plants and obtain their morphology, they usually make strong assumptions. The methods using point clouds usually require the data to be accurate, complete and without noise. This requires much preprocessing work which most of the times is done manually. In the case of works with 2D data, a controlled environment with a fixed background is mainly considered for each perspective of the plant. Among these methods, some of them reconstruct the plant in 3D, however, they have to use multiple views to recover it properly. Also, the evaluation of these model based methods usually assumes an error margin. Therefore, if the reconstruction is inside the error threshold, the metric will still report a small error even if the reconstruction is few centimeters off.

Another important disadvantage of these methods is the assumption of having a controlled environment to perform the extraction of the plant and most do not evaluate using real datasets, with different types of plants and environments. In addition, most of them use images of indoor plants with homogeneous background and/or constant light conditions [13, 14, 12]. On the other hand, the datasets that do contain a real garden usually cover the background of the plant with a constant color [147, 146]. We highlight the limitations of using these methods in a real garden because their main applications will require the robot visual system to work in an uncontrolled environment. After exploring the different datasets that previous garden robotics methods use, we found that most of them, apart from the disadvantages mentioned above, are small and lack variability, meaning that they only have a small quantity of images of a single type of plant or tree, making it difficult to evaluate how well those methods generalize compared to other datasets.

Our approach tries to overcome all these disadvantages by using a method without making assumptions about the type of environment or lighting conditions. Moreover, it is validated using five different datasets, with more than 6500 images of indoor and outdoor scenes, including real garden images in different environments, with variable lighting conditions, and without homogeneous backgrounds.

3.3 Methodology

The proposed approach to reconstructing a rose bush from stereo images is divided into several steps (see Figure 3.4): First, the left image is segmented using a Fully Convolutional Segmentation Network (FCSN). In parallel, the stereo pair is supplied to a disparity method to calculate the depth of the plant. These results are post-processed to combine the segmentation and the disparity, and to calculate the branches that make up the plant. Finally, the 3D reconstruction is obtained using the branches and depth. These steps will be explained in detail in the following sections.

3.3.1 Fully convolutional segmentation network (FCSN)

The segmentation process is the most important step in the entire proposed algorithm, since the accuracy of the final result will depend on it. Basically, this process performs a classification of the image pixels into two possible categories, indicating if the pixel belongs to the class branch or to the class not-branch, which we denominate as background. In this case, the background not only includes the garden background, but also other parts of the bush itself, such as leaves or flowers, and even other elements that can surround it and that could be similar to a branch, such as sticks of a fence, support stakes, etc. Therefore, this step eliminates everything that is not a branch and thus avoids sending incorrect coordinates to the robot.

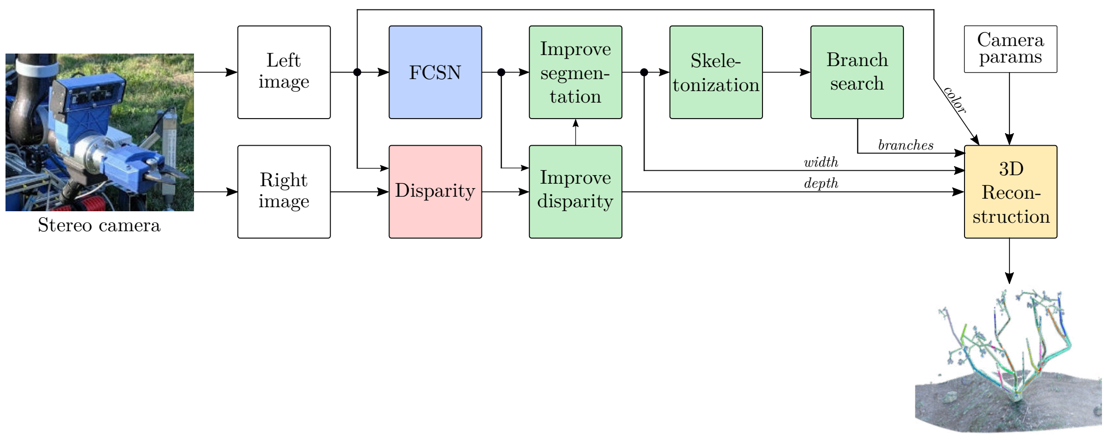


Figure 3.4: Scheme of the pipeline followed by the proposed method. First, the segmentation and disparity images are calculated. Post-processing is then carried out to combine these results and recover the morphology of the plant. Finally, the 3D reconstruction is calculated using the previous results.

It should be considered that branches are very thin objects with colors that can be very similar to the background. In addition, they can also be confused with other surrounding elements (such as a stick or a support stake), especially in this context, in which the images are not captured in a controlled environment with constant background and lighting. All this makes this task considerably more complex than a general segmentation task, in which larger and more prominent objects in the image are usually considered.

To perform the branch segmentation, we use a Fully Convolutional Segmentation Network (FCSN), based on the work of Long *et al.* [154], but instead of having a categorical vector to indicate the class of each pixel (as other networks of this type also make, such as SegNet [39]), we modify the last layer to return a matrix representing a probability map of the presence of a branch in each pixel of the input image. In other words, the proposed FCSN is trained to perform a function such that $s : \mathbb{R}^{(w \times h)} \rightarrow [0,1]^{(w \times h)}$, learning a map over a $w \times h$ input image that preserves the input shape and indicates the probability that each pixel belonging to the branch class.

As in the architecture proposed by Long *et al.* [154], the layer hierarchy of our FCSN follows the idea of auto-encoders, where first a series of convolutional layers combined with pooling layers are added to reduce the size, until an intermediate layer in which a meaningful representation of the input is attained. As these layers are applied, filters are able to relate parts of the image that were initially far apart. This first part of the network would be equivalent to the encoding stage of the auto-encoder (see Figure 3.5). Then, it follows a series of convolutional plus upsampling layers that reconstruct the image up to the same input size (this second part would be the equivalent of decoding stage). The last layer consists of a set of neurons with sigmoid activation that predict a value in the range of $[0,1]$, depending on the *selectional* threshold δ for the corresponding input feature. The δ parameter is a hyperparameter that is also learned during the training phase.

In addition, a series of modifications were made to the network architecture and the layers used. The downsampling in the network encoder part is performed by convolutions using stride, instead of resorting to pooling layers. Up-sampling is achieved through transposed convolution layers [155], which perform the inverse operation to a convolution, to increase rather than decrease the resolution of the output. Residual connections were also added to improve the accuracy of the reconstruction. For this, the encoder layers are connected with the corresponding decoder layers, similar to how it is done in U-Net [156]. But unlike the latter, instead of concatenating the feature maps we add them as is done in other architectures, such as in ResNet [157]. In this way, we can both help the training process and improve the accuracy of the reconstruction.

Feature maps are zero padded so that the dimension before and after the convolution remains the same and can be used for the skipped connections. Batch normalization [158] is performed after convolution to compensate for the covariance shifts and prevent overfitting during the training procedure. Dropout [159] layers with a probability of 0.5 were also added after each normalization layer to improve the generalization capabilities of the network. Finally, ReLU [160] was used as activation function for all layers except for the output layer, for which the sigmoid activation function is used as explained above.

To find the best network configuration for this particular problem, we applied a *grid-search* technique [161], analyzing different values of hyperparameters, including the number of layers of the network, the input size, the number of filters of each convolution, the kernel size, the normalization and the equalization types, the data augmentation factor, the dropout value, and the threshold δ value. The results of the hyperparameters exploration are included in Section 3.5.1, although we summarize the best topology found for this network in Table 3.1. Figure 3.5 also shows an outline of this architecture with the content of each layer.

Once the FCSN has been trained, detecting branches from an input image consists of feeding the image through the FCSN, which outputs the branch probability assigned to each input pixel. Those pixels whose selection value exceeds a certain threshold δ are considered to belong to a branch, whereas the others are discarded.

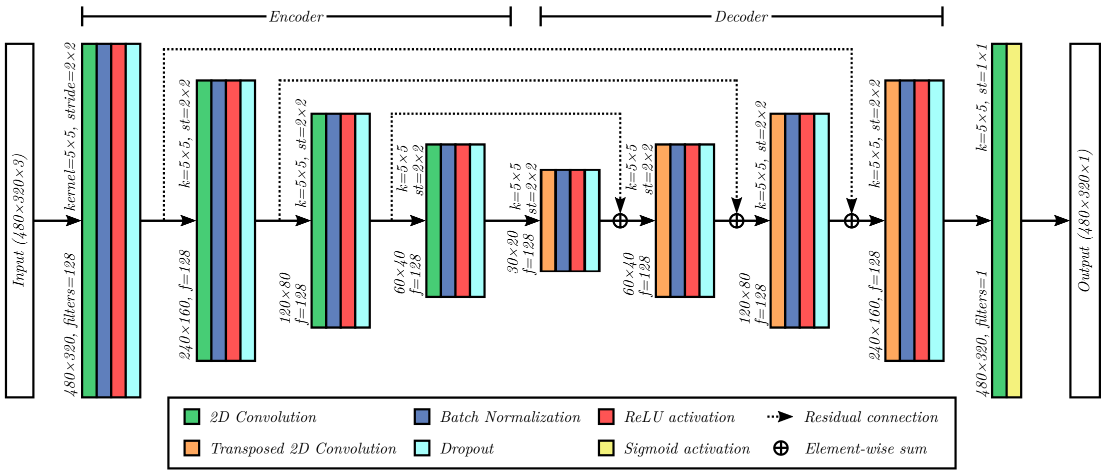


Figure 3.5: Scheme of the Fully Convolutional Segmentation Network. In this figure, the layer type is labeled with colors according to the legend. The size of each layer for convolutions and transposed convolutions is $h \times w$, where h is the height and w the width. The number of filters (f), the kernel size (k) and the stride value (st) applied for each layer are also shown.

Input image size:	480×320 px
Number of encoder/decoder layers:	4+4
Filters per layer:	128
Kernel size:	5×5
Normalization type:	Standard
Equalization type:	HSV
Data augmentation:	10 %
***Selectional* threshold δ:**	0.3

Table 3.1: Best hyperparameters found after the grid-search process for the segmentation network FCSN.

The training stage consisted of providing the FCSN with examples of images and their corresponding segmentation ground-truth, that is, binary maps over the pixels that belong to branches (see Figure 3.10c). The binary *cross-entropy* loss function between each output activation and its expected activation was used to calculate the error. The tuning of the network parameters was performed by means of back-propagation using stochastic gradient descent [162] and considering the adaptive learning rate proposed by Zeiler *et al.* [163]. The training stage lasted a maximum of 300 epochs with a mini-batch size of 8 samples, and *early stopping* when the loss did not decrease during 15 epochs.

In addition, we applied a fine tuning process during the training stage, initializing the network with the weights learned using a synthetic dataset. Data augmentation [164, 165] was also used to artificially increase the size of the training set by randomly applying different types of transformations to the original training samples. This technique usually improves the performance and helps reduce overfitting. In our case, for each image of the training set, 10 augmented images were generated. The transformations applied were randomly selected from the following set of possible transformations: horizontal flips, horizontal and vertical shifts ([-10, 10]% of the image size), zoom ([-10, 10]% of the original image size), and rotations (in the range [-5°, 5°]).

3.3.2 Disparity calculation

The classic Block Matching (BM) algorithm [166] is used to calculate the disparity of the branches. Although this method is not as accurate as others, like Semi-Global Block Matching (SGBM) [167] or DispNet [168], it gets quite competitive results with a much faster runtime. BM obtains the disparity maps in real-time ($\sim$ 30 FPS), whereas SGBM runs at $\sim$ 4 FPS and DispNet at $\sim$ 2 FPS. All the FPS were obtained by running these methods on our synthetic dataset and computing the average frames per second. The resolution of the image was $[720 \times 480]$, and the device used to run the experiments is described in Section 3.5 A fast runtime is necessary to update and keep track of the cutting points and the shape of the bush after each clipping because the branches can be slightly moved by the wind or the manipulator.

The BM method from ROS Kinetic (Robot Operating System) [169] was used to do the stereo matching. Considering that the final prototype of the robot has a stereo camera with a small baseline (0.03m) and that the manipulator's tool-tip is around 0.15m away from the camera [135] (as seen in Figure 3.2), we are only interested in objects that are equal or farther than that distance. Therefore, the parameters of correlation window size and disparity search windows were set to 15 pixels and 64 pixels respectively.

Once the disparity map is obtained, Equation 3.1 is used to convert the disparities d (in pixels) into real depth values z (in metres).

$$z = f\frac{B}{d} \tag{3.1}$$

where f is the focal length of the camera (in pixels) and B the baseline or distance between the two lenses (in metres). In our case, the disparities were obtained using rectified images and the camera parameters. These parameters were found using the calibration software Kalibr [170].

In addition, thanks to the post-processing step that combines the segmentation and the disparity (which will be presented in the next section), we can improve the accuracy of the disparity calculated by this algorithm and obtain a dense disparity for the segmented branches.

3.3.3 Combine disparity and segmentation

Although the result obtained by the segmentation network is good, sometimes it cannot segment a whole branch completely but it splits the branch into small regions, as observed in Figure 3.6(b) inside the red box. These small regions are mostly caused by thin branches or complex areas where it is difficult to separate the background from the foreground. We propose that the segments that belong to the same branch can be joined using the disparity information. This is based on observing the output of the BM algorithm, which is able to find the disparities of a whole branch, as Figure 3.6(c) shows. In addition, the BM algorithm creates "blobs" with similar disparities (usually with a size slightly larger than the branch), which allows us to use the disparity image to complete regions where the segmentation is not continuous.

To join the segmented regions, first, each "blob" of the disparity map is evaluated to find if it contains any segmented branch within its boundaries. In the case that a "blob" has two or more segmented branches, these segments are considered part of the same branch and joined if there is at least one linear connection between the pixels of the segments, as seen in Figure 3.6(d). Using this criteria, only the segmented regions that have similar disparities and are close to one another are joined.

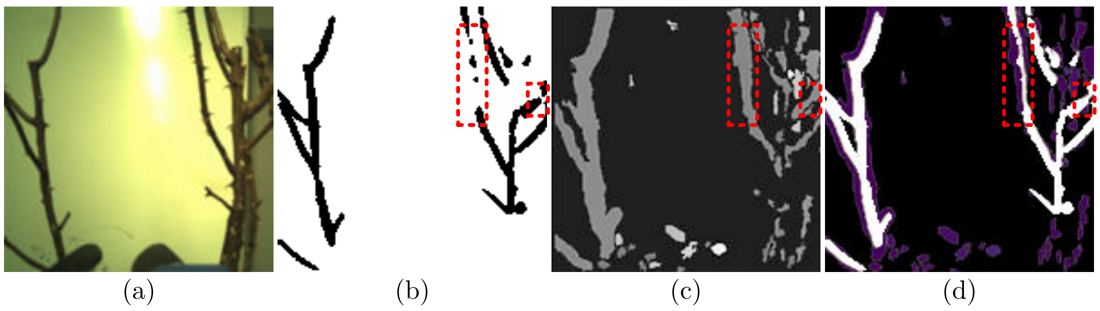


Figure 3.6: Process of the segmentation completion using the disparity map, where (a) shows the input image, (b) the segmentation obtained by FCSN, (c) the disparity output of the BM algorithm, and (d) how the regions of the branch that were not segmented completely (red rectangles) were joined. To facilitate the visualization of this process, the δ threshold was modified to generate visible errors produced by the segmentation.

On the other hand, classical stereo correspondence methods, like BM, cannot match certain parts of the image because of textureless regions or repeated patterns, and leaves them without any disparity value. This causes some regions, like parts of the branches, to not have depth information. We propose to alleviate this problem by using the segmented image obtained in the previous step as prior information for disparity completion.

The proposed method is the following. First, the segmentation mask is multiplied by the disparity map. It removes all the pixels that were not labeled as branches. As the second step, our method finds all pixels $\hat{p}_i$ that are segmented as branches and do not have any disparity assigned to them. Then, the disparity for each $\hat{p}_i$ is found by interpolating the disparities of their closest neighbors using inverse distance weighting. A better view of the method can be seen in Figure 3.7. In the case that the missing pixel does not have a neighbor with disparity, it is discarded. An example of the result obtained by this process can be seen in Figure 3.16.

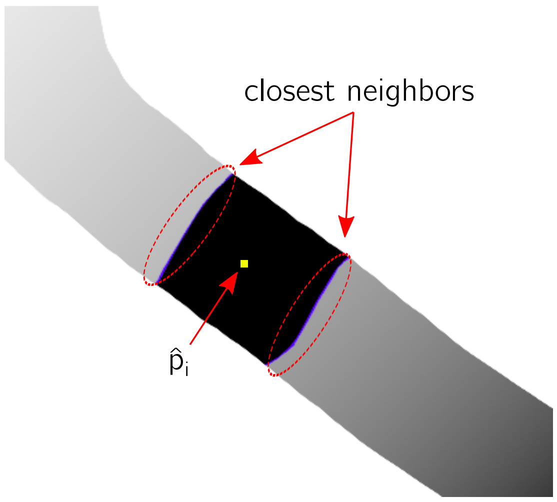


Figure 3.7: Disparity interpolation. Our method obtains the disparity of the point $\hat{p}_i$ by interpolating the known disparities of its closest neighbors (points in blue) using inverse distance weighting.

3.3.4 Skeletonization

As next step, the improved segmented image is processed further to get the morphology of the plant. For this, a 2D skeleton is extracted from the binary image.

To find the skeletonization method that suits better to our task, we evaluated five methods: Zhang *et al.* [171], Parallel thinning [172], 3D skeletonization [173], Medial axis [174], and RUSTICO [175] (see Section 3.5.3 for algorithm details and evaluation results). These methods succeed in finding the skeleton of the plant. However, most of them generated small branches that do not represent a real branch but noise or a "small bump" on the edges of the segmented plant. Finally, the method that obtained the best results was Zhang *et al.* [171], as seen in Figure 3.17, both in the accuracy of the calculated skeleton, in the number of small branches generated, and in the runtime. Therefore, we selected this method for the final implementation of the algorithm.

3.3.5 Branching search algorithm

Once the 2D skeleton is obtained, the branches of the plant are found by exploiting the basic principle of thinning: *A thinning algorithm reduces the components of a binary image to single pixel thickness lines*. This means that, if a pixel belongs to a branch, it should have at most two neighbors, in case it has more than two neighbors, it should be considered as a branching node, as seen in Figure 3.8.

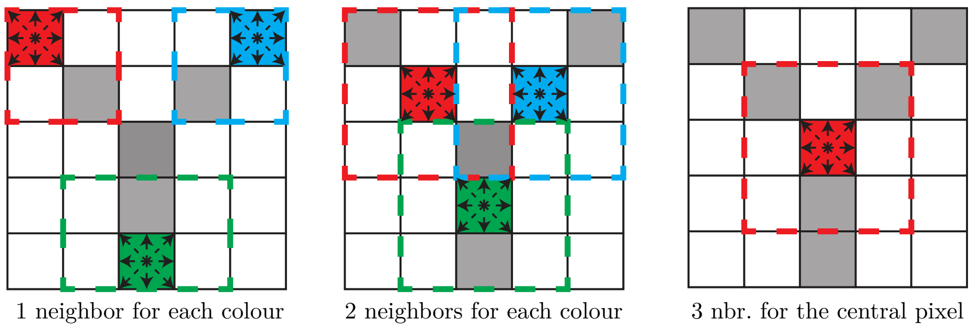

1 neighbor for each colour
2 neighbors for each colour
3 nbr. for the central pixel

Figure 3.8: Neighborhood evaluation criteria. In the first and second images, the pixels marked in red, blue, and green are considered as branch pixels because they only have 2 neighbors at most. In the third image, the red pixel is considered as a branching node because it has 3 neighbors.

The proposed algorithm solves the branching as follows. First, it picks a random pixel from the skeleton of the plant. If the pixel has more than 2 neighbors in an 8 neighborhood criteria, it is classified as a node. If the pixel has 2 or less neighbors, it is classified as a branch. To find the rest of the pixels in that branch, it explores the neighbors of the pixel in a recursive way. This means that, if we classify a pixel as a branch, we will evaluate its neighbors until the new neighbors are either classified as a node or they no longer have more neighbors. In addition, to correct possible errors of the skeletonization process, branches with lengths less than or equal to 3 pixels are eliminated.

Algorithm 1 shows the formalization of this process using pseudocode, where the function "*neighbors*" returns the set of neighbors of a pixel, the function "$\mathcal{U}$" returns the set of unvisited pixels, "*find_branch*" is a recursive function that calculates the pixels that make up a branch from an initial pixel, and the sets $\mathcal{N}$ and $\mathcal{B}$ contain the lists of branching nodes and branches found, respectively. In the case of set $\mathcal{B}$, the algorithm creates a sub-list of pixels for each branch.

```
Algorithm 1: Branching search
  $\mathcal{S} \leftarrow$ Rose bush skeleton image
  $\mathcal{N} \leftarrow \{\emptyset\}$                                    ▷ List of branching nodes
  $\mathcal{B} \leftarrow \{\emptyset\}$                                    ▷ List of branch pixels
  while $|\mathcal{U}(\mathcal{S})| > 0$ do
    $p \leftarrow \mathcal{U}(\mathcal{S})$                         ▷ Extract a random unvisited pixel
    $\mathcal{N}, \mathcal{B} \leftarrow find\_branch(p, \mathcal{N}, \mathcal{B})$
  end
  function find_branch(p, $\mathcal{N}, \mathcal{B}$) is
    if $|neighbors(p)| > 2$ then
      $\mathcal{N} \leftarrow \mathcal{N} \cup \{p\}$
    else
      $\mathcal{B}' \leftarrow \{p\}$
      foreach $p' \in \mathcal{U}(neighbors(p))$ do
        $\mathcal{N}, \mathcal{B}' \leftarrow find\_branch(p', \mathcal{N}, \mathcal{B}')$        ▷ Find rest of the branch
      end
      $\mathcal{B} \leftarrow \mathcal{B} \cup \{\mathcal{B}'\}$
    end
    return $\mathcal{N}, \mathcal{B}$
  end
```

3.3.6 3D reconstruction

Finally, we reconstruct the plant in 3D using the information obtained in the previous steps. Algorithm 2 shows the functional definition of the complete algorithm. As seen, the reconstruction ($\mathcal{R}_{3D}$) is performed using the left input image (I_L), the improved segmentation (S') and disparity ($\mathcal{D}'$) images, the branches ($\mathcal{B}$) found by the Algorithm 1, and the intrinsic parameters of the calibrated stereo camera ($\mathcal{C}_{params}$).

To calculate the 3D reconstruction of the plant, each branch pixel found by the branching search algorithm is represented by a set of features (see Figure 3.9), these are: its coordinates (x,y,z), diameter or width of the branch (w), and average color of that area of the branch (c).

To calculate the color of a branch area, we extract a region of size 2β around a point (x,y) from the original input image, and, within this region, we calculate the dominant

Algorithm 2: Functional definition of the full algorithm

```
$I_L, I_R \leftarrow$ Stereo input images
$\mathcal{S} \leftarrow FCSN(I_L)$                                    ▷ Section 3.3.1
$\mathcal{D} \leftarrow Disparity(I_L, I_R)$                          ▷ Section 3.3.2
$\mathcal{S}' \leftarrow Improve\_segmentation(\mathcal{S}, \mathcal{D})$   ▷ Section 3.3.3
$\mathcal{D}' \leftarrow Improve\_disparity(\mathcal{D}, \mathcal{S}')$     ▷ Section 3.3.3
$\mathcal{K} \leftarrow Skeletonization(\mathcal{S}')$                ▷ Section 3.3.4
$\mathcal{B} \leftarrow Branching\_search(\mathcal{K})$               ▷ Section 3.3.5
$\mathcal{R}_{3D} \leftarrow 3D\_reconstruction(I_L, \mathcal{S}', \mathcal{D}', \mathcal{B}, C_{params})$   ▷ Section 3.3.6
```

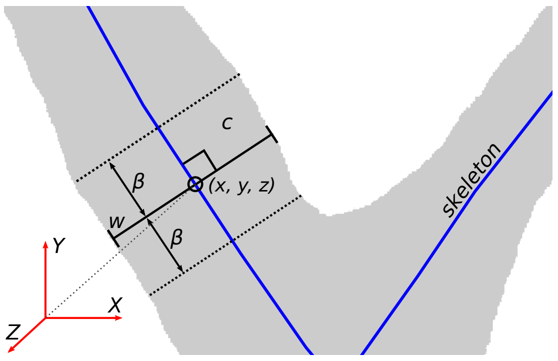


Figure 3.9: Features extracted for each point of a branch: position (x,y,z), width (w), and principal color (c) in a region of size 2β around the point.

color using the color quantization method proposed by Orchard *et al.* [176].

To calculate the width w of the branches, firstly we transform the skeleton into a set of lines. We do this by using the probabilistic implementation of the Hough transform [177], allowing a small gap (3 pixels in our implementation) between pixels to create the lines. We also set the threshold parameter to 10 (minimum number of intersecting points to detect a line) and the minimum segment length to 3 pixels in order to better adjust the lines according to the curvature of the branch. Once the line is obtained, we calculate a perpendicular line to each Hough line. These perpendicular lines start and finish at the boundaries of the segmentation image (see Figure 3.9).

The conversion from disparity values into real depth values can be performed directly on the basis of data obtained during the calibration process. Thus, the 3D coordinates (X, Y, Z) of the plant projected in the 3D space are calculated using the row x and column y of pixels in the 2D image, and their corresponding depth values z (obtained from the disparity map using Equation 3.1). Equation 3.2 shows how to calculate this equivalence.

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} \frac{z(x-c_x)}{f} \\ \frac{z(y-c_y)}{f} \\ z \end{pmatrix} \tag{3.2}$$

where c_x and c_y are the principal point (image center) and f is the focal length of the camera (in pixels).

The obtained reconstruction is processed to determine the branches to be cut according to a series of pruning rules. In particular, the information obtained allows us to analyze the rose bush to select the cut points according to different criteria, such as those that exceed a certain height, grow towards the center of the plant, or have a given branch thickness or color (dry branches usually have dark colors).

3.4 Datasets

For the evaluation of the proposed method we used a total of five datasets, two downloaded from the state of the art (called TB-Roses v2 [178] and Arabidopsis [151]), and three datasets created by the authors, which jointly are called ROSeS (*Roses for Object Segmentation and Skeletonization*)[3]. These three datasets are divided into: S-ROSeS, a synthetic dataset of rose bushes, H-ROSeS, a hybrid dataset with 200 real indoor images but with realistic plastic plants, and R-ROSeS, a completely real dataset with 100 photos of rose bushes taken in different botanic gardens.

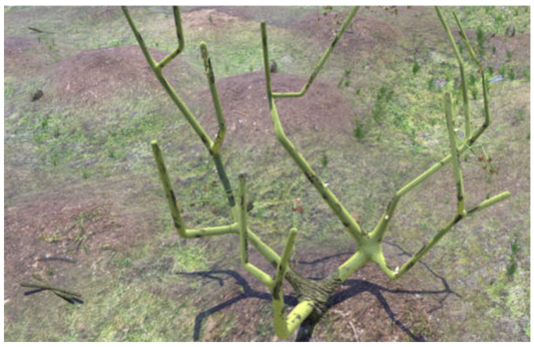
H-ROSeS and R-ROSeS were captured using the same stereo camera, with a resolution of 720×480 px, an interocular distance (base line) of 0.03m. The segmented ground truth was obtained by labelling manually each image at the pixel level.

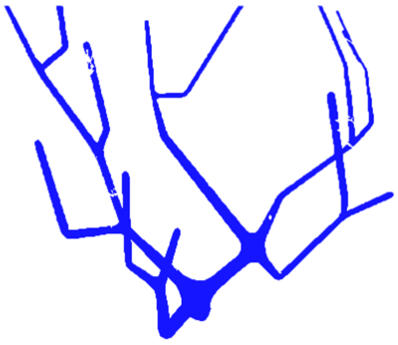
S-ROSeS consists of 5760 stereo images with a resolution of 720×480 px obtained from 160 different views of 36 synthetic rose bushes generated with Blender[4]. The images were generated at different distances and perspectives by rotating the camera around each plant and moving the camera closer and farther from it. Each of these bushes is unique. They were created following the morphology of real rose bushes with a mean height of 0.6m $\pm$ 0.20m. The 36 synthetic bushes are distributed in 4 different outdoor environments (9 bushes per environment). The environments differ

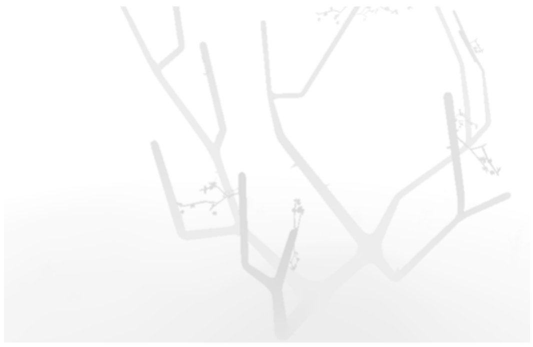
[3]The three datasets of ROSeS are available for the scientific community at `http://trimbot2020.webhosting.rug.nl/resources/public-datasets/`

[4]Blender (`https://www.blender.org/`) is a free and open source 3D creation suite.

in background, light conditions and position of the sun. Each stereo pair has its corresponding ground truth with the segmented image, the disparity map, and the skeleton for both left and right views (see Figure 3.10). These images were also generated using Blender's properties. To obtain the depth map of the synthetic dataset, we simulated a stereo camera with a parallel stereoscopic configuration, a sensor size of 32 mm, a focal length of 0.035m, and a baseline of 0.03m.


(a) Input image


(b) Segmentation


(c) Depth map

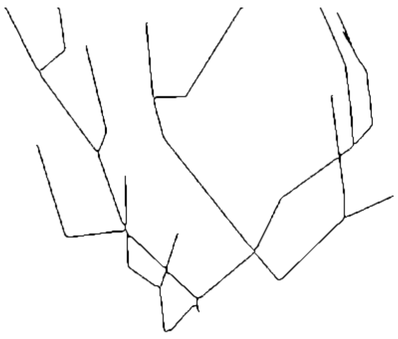

(d) Skeleton

Figure 3.10: Ground truth of the S-ROSeS dataset, where (a) is the synthetic input image, (b) is the pixelwise segmentation, (c) is the depth map, and (d) is the skeleton of the plant. These example images are from the view of the left camera. The dataset also includes the images obtained from the right camera view.

TB-Roses v2 dataset [178] is composed of 319 images of rose bushes recorded in a real garden with a resolution of 960×540 pixels[5]. It was designed for testing algorithms for segmentation and delineation of rose branches in applications of gardening robotics. The images are provided together with the ground truth marking the segmented branches.

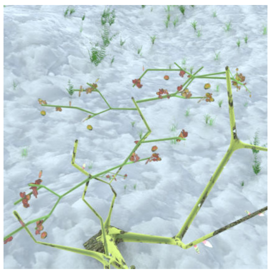
The Arabidopsis dataset [151] consists of 160 images with a size of 2208×1656 px

[5]TB-Roses v2 is publicly available at: `https://gitlab.com/nicstrisc/RUSTICO/tree/master/data`

of twelve *Arabidopsis* genus plants taken indoor[6]. The images were taken by rotating a turntable by 10 degrees of yaw. This dataset includes the ground truth with the segmentation and the branching structure.

Figure 3.11 shows some example images of the datasets. For S-ROSeS, four examples with four different backgrounds are included (see Figures 3.11a, 3.11b, 3.11c, and 3.11d). For the rest, several examples per dataset are also included (Figures 3.11e to 3.11l). As can be seen, we collected a variety of datasets, not only in the type and morphology of the plants but also in the type of environment (including indoor and outdoor), backgrounds (without always using a homogeneous color), and different lighting conditions (Figures 3.11d and 3.11h are darker, and the camera in Figure 3.11i faces the sun).

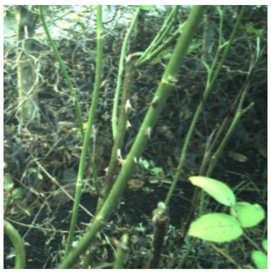
(a) S-ROSeS

(b) S-ROSeS

(c) S-ROSeS

(d) S-ROSeS

(e) H-ROSeS

(f) H-ROSeS

(g) R-ROSeS

(h) R-ROSeS

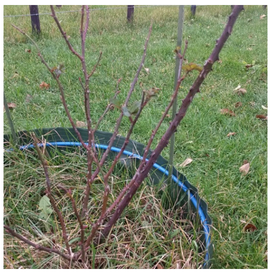
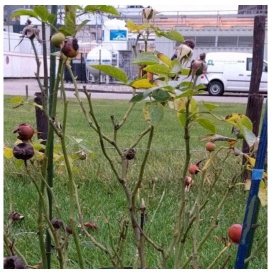
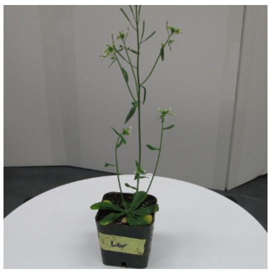
(i) R-ROSeS

(j) TB-Roses v2

(k) TB-Roses v2

(l) Arabidopsis

Figure 3.11: Example images of the different datasets used for the evaluation.

[6]Arabidopsis dataset is publicly available at: `http://kobus.ca/research/data/eccv_16_plants/index.html`

Table 3.2 shows a summary of characteristics of the different datasets used, including their type, number of samples, resolution, brightness, and types of ground truth included. As can be seen, some types of ground truth are not available for all datasets. Segmentation ground truth is the only one that is available for all, but the disparity and the skeleton are only included in S-ROSeS. So the validation in each case can only be done for the available data. This table also includes the maximum and minimum average image brightness values per dataset (in the range [0, 255]). These values clearly show how the outdoor datasets have much more variable lighting.

Name	**# Images**	**Resolution (pixels)**	**Type**	**Min/Max average brightness**	**Segment.**	**Disparity**	**Skeleton**	**Indoor**	**Variable lightning**
S-ROSeS	5760	720×480	Synthetic	87 / 193	✓	✓	✓	–	✓
H-ROSeS	200	720×480	Hybrid	115 / 136	✓	–	–	✓	–
R-ROSeS	100	720×480	Real	70 / 201	✓	–	–	–	✓
TB-Roses v2	319	960×540	Real	85 / 169	✓	–	–	–	✓
Arabidopsis	160	2208×1656	Real	111 / 143	✓	–	–	✓	–

Table 3.2: Summary of characteristics of the datasets evaluated, including the number of images, their resolution, their type (real, hybrid or synthetic), the minimum/maximum average image brightness values (in the range [0, 255]), the ground truths (segmentation, disparity and/or skeleton), whether they are indoor or outdoor, and if they have variable lighting.

In all the experiments we used an n-fold cross validation, which yields a better Monte-Carlo estimate than when performing the tests with a single random partition [179]. Therefore, the datasets were divided into n subsets, using, for each fold, one of the partitions for test (with $1/n$ of the samples) and the rest for training $(1-1/n)$.

Partitions were created by separating the datasets according to the sequences or backgrounds used, with the intention of creating mutually exclusive subsets. For example, for S-ROSeS, $n = 4$ partitions were created (1 for each type of background), for H-ROSeS $n = 2$ partitions (corresponding to the two different sequences), for R-ROSeS $n = 4$ (since the images were taken in two different gardens and for each one 2 sequences were recorded), for TB-Roses $n = 3$ (corresponding to three sequences of images), and for Arabidopsis $n = 4$ (with 3 plants per partition, since it has 12 plants in total).

For tuning the hyperparameters (see Section 3.5.1), the training partition was divided into two, assigning 10% of these samples for validation and the rest for training. The classifier was trained and evaluated n times using these sets, after which the average results plus the standard deviation σ were reported.

3.5 Experiments

In this section we evaluate the different parts of the proposed method using the datasets described in Section 3.4. First, the proposed FCSN segmentation network is evaluated, analyzing its different hyperparameters and comparing it with other state-of-the-art methods (see Section 3.5.1). Next, the calculation of the disparity and the post-processing step used to combine the segmentation and disparity images are assessed in Section 3.5.2. Section 3.5.3 compares five different skeletonization methods used for the detection of branches. Finally, the accuracy of the 3D reconstruction obtained is evaluated in Section 3.5.4.

All these experiments were performed using a Razer Blade 14 with Intel(R) Core i7-6700 CPU @ 3.40GHz (4th Gen) with 16 GB DDR4 RAM, an Nvidia GeForce GTX 1070 GPU, and ROS Kinetic with Ubuntu 16.04 as operating system.

3.5.1 FCSN evaluation

In this section we evaluate the FCSN proposed to perform the segmentation of the branches. First, different network hyperparameters are analyzed. For these initial experiments we used the 4 sets of plants from the S-ROSeS dataset. Subsequently, using the best model and parameters found, we evaluate (using the same parameters) the rest of the datasets and compare the proposed method with other approaches from the state of the art.

In order to assess the performance of the proposed method, three evaluation metrics widely used for this kind of tasks were chosen, they are: Precision, Recall, and F_1, which can be defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}} \tag{3.3}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}} \tag{3.4}$$

$$F_1 = \frac{2\cdot\text{TP}}{2\cdot\text{TP}+\text{FN}+\text{FP}} \tag{3.5}$$

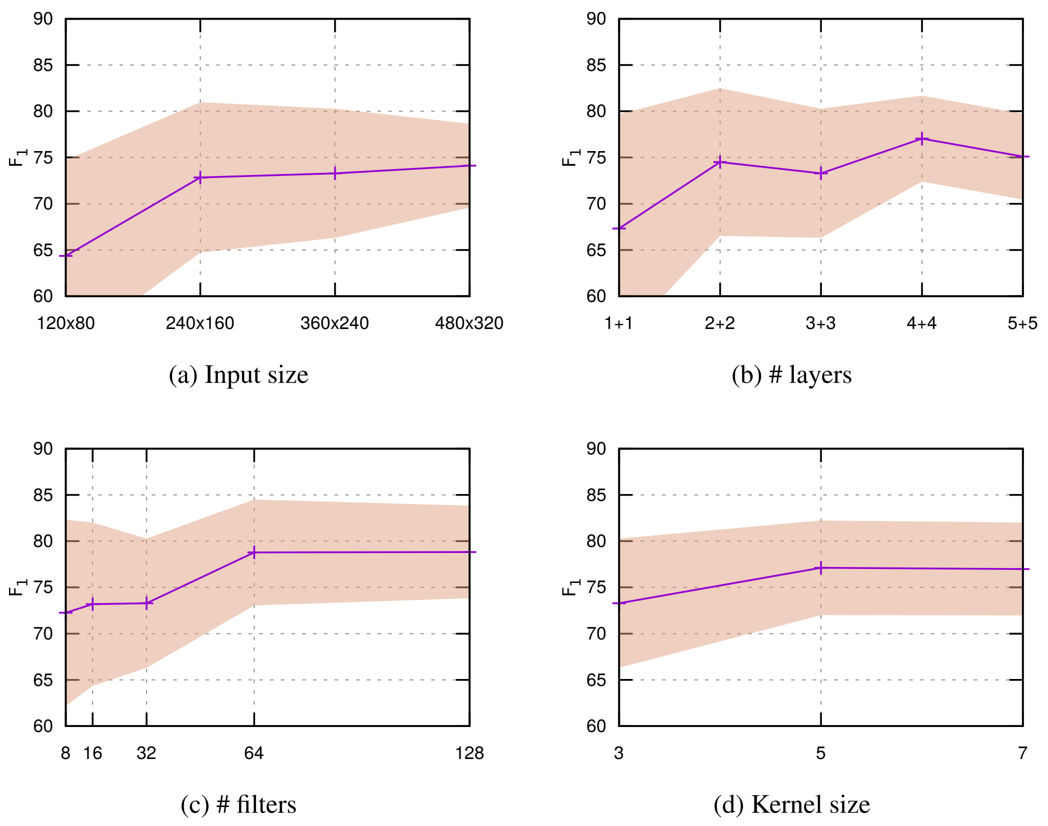
where TP (True Positives) denotes the number of correctly segmented branch pixels, FN (False Negatives) the number of non-segmented branch pixels, and FP (False Positives) the number of background pixels incorrectly given as branch pixels.

3.5.1.1 Hyperparameters evaluation

To select the best hyperparameters and configuration for the FCSN, we performed a *grid-search* process [161] using the S-ROSeS dataset. The configurations evaluated include variations in the network input size (from 120px to 480px of width and maintaining the original aspect ratio to calculate the height), in the number of layers (from 2 to 10), in the number of filters per layer (between 8 and 128), and in the kernel size (between 3 and 7). In each experiment, only one parameter was changed, setting the rest to a standard network configuration with an input size of 360×240px, 6 layers of depth (3 encoding + 3 decoding layers), 32 filters per layer with a kernel size of 3×3, with a *selectional* threshold δ of 0.5, without equalizing the input image, and only normalizing the input values dividing by 255.

Figure 3.12 shows the average results plus the standard deviation of these experiments. The reported results are the average of the 4-fold cross validation, where, for each fold, we used one of the S-ROSeS dataset partitions for test (25% of the samples) and the rest for training (75%), without mixing neither the types of plants nor the backgrounds, consequently dividing it into 4 mutually exclusive sub-sets. As the stopping criterion for tuning the hyperparameters, the training partition was divided into two, assigning 10% of the samples for validation and the rest for training.

The average F_1 when varying the input size is shown in Figure 3.12a. As seen, the best results (with the lowest standard deviation) were obtained using the larger input size (480×320px). Even larger sizes were also evaluated, but they increased the requirements of the machine, which did not allow their use due to the hardware specifications of the robotic platform. In the case of the number of layers (Figure 3.12b), an increasing trend is also observed, obtaining the best result with 4+4 layers and then getting worse slightly. The standard deviation also decreases up to 4+4 layers and it then stabilizes. In this case, it was not possible to add more layers due to the base input size (360×240px), since in each layer the size is divided by 2. Figure 3.12c shows the result obtained when varying the number of filters per layer. In this case, using more filters also increases the F_1 and reduces the standard deviation. This improvement is more noticeable from 16 to 64 filters, increasing more slightly later when adding more filters. Figure 3.12d shows the average F_1 for the three kernel sizes evaluated, obtaining the best results with a kernel size of 5×5. In this case, the standard deviation hardly varies.



(a) Input size (b) # layers

(c) # filters (d) Kernel size

Figure 3.12: Average F_1 (%) plus standard deviation (in light red) of the grid-search process when varying (a) the input image size, (b) the number of encoding + decoding layers, (c) the number of filters per layer, and (d) the kernel size of the convolutional filters.

We now proceed to analyze the influence of the normalization type as well as the equalization applied to the input data, using the best configuration found in the previous experiments.

Literature cites different ways to normalize the data used to feed a network [180, 164], but the most appropriate technique depends on the particular problem. The most common normalization methods are:

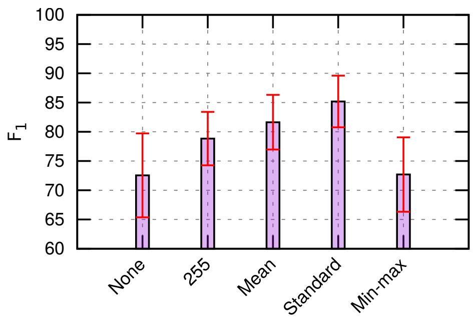
$$Z_{standard} = \frac{M - mean(M)}{std(M)} \qquad Z_{mean} = M - mean(M)$$

$$Z_{min-max} = \frac{M - min(M)}{max(M) - min(M)} \qquad Z_{norm} = M/255$$

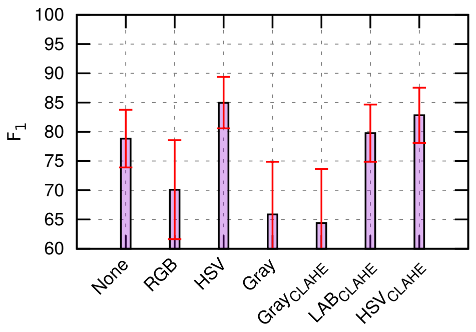
where M is the input matrix containing the raw image pixels from the training set. For the normalization of the test set we used the same mean, deviation, max, and min values calculated for the training set.

Moreover, since it was observed that the image contrast and brightness affected the result obtained significantly, different types of equalization were also analyzed. The equalization of the histogram applies a transformation on the image in order to obtain a histogram with a uniform distribution of colors (or levels of gray) that improves the contrast of the image. For this, the histogram equalization was evaluated using gray values and both RGB and HSV color spaces. In addition, the CLAHE (Contrast Limited Adaptive Histogram Equalization) method [181] was evaluated in gray and both CIELAB and HSV color spaces. This method performs an adaptive equalization by regions controlling the limit of the equalization made to not overamplify noise in homogeneous regions.

We evaluated these types of normalization and equalization on the proposed network, including the option of not normalizing or equalizing the data. Figure 3.13 shows the average results plus the standard deviation of these experiments. As before, each result is the average of the 4 folds using the best configuration found in the previous experiments and only varying the type of normalization or equalization. The type of data normalization (see Figure 3.13a) considerably affects the result obtained, since the difference between the best and the worst result exceeds 12%. The best F_1 is obtained using the standard normalization, followed by the mean norm. In addition, in these two cases, the standard deviation is quite similar. For the equalization type (see Figure 3.13b), a significant difference in the results obtained is also shown. Both gray scale and RGB equalizations seem to worsen the result, however, HSV and CLAHE (both in RGB and HSV color spaces) equalizations does improve it. The best result is obtained with the HSV equalization, which also reports the lowest standard deviation and is better than the option of not equalizing by 6.15%.



(a) Normalization type

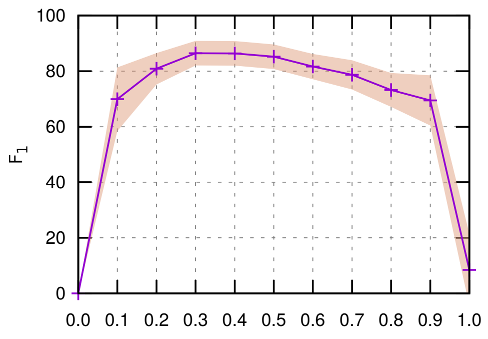


(b) Equalization type

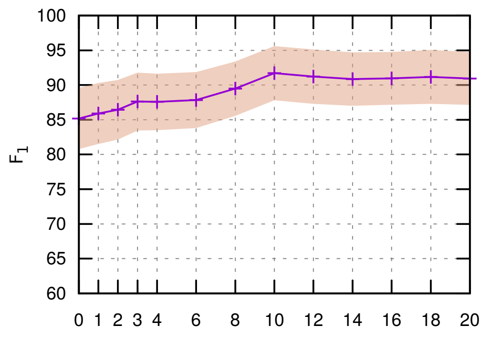
Figure 3.13: Average F_1 (%) plus standard deviation obtained for the different types of (a) normalization and (b) equalization considered.

Subsequently, we also evaluated the *selectional* threshold δ that is applied to the output of the network to determine (select) the pixels belonging to a branch. Figure 3.14a shows the result of varying this threshold between 0 and 1. The network obtains results higher than 80% with thresholds between 0.2 and 0.6, obtaining the best F_1 and the lowest standard deviation with a threshold of 0.3.

Finally, the influence of the data augmentation process was evaluated. For this, the number of samples of the training set was artificially increased by applying different types of random transformations to these images (This process is described in Section 3.3.1). In order to evaluate the improvement obtained with this augmentation process, we carried out an experiment in which we gradually increased the number of random transformations applied to each image from our training set, and evaluated it using the test set. Figure 3.14b shows the average results plus the standard deviation of such experiment, where the horizontal axis represents the augmentation factor and the vertical axis the F_1 obtained. As seen, the highest improvement is obtained at the beginning, after which the results begin to stabilize and stop improving after 10 augmentations. In this case, the standard deviation is quite stable, it is only slightly reduced by adding the first augmented images.



(a) *Selectional* threshold δ



(b) Data augmentation factor

Figure 3.14: (a) Influence of the *selectional* threshold δ (horizontal axis) on the final F_1 result obtained (vertical axis). (b) Average results of the data augmentation process. The horizontal axis represents the number of augmentations and the vertical axis the average F_1 (in percentage). Standard deviation is also shown in light red.

The finally selected configuration for the FCSN network was an input size of 480×320px, 4+4 layers, 128 filters per layer with a kernel size of 5×5, standard normalization, HSV equalization, a *selectional* threshold δ of 0.3, and augmenting the data by a factor of 10 (i.e. generating 10 random transformations for each training image). Table 3.1 shows a summary with these settings.

3.5.1.2 FCSN results

Once the best architecture for the FCSN network was obtained, it was evaluated with all the datasets described in Section 3.4. In addition, we compared it with the results obtained by other state-of-the-art methods, these are: U-Net [156], SegNet [39], and DeepLabv3 [182]

In addition to Precision, Recall and F_1 metrics, the Intersection over Union (IoU) [183] metric was used for evaluation. The metrics previously used are suitable for unbalanced datasets but in some cases they are not the most fair because they measure the precision at the pixel level, but not whether the algorithm has detected a branch or not. So it is possible that thick branches are detected very well but thin branches not, and still obtain good results. It is also possible that the algorithm makes mistakes in the branches' edges and gets a low result with these metrics when, in fact, all the branches are being detected correctly.

The IoU metric helps to measure whether the algorithm correctly detects all the branches and also how well it detects their size and location. To calculate this metric, we map each branch[7] of the ground truth (gt) onto the segmentation proposals (bp) with which it has a maximum IoU overlap according to the following equation:

$$IoU = \frac{\text{area}(B_{bp} \cap B_{gt})}{\text{area}(B_{bp} \cup B_{gt})} \tag{3.6}$$

where $\text{area}(B_{bp} \cap B_{gt})$ depicts the intersection between a branch proposal and the ground truth, and $\text{area}(B_{bp} \cup B_{gt})$ depicts its union.

We also calculate the metric F_1 at the branch level considering as TP when the IoU value exceeds a certain threshold λ (by convention $\lambda = 0.5$). We consider as FP the wrong detections (i.e., when a B_{bp} does not overlap with any B_{gt}), and as FN when a ground truth branch is not detected. Note that if multiple detections of the same branch are predicted, only the first one is counted as positive and the rest as negatives.

Table 3.3 shows the results obtained by each of the methods compared. The results have been grouped by dataset, also showing the average at the end of the table. The best result obtained by dataset and on average is marked in bold for each metric.

As can be seen, the proposed method obtains better results than the rest of approaches, except for the precision with S-ROSeS. However, this result is only 0.6 worse than that obtained by U-Net, and, given the standard deviation values (3.2 and 4.5, respectively), this difference can be neglected. Moreover, if we analyze the F_1 metric

[7]To perform this process, we manually separated each branch from the ground truth.

at pixel level, it shows that FSCN obtains a better overall segmentation. Therefore, it correctly recovers a greater number of branches, as shown by the metric F_1 at branch level. On average, the proposed method obtains a F_1 8.18% better than the next best result at pixel level, and 7.33% if we analyze it at branch level. The datasets for which a greater improvement is obtained with respect to the other methods are Arabidopsis (10.75% better than the next best result at pixel level), H-ROSeS (8.55% better), and R-ROSeS (5.98% better), which are the datasets that present a greater difficulty since they have thinner branches, more complex backgrounds, and a greater number of branches to segment.

		Pixel level			**Branch level**	
Dataset	**Method**	**Precision**	**Recall**	F_1	**Avg. IoU**	F_1
S-ROSeS	U-Net	**91.08** ± 3.2	88.50 ± 4.5	89.77 ± 3.9	90.21 ± 9.5	93.63 ± 7.6
	SegNet	88.84 ± 2.9	77.07 ± 11.1	81.81 ± 8.3	83.60 ± 7.9	87.79 ± 5.0
	DeepLabv3	76.94 ± 6.2	84.63 ± 5.5	80.59 ± 5.8	79.64 ± 6.8	81.21 ± 6.5
	FCSN	90.47 ± 4.5	**93.18** ± 3.0	**91.70** ± 3.9	**94.16** ± 7.3	**97.09** ± 4.2
H-ROSeS	U-Net	48.27 ± 0.7	66.15 ± 4.3	55.80 ± 2.0	66.11 ± 9.5	76.61 ± 7.1
	SegNet	56.92 ± 1.7	58.95 ± 1.5	57.48 ± 1.6	55.11 ± 11.5	64.98 ± 11.2
	DeepLabv3	50.79 ± 0.3	55.67 ± 0.7	53.12 ± 0.5	61.75 ± 10.7	67.34 ± 11.8
	FCSN	**61.20** ± 1.2	**71.79** ± 3.5	**66.03** ± 2.6	**69.40** ± 8.3	**80.35** ± 7.3
R-ROSeS	U-Net	69.45 ± 8.0	71.00 ± 8.9	70.19 ± 8.4	68.15 ± 13.0	73.88 ± 13.5
	SegNet	70.58 ± 9.5	73.69 ± 8.2	72.08 ± 8.8	70.08 ± 12.5	75.69 ± 12.2
	DeepLabv3	61.09 ± 2.5	61.49 ± 2.9	61.21 ± 2.7	71.85 ± 8.4	74.19 ± 8.9
	FCSN	**77.48** ± 7.8	**78.72** ± 5.6	**78.06** ± 6.6	**76.71** ± 8.7	**82.18** ± 8.2
TB-Roses	U-Net	65.53 ± 4.8	73.74 ± 3.0	69.38 ± 4.1	77.04 ± 6.5	80.09 ± 6.0
	SegNet	69.73 ± 0.4	79.48 ± 2.9	74.27 ± 1.0	82.71 ± 11.9	86.28 ± 12.2
	DeepLabv3	47.45 ± 0.9	59.23 ± 1.0	52.69 ± 0.9	80.89 ± 8.9	82.50 ± 9.0
	FCSN	**75.11** ± 0.7	**80.17** ± 2.4	**77.55** ± 1.5	**84.12** ± 7.0	**87.44** ± 7.7
Arabidop.	U-Net	47.45 ± 0.7	59.23 ± 0.8	52.69 ± 0.8	73.81 ± 10.8	78.06 ± 11.9
	SegNet	54.45 ± 2.9	65.46 ± 7.3	58.52 ± 8.1	77.49 ± 15.8	82.50 ± 16.4
	DeepLabv3	60.91 ± 3.4	61.02 ± 3.4	60.96 ± 3.4	79.14 ± 7.7	85.56 ± 8.2
	FCSN	**71.52** ± 3.1	**72.44** ± 4.2	**71.71** ± 3.5	**87.67** ± 7.1	**91.88** ± 7.3
Average	U-Net	64.35 ± 5.2	71.72 ± 5.1	67.56 ± 5.0	75.06 ± 10.1	80.46 ± 9.7
	SegNet	68.10 ± 5.5	70.93 ± 10.5	68.83 ± 6.5	73.80 ± 12.2	79.45 ± 12.0
	DeepLabv3	59.44 ± 3.4	64.41 ± 3.2	61.71 ± 3.1	74.65 ± 8.6	78.16 ± 9.1
	FCSN	**75.16** ± 5.4	**79.26** ± 5.6	**77.01** ± 4.6	**82.41** ± 7.7	**87.79** ± 7.1

Table 3.3: Results obtained by the different algorithms evaluated for each of the datasets. The best result obtained by dataset and on average is marked in bold for each metric.

To rigorously analyze the results obtained, we performed a statistical significance comparison by considering the paired sample non-parametric Wilcoxon signed-rank test [184]. More precisely, the idea is to assess whether the improvement observed in the segmentation performance (F_1 score at the pixel level) with the use of the FCSN is statistically significant in comparison with the result obtained by the other methods. For this, the results obtained for each dataset and fold were pairwise compared. By making this comparison, the proposed method obtains a p-value of 0.005, if we compare it with SegNet, and 0.002 with respect to U-Net and DeepLabv3. Therefore, this test reflects that our proposal significantly outperforms (considering a statistical significance threshold of $p < 0.01$, the most restrictive threshold normally used) the results obtained by the other state-of-the-art methods.

Figure 3.15 shows an example of the segmentation result obtained by each method for a sample image of each of the evaluated datasets. To make this figure more informative, we selected samples in which all methods made visible mistakes. The input image is shown in the first column. The rest of the columns correspond to the results obtained by each method, where black and white areas depict correct detections of branches and background, respectively, and red and blue pixels depict FP and FN of branches, respectively. These figures help to visualize the accuracy of the methods and to understand where the errors occur. As can be seen, U-Net and SegNet generate more noise due to non-branch elements (general background, leaves, etc.), while DeepLabv3 tends to make a thicker segmentation. The proposed method also makes mistakes, but they are mainly produced at the contours of the branches. This qualitative analysis also helps us to determine that for real roses, an F_1 score above 70% is enough to segment the branches from the background without losing important information of the foreground object. However, for extreme scenes like the images in R-ROSeS, it would be difficult to recover a complete information of the plant using the segmented result.

3.5.2 Assess segmentation and disparity combination process

The post-processing step used to combine the segmentation and disparity images is evaluated in this section. Several depth metrics from prior works [185] are used for the evaluation of the disparity results, which are:

- Root Mean Squared Error (RMSE): $\sqrt{\frac{1}{T}\sum_i^T ||d_i - d_i^{gt}||^2}$
- Squared Relative difference (Sq-Rel): $\frac{1}{T}\sum_i^T \frac{||d_i - d_i^{gt}||^2}{d_i^{gt}}$

where T is the total pixels in the image, d_i is the estimated depth for the pixel i, and d_i^{gt} is the ground-truth depth.

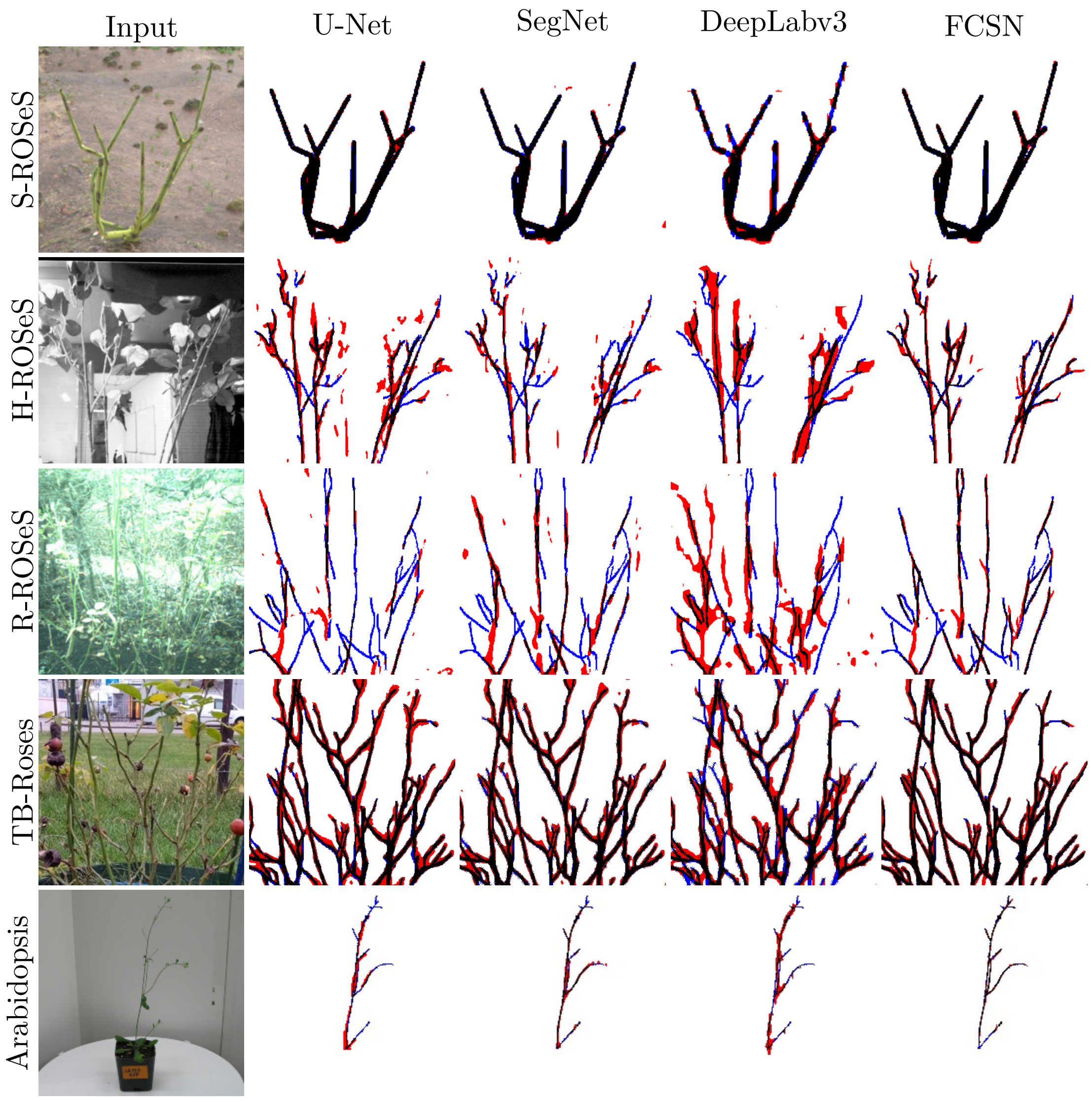


Figure 3.15: Examples of the segmentation result obtained by each method for a sample image of each of the evaluated datasets. The first column shows the input image. The rest of the columns correspond to the results obtained by each method, where black and white areas depict correct detections of branches and background, respectively, and red and blue pixels depict FP and FN of branches, respectively.

Table 3.4 shows the results of this experiment, where the disparity evaluation is shown in the first four columns and the results obtained for segmentation in the last three. The disparity was evaluated for the complete image (first two columns) and also at the branch level (i.e. taking into account only the pixels within the segmentation ground truth, see third and fourth columns). In our case, the results at the branch level are the most interesting, since on the one hand the disparity improvement process only affects branches, and on the other hand, the rest of the process only uses these depth values. As seen, these processes help to improve the original result obtained for both segmentation and disparity (with the BM method). The improvement obtained for the disparity is the most significant, reducing the RMSE from 0.2527 to 0.0869 at branch level.

	Disparity				Segmentation		
	Full image		Branch level				
	RMSE	Sq-Rel	RMSE	Sq-Rel	Precision	Recall	F_1
Original	0.5648±0.2	0.0917±0.1	0.2527±0.2	0.0310±0.1	90.47±4.5	93.18±3.0	91.70±3.9
Improved	0.5082±0.2	0.0630±0.0	0.0869±0.1	0.0039±0.0	92.86±4.4	92.59±3.1	92.96±3.8

Table 3.4: Results obtained for the calculation of the disparity and segmentation with the S-ROSeS dataset before and after applying the segmentation and disparity combination process. For the disparity, the error calculated for the complete image and at the branch level is shown.

Figure 3.16 shows an example of the result obtained after applying the segmentation and disparity combination process. As can be seen in the figure on the right, the final result substantially improves the original disparity (central image, obtained with the BM method), since this method fills the empty areas, obtaining a dense disparity for the bush branches. Using this result, the intersection with the segmentation image is then calculated to eliminate all the background noise and leave only disparity values for branches.

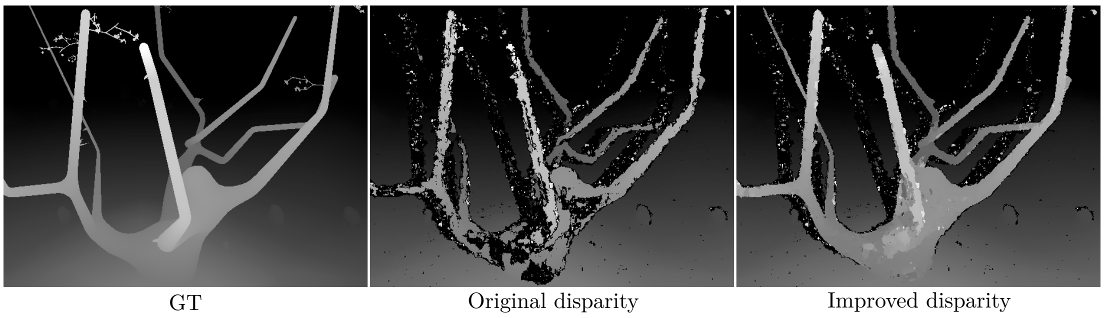


Figure 3.16: Example of the result obtained through the post-process to improve the disparity. The first image shows the disparity ground truth, the central image shows the original result obtained by the BM method, and the right image shows the improved disparity obtained after the segmentation and disparity combination process.

3.5.3 Skeleton evaluation

In this section we evaluate the skeletonization process individually, without considering the errors made in the previous steps. For this, we analyze the result using the segmentation ground truth as input of this algorithm, leaving the evaluation of the whole workflow for the 3D reconstruction section.

To evaluate the skeletonization, the F_1 score is used by comparing the obtained 3D skeleton S_o and the ground truth skeleton S_{gt} at the pixel level. Because the skeleton is only one pixel wide, a certain tolerance distance d is considered to determine if a pixel is a TP (similar to Zou *et al.* [186]). Therefore, a pixel is considered to be TP if it is located no more than d pixels away from the ground truth. These experiments were performed using the hardware specifications described at the beginning of this section.

The following five methods were evaluated using this criteria to find the skeletonization method that suits better to our problem: Zhang *et al.* [171], Parallel thinning [172], 3D skeletonization [173], Medial axis [174], and RUSTICO [175]

Given that RUSTICO is not defined as a skeletonization method per se and uses as input a gray scale image instead of a binary one, it delineates curved objects from the background as well, as seen in Figure 3.17h. Because of this, two experiments using this method were performed. One by comparing the raw output of RUSTICO with the ground truth skeleton and other by only considering the skeleton obtained by RUSTICO inside the ground truth segmented image (denoted by RUSTICO+seg), to make a more fair comparison with the other methods (which directly use the segmented image).

Table 3.5 shows the quantitative results of this experiment using the S-ROSeS dataset and different d values ($d = [0,1,2]$). As seen, all the skeletonization methods achieved F_1 scores above 0.3, 0.7 and 0.9 for $d = [0,1,2]$ respectively, with the exception of RUSTICO and RUSTICO+seg, which got the lowest results. However, these results are similar to those obtained in the original paper for the branch segmentation task ($F_1 = 42.65$ for the TB-Roses v2 dataset with a tolerance of 2 pixels [175]). The main cause of the low performance of RUSTICO is its optimization goal, which is not suited to find branches but to find curvilinear structures within a given width.

On average, the best F_1 results are obtained by the 3D skeleton method and medial axis. However, Zhang *et al.* [171] and parallel thinning are only $\sim 1\%$ worse. The main problem with the 3D skeleton is that it is computationally more expensive than the other methods (4.6121 sec. vs. 0.0390 sec. for Zhang *et al.* [171][8]) because it has to evaluate regions of $[3 \times 3 \times 3]$ per disparity layer. The problem with the medial axis method is that it is affected by the noise and bumps at the edges of the segmentation, which creates small branches that connects the main axis with those small protuberances. The parallel thinning method has a similar problem. The method

[8]These experiments were performed using the hardware specifications described at the beginning of this section.

that creates less small branches in the skeleton and runs faster is by Zhang *et al.* [171]. Given that we want to implement a method that runs fast and also that creates as few FP branches as possible, we eventually selected this method.

Method	**Precision**			**Recall**			F_1		
	d=0	d=1	d=2	d=0	d=1	d=2	d=0	d=1	d=2
Zhang & Suen	**90.25**	**98.61**	**99.47**	18.47	61.88	83.96	30.66	76.04	91.06
Parallel thin.	90.22	98.60	99.46	18.26	62.06	84.33	30.37	76.17	91.27
3D skel.	87.97	97.66	98.82	**19.05**	**64.99**	**87.40**	**31.32**	78.04	**92.76**
Medial axis	89.68	98.49	99.39	18.92	64.63	86.12	31.25	**78.05**	92.28
RUSTICO	21.74	45.47	54.99	15.08	42.82	60.96	17.81	44.11	57.83
RUSTICO+seg	73.25	93.63	96.81	14.92	42.46	60.35	24.79	58.43	74.35

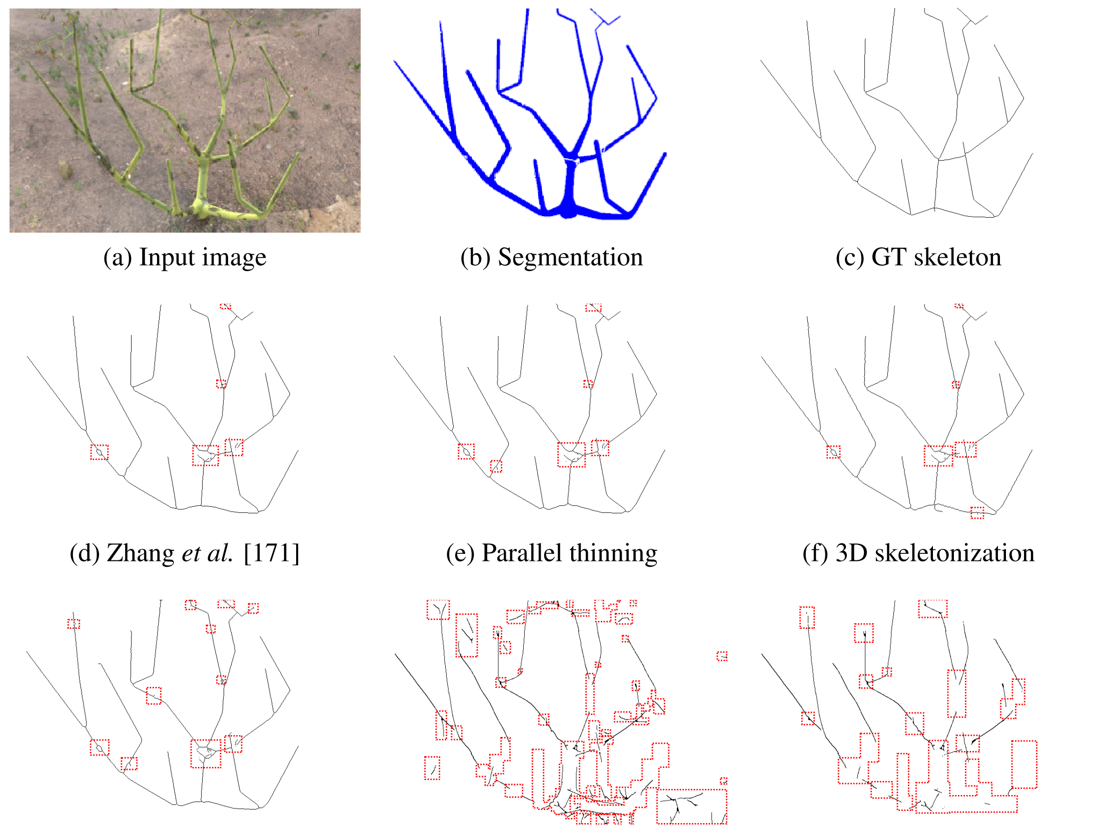
Table 3.5: Comparison of the results obtained by the different skeletonization methods considered using the segmentation ground truth from the S-ROSeS dataset. The best result obtained for each metric and threshold d is marked in bold.

Figure 3.17 shows an example of the skeleton obtained by each method for a sample image from the S-ROSeS dataset. Red rectangles mark the errors made by each method. As can be seen, visually (if we compare it with the ground truth shown in Figure 3.17c) all methods, except RUSTICO, get a good result. RUSTICO correctly detects the main branches but introduces a lot of noise and fails for the small branches (even after intersecting it with the segmentation, RUSTICO+seg). The other four methods work better, obtaining similar results, however, Zhang *et al.* [171] generates fewer incorrect small branches.

3.5.4 3D reconstruction

To evaluate the performance of the 3D reconstruction, we compared it with the 3D skeleton ground truth of the synthetic images. This ground truth has 2880 pointclouds, each one containing the 3D skeleton of a synthetic rose. The evaluation consisted of finding the mean of the minimum distance between each point of the ground truth and the reconstructed skeleton. Therefore, this metric measures how far the reconstructed skeleton is, on average, with respect to the ground truth. To make a fair evaluation, the depths of the reconstructed points were normalized by the furthest point in the ground truth per each plant.

Table 3.6 shows the result of this evaluation. It also shows the distance in each axis x, y and z, where z represents the depth, and y the axis that points to the ground. As can be seen, the average distance is less than 1 cm, being the error in the x-y plane less than the one made in the z-axis (due to the calculation of depth). However, this error is still less than 1 cm. This accuracy is enough for the pruning process because the end



(a) Input image (b) Segmentation (c) GT skeleton

(d) Zhang *et al.* [171] (e) Parallel thinning (f) 3D skeletonization

(g) Medial axis (h) RUSTICO (i) RUSTICO+seg

Figure 3.17: Examples of the skeleton obtained by each method for a sample image from the S-ROSeS dataset. First row shows: (a) the input image, (b) the segmentation GT, and (c) the skeletonization GT. The other two rows show the skeletonization results. Red rectangles indicate the mistakes made by each method.

effector has an opening size of 1.4 cm and is curved at the tool-tip (as seen in Figure 3.2), which allows the branch to slide into the center of the cutter even if the branch is not completely aligned with the center of the tool.

Figure 3.18 shows some examples of the 3D reconstruction obtained with different input images. This figure allows you to visually compare the result obtained by the proposed method and the ground truth. In addition, the branch separation obtained by Algorithm 1 is included in the 4th column.

3.6 Conclusions

A new method for the segmentation and 3D reconstruction of rose bushes from stereo images was presented. This method is part of a robotic system that is capable of moving through a garden towards a rose bush and pruning it according to a series of rules.

Min. distance	Mean (m)	std (m)
x axis	0.00293	0.00477
y axis	0.00336	0.00497
z axis	0.00619	0.00827
Overall	0.00859	0.01001

Table 3.6: Average distance between the 3D reconstruction and the ground truth pointclouds.

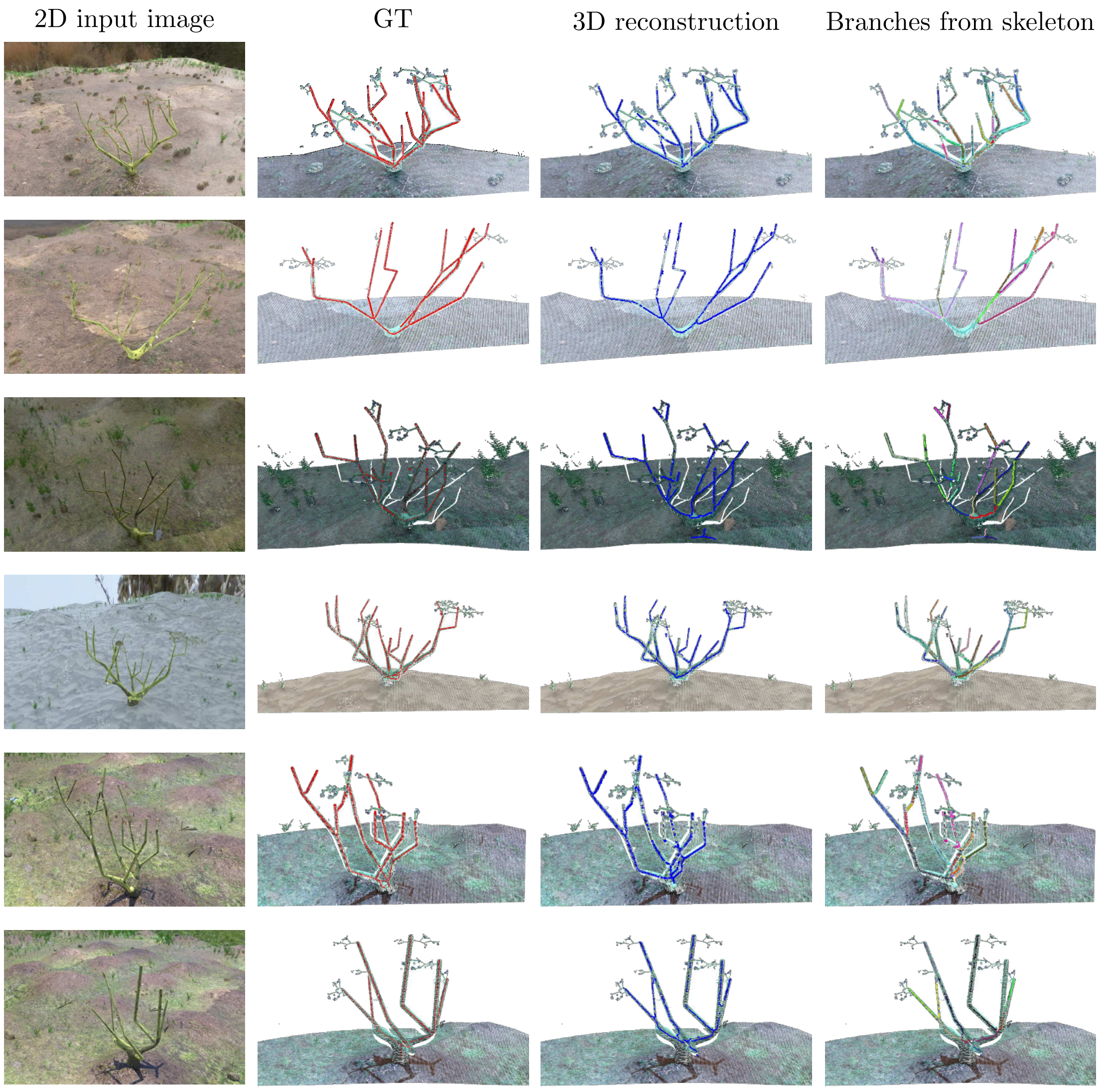


Figure 3.18: Output examples of the 3D skeletons obtained by our method (blue) and the ground truth skeleton (red), both superimposed over the pointcloud of the plant. The branches found using Algorithm 1 are also shown in the third column (each branch is marked using different colors).

The proposed vision method tries to solve this task without making assumptions about the characteristics of the plant, the type of environment, or the lighting conditions. The method is divided into several steps that try to improve the robustness of the result and to gather all the necessary information that allows the robot to select the branches to be pruned.

The main contribution of our workflow is its application in real environments and with different variations of rose bushes. Also, our method uses 2D morphological skeletonization to obtain the skeleton and branching structure of the plant and then back-projects this information into the 3D world using the camera parameters. The main goal of 2D skeletonization is to reduce the foreground regions in a binary image to a skeletal remnant that largely preserves the extent and connectivity of the original region while throwing away most of the original foreground pixels. Unlike methods that use 3D point clouds, the preservation of the connectivity of the regions is a good property to find branching structures without searching for an optimal distance between neighbor points to cluster them as part of the same branch.

The branch segmentation is one of the most important steps of the whole process, since it allows to select only those parts of the image that are branches (differentiating them from the general background, as well as other surrounding elements that can be very similar to branches, such as sticks, support stakes, fences, etc.) so that the rest of the processing pipeline can work with correct data. For this reason, the proposed method was specifically adjusted to work well in this type of task.

Five different rose bushes datasets were used to evaluate the pipeline, three of them compiled by the authors and two from the state of the art, including interior and exterior environments, as well as different types of rose bushes, backgrounds, and lighting conditions. As shown in Table 3.3, the segmentation method obtained an average F_1 score of 77% at the pixel level. It is 8.18% better than the next best result from the state of art. When evaluating at the branch level, the method correctly detected 88% of the branches, 7.33% better than the next best result. In addition, the significance of these results was validated by statistical tests.

The process proposed for the combination of segmentation and disparity improved the accuracy of both results, increasing the segmentation F_1 score by 1.26%, and reducing the RMSE of the disparity calculated at the branch level from 0.2527, for the original algorithm, to 0.0869 (see Table 3.4). As future work, this refinement can be improved by methods that focus on connecting close thin objects and penalize disconnected regions. Here, the pixel-wise accuracy metric should be replaced by alternative metrics such as rand-index [187] which penalizes disconnected regions.

Five different algorithms were evaluated for the 2D skeletonization, from which Zhang *et al.* [171] was chosen to create the skeleton of the segmented binary map of the plant. This method was chosen because it obtains a good F_1 score of 91.06% (Table 3.5) and also generates fewer small branches than the other methods, which is convenient when dealing with thin objects with many bifurcations.

The 3D reconstruction of the plant is obtained by using the camera parameters, the branching search algorithm, the disparity map and the segmentation obtained by the previous steps. The overall reconstruction is on average 0.859 cm far from the ground truth, as shown in Table 3.6, making it an accurate reconstruction both quantitatively and qualitatively. At the same time, the branches found in 2D helps the method to find the branching structure in 3D, without using any geometrical primitives or having different views of the plant.

As future work, the segmentation and disparity methods could be improved by including additional input information, such as the curvilinear structures obtained by RUSTICO. It could also improve the calculation of the branches to be pruned, so that, in addition to the pruning rules currently considered, take into account the shape and appearance of the rose bush, so that, for example, it does not grow in an unbalanced way (more on one side than on the other).

Chapter 4

Visual Servoing

The previous chapter shows how a classic image processing pipeline and deep learning can be combined to achieve a fast segmentation and disparity estimation of plants using a stereo camera. Moreover, it can be combined with 2D skeletonization methods to find the branching structure and radius of the plants.

However, knowing the morphology of the plant is not enough to make a robot follow gardening rules. The next step is to locate specific parts in the branches that needed to be cut. This chapter presents a solution for this part using visual servoing, to not only locate the parts in the branch but also to help the robot navigate towards them by refining their location. Also, unlike previous methods that need to constrain the background or place a set of cameras around the plant, our approach only requires a stereo camera and a manipulator to cut the branches. Our method also works in a real environment.

This chapter reuses the disparity estimation and plant segmentation capability from the previous chapter. However, we do not use the morphology estimator because it will make the performance of our method more difficult to assess, as the main goal is to develop a general localization method for visual servoing that works on branches.

The content of this chapter is from the published paper [2].

4.1 Introduction

Gardening is a repetitive and human-intensive task, however, it is difficult to automate given the challenges that it involves. These challenges are related to the unconstrained environment in a real garden. There are simple gardening tasks that have being automated like grass cutting [188, 189, 190], stationary fruit harvesting [191] and plant

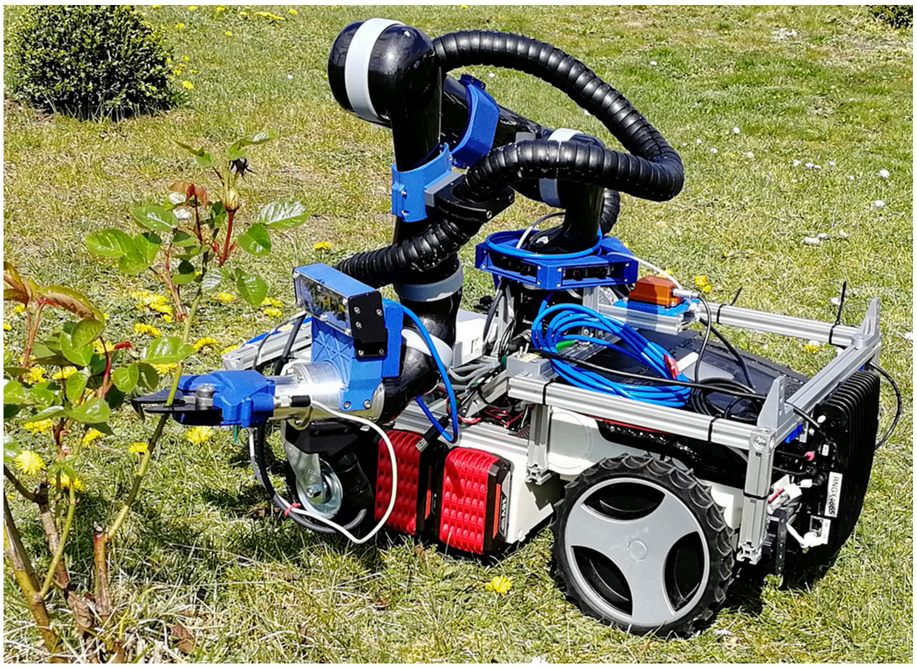


Figure 4.1: Overview photo of the TrimBot2020 rose cutter robot.

irrigation [192]. Although these tasks can be considered solved and highly commercialized, there are others that require more dexterity in terms of manipulation and approach.

One of these complex jobs is rose pruning. A rose bush requires regular pruning in late winter or early spring to keep it healthy, aesthetic and allow it to grow new and strong roses. This job requires a person to cut stems according to some gardening rules and to be able to approach the stems in different positions to cut them. As rose pruning is complex and demanding, its automation is useful and helpful in garden maintenance. The benefits go from small gardens at a house to big ones, e.g. national gardens.

There are very few previous works on rose pruning robots, with [12, 14] one of the few in the area. In these papers, they localize cutting points on a branch by using a relation between the diameter and the length of the branch. After that, they use position based servoing to reach the points. Unlike our approach, they cut only a single rose stem under a controlled environment with constant light and background.

A close approach to rose pruning is tree pruning [193, 194, 195]. [13] is the closest approach to our work. Here, the authors model a grape vine using multiple stereo cameras that surround the plant to find cutting points on stems. This process is done in a controlled environment inside a box, which covers completely the whole plant from its surroundings, resulting in a constant illumination and background.

Rose pruning can be considered as lying in the research field of precision agriculture, together with bush trimming [123], and fruit harvesting [124]. In [123], they trim bushes based on 3D shape fitting using an eye-in-hand arm robot. They send all the cutting positions as programmed trajectories to the robot without allowing any re-planning while approaching the bush. In [124], they harvest tomatoes using an image based visual servoing approach. They find the position of the fruit based on their round shape and characteristic color. Further work on fruit harvesting can be found in [196, 197]. Rose pruning and fruit harvesting share the same principle: find key points where the robot should cut or pick a target. Fruit harvesting can be considered an easier task given that the targets usually have distinctive characteristics like color and shape, making them easier to recognize. To prune a rose bush, the robot has to find cutting points on a stem in a highly populated[1] and small space where all the stems look alike and do not have any distinctive characteristic that shows the location of the cutting point.

Our task can be seen as a special case of grasping based on visual servoing. The difference to a normal grasping model is that these methods usually work with big objects with a defined body to grasp, whereas to prune a rose, the robot has to be really precise to fit a thin stem into the cutting tool (see Fig. 4.2).

To the best of our knowledge there is no robot able to prune a complete rose bush in a natural environment. To prune a rose bush aesthetically and keep it healthy, a robot has to cut its stems at a certain height from the ground. Therefore, the robot should be able to locate the stems and find where it should cut. Since the robot is supposed to prune roses under bright sunlight, active light sensors are not suitable, and laser scanners are expensive and limited by their coverage. Thus, RGB stereo cameras suit well in this task. Another important feature is robustness under changing conditions. The robot should be robust to track the cutting points when the stems are moved by the wind. Therefore, close-loop control based on vision is used to update the information while it approaches to the target.

Unlike normal grasping or fruit harvesting, having a proper end-effector is important for rose pruning because stems are thin and rose bushes are usually populated by many stems. In addition, the tool must be light enough to be carried by a small size mobile robot as its end-effector. Currently there is no pruner that can work and interact with a robot on the market. For this reason, designing a tool capable of cutting thin stems under program control is important for the success of the process.

This chapter presents a novel approach which is divided in the following steps.

[1]Highly populated by stems.

1. Scan the rose bush based on a pre-planned robot arm trajectory to acquire a complete 3D description of the bush in the form of a 3D pointcloud.
2. Segment the stems using an encoder-decoder CNN to separate the stems from the background and leaves.
3. Evaluate the pointcloud and find cutting locations based on the desired height.
4. Navigate towards the target stem location updating in real-time the location of the current target.
5. If the end-effector reaches the desired target (stem), send a signal to the cutting tool to cut it.

These steps result in not only a good estimation of the cutting points but also a robust servoing under changing conditions without compromising real-time target update and execution. The navigation of the mobile robot to the bush is not part of this pipeline because we assume the robot is already positioned in front of the bush at a cutting distance, and the ground surface around the plant is approximately planar and horizontal as in [135]. The proposed approach has the following contributions:

- **A novel rose pruning pipeline** based on stereo visual servoing and real time target update.
- **A real-time 2.5D servoing algorithm** using pointcloud and 2D images to locate stems and find cutting locations on them.
- **A dataset with more than 1200 labelled images of rose stems** used to train the encoder-decoder CNN to segment rose bushes[2].

4.2 Robot Setup

This section describes the setup of the robot. Fig. 4.1 shows an overview of the robot and its components. Although the figure shows a specific design, the proposed pipeline is more general and is not constrained to any type of robot or stereo camera, making it flexible for a re-implementation on different hardware[3].

[2]The dataset is at: `http://trimbot2020.webhosting.rug.nl/resources/public-datasets/`

[3]The robot setup was done by the Wageningen university & research team as specified in the declaration page.

4.2.1 Robot arm

The arm consists of a Kinova Jaco2 robot [198], a light-weight manipulator with lower power consumption mounted on a mobile robot; the details of the mobile robot base can be found in [135]. The 6 rotational joints provide the dexterity needed for the scanning and cutting actions. The arm has a spherical wrist configuration rather than a curved wrist. This configuration allows easier inverse kinematics (IK) computation and smoother movements than the usual curved wrist configuration.

4.2.2 Clipping tool

Market research started the design process of the clipping tool to find readymade solutions to cut stems in a cluttered environment. The commercially available Ciso pruner from Bosch was chosen as the base because of its simple on-off control system and the built in feedback mechanism, by means of position switches, to sense whether the knife is open or closed. The pruner is able to cut stems up to 14 mm in diameter and was mechanically modified in order to mount it on the Kinova robot arm (Fig. 4.2). Because of the ROS control architecture used for the main platform, a Maxon motor controller (Epos 2, 24/5, Maxon Motor AG, Switzerland) is used to drive the DC motor that opens and closes the jaws of the knife. An in-house modified version of the Maxon motor ROS wrapper *epos_harware* [199], that allows the reading of the digital I/O of Epos 2, is used to control the motor and read out the state of the position switches.

4.2.3 Stereo camera

On the last joint of the arm, on top of the clipping tool, a custom stereo camera [115] is mounted (Fig. 4.2). The camera module is mechanically configured to not obstruct the cutting action and to see as much of the scene as possible. The stereo camera has a resolution of 752×480 px, a diagonal $FoV \approx 68^\circ$ and a baseline ≈ 3 cm. The stereo calibration was done using Kalibr [200]. The camera uses as its global coordinate the base frame of the robot. Figure 4.2 shows that the camera housing has two pairs of stereo cameras with different baselines, however, only the camera with the smallest baseline is used because the cutter needs to get close to the target.

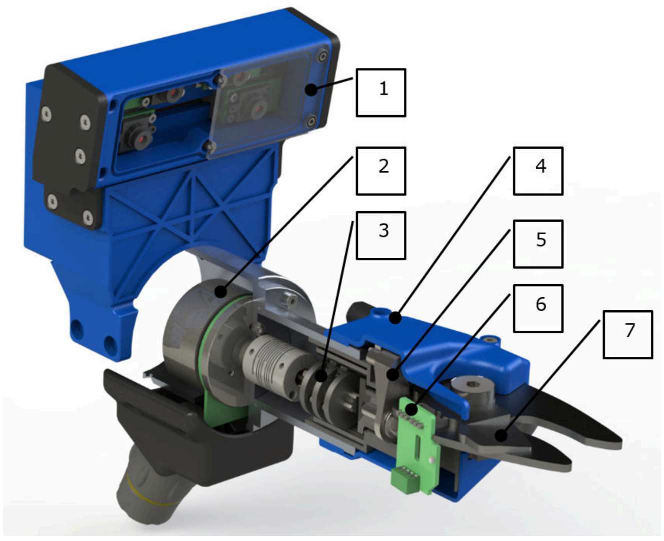


Figure 4.2: 3D design of the clipping tool. 1: 2 pairs of cameras in a 3D printed housing, 2: DC motor, 3: gear box, 4: protective cover, 5: drive lever of cutting blade, 6: position feedback switches, 7: cutting blade.

4.3 Methodology

The proposed method is divided into 5 steps which can be seen in Fig. 4.5. The following sections will describe each step of the process individually.

4.3.1 Scanning

The pipeline starts by scanning a rose bush in a predefined square shape trajectory, making a short pause at given poses to avoid motion blur when recording the image pairs. The square scanning trajectory has a size of 20×20 cm with its center located at the cutting height h. This is the starting position of the arm. The end-effector of the arm is ~ 0.6 m away from the bush. A sample of the images captured in each pose can be seen in Fig. 4.3a. The scanning is performed because a single viewpoint can only provide limited information due to complex occlusions between stems. Also, a rose bush is usually too big to be captured in a single shot by an eye-in-hand camera. One can argue that the robot can be placed far away from the plant to capture the whole bush at once, however, the farther it gets, the more difficult it is for a stereo matching algorithm to find the disparity of a single stem.

4.3.2 Stem detection

This section summarises the algorithm presented in Chapter 3, in order to provide context for the servoing and cutting process.

For each pose in the trajectory, a color image from the left camera and a disparity map are obtained. The disparity maps are computed by using Block Matching Stereo (BM). Although BM is not as accurate as state-of-the-art methods like Semi Global Block Matching (SGBM) or DispNet [201], it is a faster approach. BM obtains the disparity maps in real-time, whereas SGBM runs at ~ 4 FPS and DispNet at ~ 2 FPS[4].

The image from the left camera is used as input to an encoder-decoder CNN with residual connections to segment the stems from the background, similar to [202]. The network outputs a binary image where it assigns a value of 1 to all the pixels that form part of a stem and 0 to the background. This network outperforms most of the state-of-the-art segmentations for the branch segmentation task [1]. The binary output is used to mask the disparity image to obtain only the disparities of the stems. This masked disparity is then converted into a pointcloud. Figure 4.3c shows the pointcloud of the bush after performing the segmentation and post-processing.

4.3.3 Pointcloud post-processing

After segmenting the stems, each individual pointcloud is downsampled using a Voxel Grid filter of size 0.1 mm. Then, the pose of the pointclouds are transformed to the global frame (robot base) and merged by accumulating all the points. To make the cutting point localization process faster, the merged pointcloud is spatially subsampled and the noise removed. All the points that do not have at least 20 neighbors within a range of 0.5 mm are considered noise.

4.3.4 Cutting points localization

Here, the criterion to find the cutting point is based on the height of the plant. Those stems with height above h cm will become candidates to be pruned. This height varies depending on the type of the rose. The cutting points are found by creating a virtual plane π_h at h cm above the ground and finding all the points p_i in the pointcloud $P \in \mathbb{R}^3$

[4]Under our setup (see Section 4.4).

that are close to the plane (4.1).

$$P_h = \{p_i | dist(p_i, \pi_h) < 1.5\,cm\} \tag{4.1}$$

where:

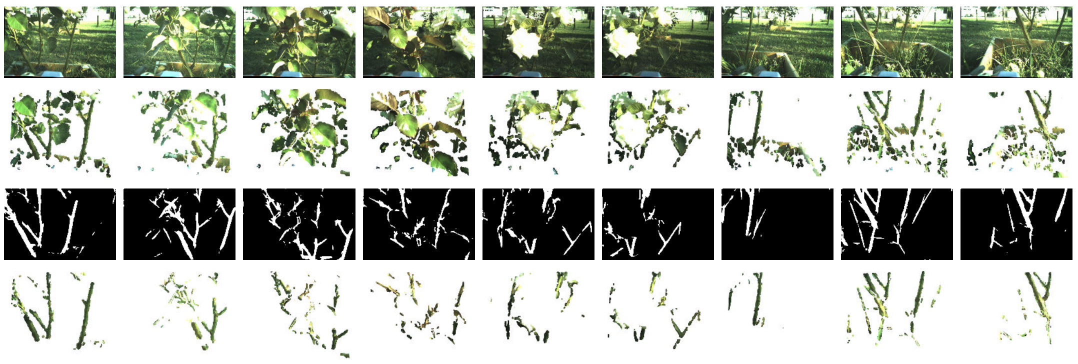
$$dist(p_i, \pi_h) = \left| \frac{\pi_h}{||\pi_h||} \cdot \left(p_i - p_{plane}\right) \right| \tag{4.2}$$

Equation 4.1 outputs all the points p_i in the bush that are 1.5 cm or closer to the desired height h and stores it in the set P_h. Equation 4.2 is the distance from plane to point where p_{plane} is any point in the plane.

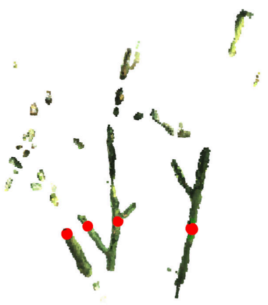
The points in P_h are clustered to find the parts of the plant where the cutting points are located. These points are clustered based on distance using DBSCAN (Density Based Spatial Clustering of Applications with Noise) [203]. DBSCAN is a density based clustering method, known to work well with groups that are closely packed together and where the number of clusters is not known beforehand. A group of points is considered a cluster if the distance between them is less than a threshold $d_{cluster}$ and the quantity of points existing below this threshold is higher than a minimum min_p. A minimum number of points min_p of 40 and a $d_{cluster} = 0.8$ cm was used in our configuration (these values were found empirically).

A simple approach would choose the center of gravity of those clusters as cutting points, however, it might lead to false positives. This could happen for two main reasons. First, the points inside a cluster might belong to a branching part of the plant. Second, two stems that were really close might get clustered together. In both cases, the cutting point would be wrongly located between the middle of two stems. This problem is tackled by dividing the clusters into horizontal slices (7 in our implementation) and clustering each slice with a distance threshold of $d_{cluster} = 0.5$ cm and minimum number of points of 5. In this way, thinner branches in the cluster can be found. This second clustering will be called sub-cluster. The cutting points are found by evaluating these sub-clusters in the following way:

- If at least four of the upper slices have two sub-clusters, this means that the cluster is located in a branching section of the plant, therefore, two cutting points are generated. These two cutting points are the center of gravity of the two sub-clusters that have the farthest distance between each other (yellow slice in Fig. 4.4a). This criteria aims to have two points in the cluster that are far from each other so the cutting tool does not get stuck between both stems.


(a) Scanning trajectory at 9 stopping poses. From top to bottom, color images captured by the left camera, pointclouds of the rose bush, branch segmentation network output, and the final masked pointclouds.

(b) Merged pointcloud without branch segmentation.


(c) Merged pointcloud after segmenting the stems with cutting points localized.

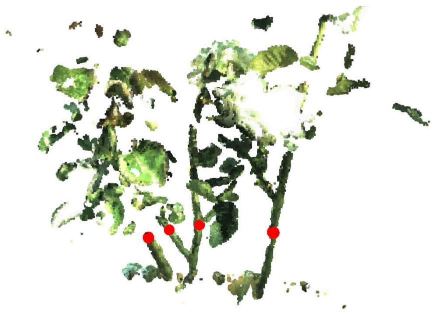
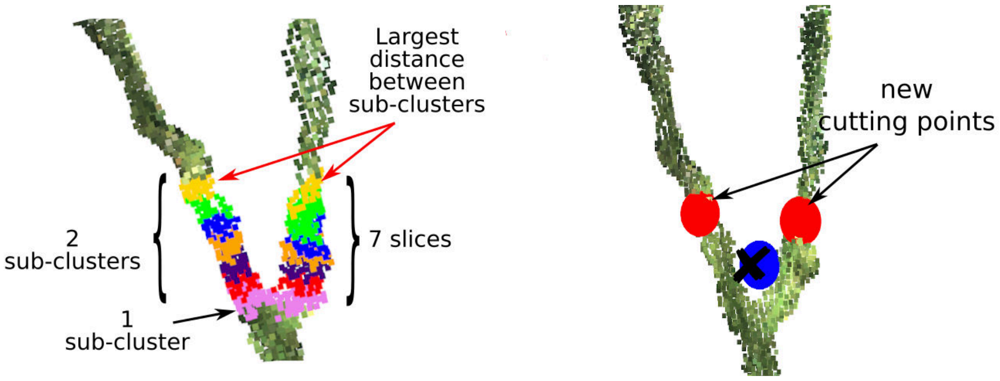

(d) Cutting points on the original pointcloud of the bush.

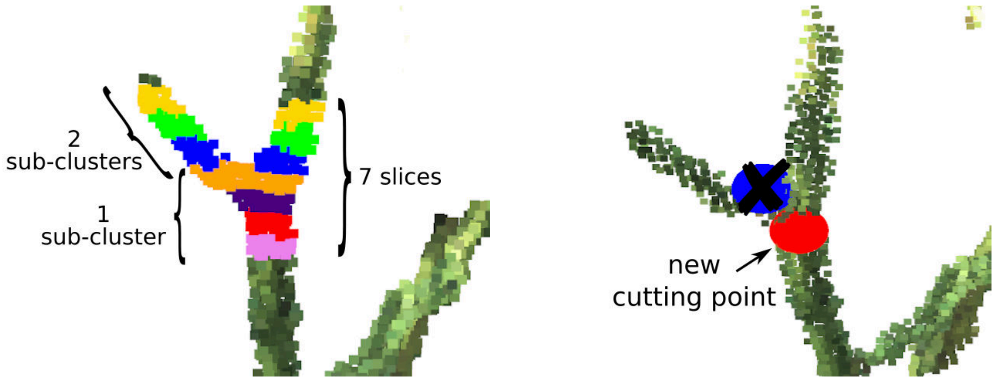
Figure 4.3: Scanning trajectory and pointcloud of the rose bush with cutting points (red dots).

- If all the upper slices are divided but there are at least 4 slices at the bottom part of the cluster that have not been divided, the center of the bottom slice is chosen as a cutting point (pink slice in Fig. 4.4b).

- If all the slices have one sub-cluster, the center of gravity of the middle slice is chosen as cutting point.



(a)



(b)

Figure 4.4: Clustering and slicing approach to find cutting points. The first column shows the slices (colored) found inside a cluster. The second column shows the resulting cutting points after evaluating the slices. The red circles indicate the chosen cutting points and the blue circles the center of gravity of the cluster. (a) Shows the case where 6 out of 7 slices in the cluster can be split in two, generating, in this way, two cutting points. (b) Shows the case where the 4 bottom slices of the cluster are not divided, which creates a cutting point in the bottom slice.

Note that because DBSCAN relies on the distance between points to form a cluster, it usually groups the points of a thorn as part of the branch it belongs to; this is because the size of the thorns are small (~ 1.5 cm) and do not tend to grow far from the branch.

4.3.5 Visual servoing and navigation

Once the cutting points are found, their positions are stored and the robot is sent to each location one by one from closest to farthest. For each stored point, the following pipeline is executed. 1) The robot starts at home position, which is located at height h and points towards the rose bush, as shown in Figure 4.1. 2) The robot navigates to the current target using our visual servoing method, once the robot reaches the target, it sends the signal to activate the cutter. 3) The robot returns to the home position and the process is repeated for the next target point. This section will give an in-depth description of our visual servoing approach.

The robot starts the servoing by rotating the end-effector 45° for sideways cut following gardening rules. The arm navigates to the first target. While it navigates to the current cutting goal p_{goal}, the visual servoing looks for a new cutting point candidate p_{can} in a neighborhood of radius 1.5 cm around p_{goal}. If there is any candidate in this radius, the goal gets updated using a convex combination (4.3) between the current goal position and the new goal. The convex combination is weighted by a "blending" factor $\alpha \in (0,1]$ with $\alpha = 0.1$ in our implementation. This combination guarantees a smoother update of p_{goal} by avoiding fast changes in position between consecutive times $(t, t+1)$, where $p_{goal}(t+1)$ is the new goal and $p_{goal}(t)$ is the current goal.

$$p_{goal}(t+1) = \alpha p_{can}(t) + (1-\alpha) p_{goal}(t) \tag{4.3}$$

The neighbors of the cutting goal are found by running a process in parallel which captures the position of the arm and pointcloud at the current time t, and outputs the positions of the cutting points on the scene using the methods from Section 4.3.3, 4.3.2 and 4.3.4 (stem detection, Pointcloud post-processing, Cutting points localization). This process can be better appreciated in Fig. 4.5.

The navigation of the arm, from the start position to a cutting point, is performed using proportional velocity control. ROS MoveIt! software [204] is used to find the inverse of the Jacobian $J^{\dagger}$ to obtain the joint difference Δq from the distance ΔX; the ΔX is the distance between the end-effector of the robot and the target p_{goal}. For the approach, a proportional controller is enough to have a smooth trajectory. The proportional value K is not a constant but a dynamic value that changes based on ΔX because the robot should "decelerate" when it gets close to the cutting location and should increase the velocity when the end-effector is far from the target. However, in practice, it is not desirable that only the distance between the end-effector and the

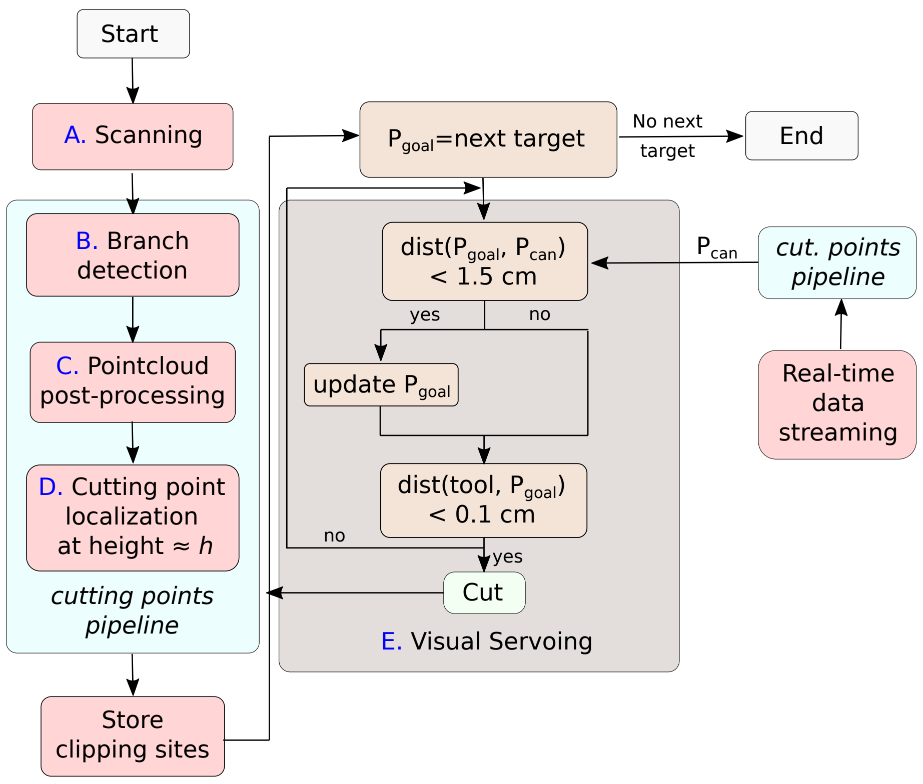


Figure 4.5: Pipeline for cutting roses.

target controls the value of K, specially because if ΔX is big, $\dot{q}$ will have an undesirable high speed. Thus, a maximum velocity must be set to avoid this. Similarly, a lower bound is set to avoid the velocity becoming 0 when the end-effector gets really close to the target stem. Equation 4.4 shows how the velocity of a joint q_i is calculated.

$$\dot{q}_i = \begin{cases} 10[\frac{deg}{s}] & if\ K\Delta q_i > 10[\frac{deg}{s}] \\ 3[\frac{deg}{s}] & if\ K\Delta q_i < 3[\frac{deg}{s}] \\ K\Delta q_i & otherwise \end{cases} \tag{4.4}$$

4.4 Experiments

The pruning pipeline was tested using several rose bushes. This includes rose stems placed inside a pot and a real bush in a garden. The thickness of the stems ranged between 0.6 to 1.0 cm. All the tests were performed outdoors in a garden, meaning that the system was tested in an uncontrolled environment. A sample of the plants used in the evaluation is shown in Fig. 4.6 and the result of the rose pruning in Fig.4.7.

The system was tested using a Razer Blade 14 i7 with 8 cores and a GTX 1060 Nvidia GPU. The connection between the software and hardware was done through

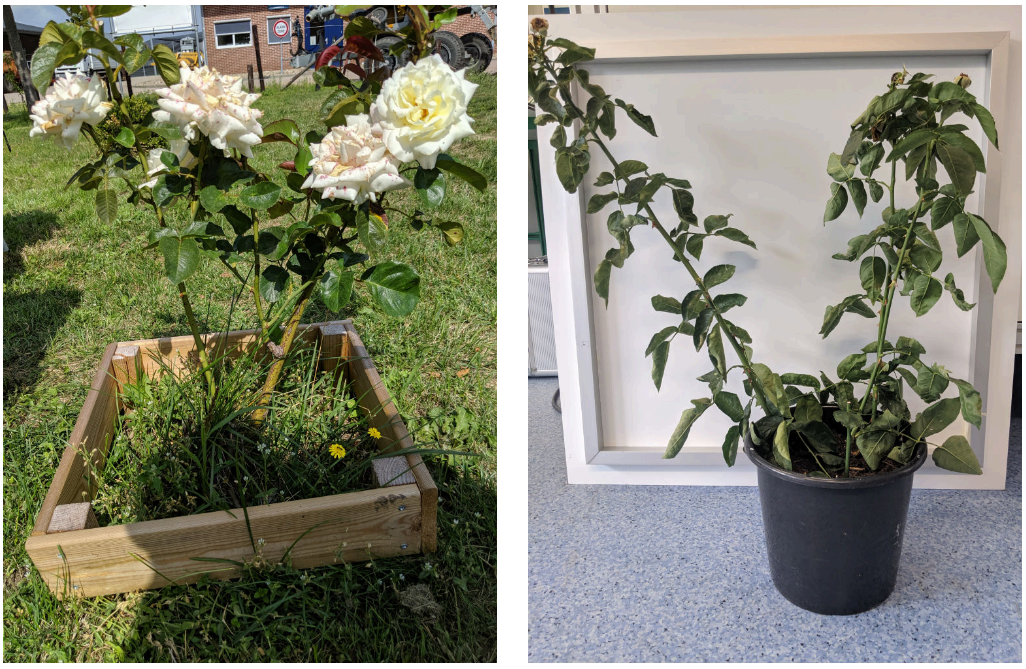


Figure 4.6: A sample of the rose bushes used in the evaluation.

the Robot Operating System (ROS) [169].

Fold	**Precision**	**Recall**	F_1
0	0.8482	0.8228	0.8353
1	0.8120	0.8224	0.8171
2	0.8166	0.8265	0.8215
Macro Avg.	0.8256 ± 0.020	0.8239 ± 0.002	0.8246 ± 0.010

Table 4.1: Pixel-wise branch detector results of the different folds and macro averages.

4.4.1 Branch segmentation evaluation

The branch segmentation CNN was trained and evaluated using our new dataset of rose bushes which consist of 1360 manually labelled images, each image with a size of 752×480 px. A sample of the dataset can be seen in Fig. 4.8.

The architecture of the branch segmentation CNN is shown in Table 4.2. The performance of the network was evaluated using k-fold cross validation with $k = 3$, F_1 score for each fold and macro F_1 for the whole network. Table 4.1 and Fig. 4.3a show the results of the branch segmentation.

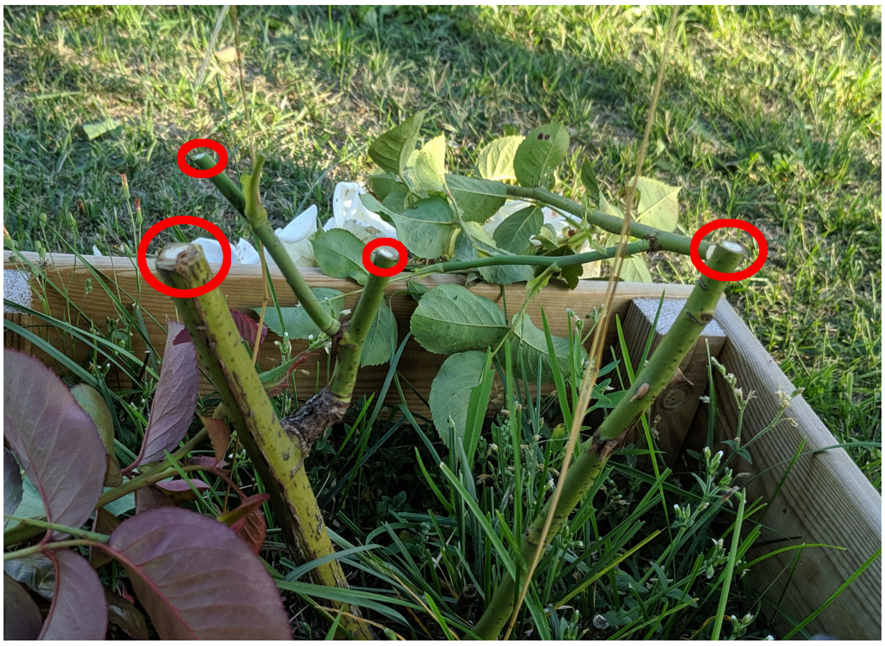


Figure 4.7: Rose bush after pruning. Figure 4.6 shows the plant before being cut.

Figure 4.8: A sample of the dataset collected to train the network.

4.4.2 Evaluation of cutting point detection from scanning

The accuracy of the scanning was measured by comparing the total number of targets found after scanning a bush and processing it (Section 4.3.3, 4.3.2 and 4.3.4) and the real number of targets. This value is also compared against the number of targets found by considering only the data captured by a single pose. The number of cutting points found by a single view was obtained by counting the total cutting locations found for each view and averaging them together. The real targets are the stems that exceed the desired cutting height h. The heights used for the evaluation are 10, 15, 20, 25, and 30 cm. The ground truth height is determined as follows. The robotic arm is mounted on top of a mobile robot [115], whose dimensions and position (height) with respect to the ground is known. Therefore, to find the height of the plant, we only have to transform the points of the stems from the camera frame to the mobile robot origin.

The data is taken by moving the robotic arm in a square trajectory of 20 cm and stopping at 18 locations. The data was captured using 19 different bushes, each bush

Input image size:	480×320 px
Number of layers:	4+4
Filters per layer:	128
Kernel size:	5×5
Normalization type:	Standard
Data augmentation:	10 %

Table 4.2: Branch segmentation CNN architecture.

containing 3 to 4 cutting locations. The total number of cutting locations was 60. Table 4.3 shows that the fused and segmented pointcloud, after scanning the bush, led to a detection accuracy of 90% (true positive rate). However, if only one single pose (one single view) is used to find the cutting locations, the accuracy drops to 30%. The cutting points that were difficult to find were usually those that belong to the stems at the back of the bush. It was caused mostly by the dense population of stems and leaves covering them.

	Detected cutting points	Detection accuracy
Fused views from scanning	**54**	**0.90**
Single view (average)	17.89	0.30

Table 4.3: Cutting point detection accuracy of targets found (out of 60) after scanning and post-processing the pointcloud, and the average cutting points found by a single view.

4.4.3 Visual servoing navigation quality

The quality of the visual servoing system was evaluated by letting the robot navigate towards the cutting locations after the scanning. This result was compared against a global planner. The global planner gives the location of the targets to the planner and moves the manipulator towards them without updating the target position. A navigation is considered successful if the robot reaches the target location and the target stem gets into the cutter. The total number of cutting locations found by the robot after scanning the bush were 54 out of 60 real cutting locations. Therefore, the evaluation of the visual servoing process was done using only the 54 locations found by the scanning process. Table 4.4 shows that a global planner is not sufficient to drive the end-effector to the target location. In practice, the stem randomly moves up to 2 cm sideways due to the combined effect of external forces (like wind) and the interaction of the end-effector with other connected parts of the bush. This makes the end-effector usually end up on one side of the stem. On the other hand, our visual servoing approach is robust enough to make the robot navigate and reach the cutting points 94% of the time under dynamic conditions.

	Reached points	Accuracy
Visual servoing	**51**	**0.94**
global planner	27	0.5

Table 4.4: Visual servoing and global planner results out of 54 detected cutting targets.

4.5 Conclusions

The presented approach was designed to effectively cut rose bushes in a garden in an unconstrained outdoor environment with dynamic targets. The experimental evaluation shows that the neural network is capable of segmenting the stems of rose bushes from the background, even when the background and the stems have similar color. This result also demonstrates that the large dataset introduced in the chapter can indeed be used to successfully train a neural network to segment branches of different type of roses.

The proposed target localization approach, which consists of the combined process of stem detection, clustering and pointcloud merging can successfully find the cutting points 90% of the times. These targets are found even when they are occluded by other stems or leaves. This approach also proves to be robust but fast enough to be used by the visual servoing to update the target location on the fly.

The visual feedback is a key element to navigate in a garden where the wind can change the position of the stems, thus change the location of a target. The proposed visual servoing performs a good navigation with an accuracy of 94%. The combination of these steps results in a pipeline capable of finding cutting points in stems that are occluded by other stems or leaves and navigating towards them successfully in ~ 12 secs with an average initial distance between the center of the cutting tool and a target stem of 0.6 m.

Scanning the rose bush in a square path is a simple yet effective to capture the structure of the plant. Different scanning methods can be done to improve the scanned bush model, like having different poses instead of a square shape or scanning the bush by navigating around it, however this will lead to further problems like localization and drifting.

4.5.1 Limitations and future work

This work is a significant step towards an automated robotic rose pruning system with visual input as close-loop feedback able to work in a real environment. Currently, our visual servoing accuracy is 94% when the robot is in front of the rose bush. Although this is an acceptable assumption for rose bushes, other types of plants are denser and wider. Thus, our method might not find all the cutting points. To make our approach more general, a possible future work would be to scan the plants from different sides and fuse the views.

As mentioned at the beginning of the chapter, the 3D morphology estimator was not used. Thus, our method was only tested for one gardening rule: cutting the branches at a certain height. A good extension of this work would be to combine both methods to deal with more complex gardening rules such as cutting crossing inward branches, dead branches, thin branches, and finding branches above eye buds.

Chapter 5

Multi-task Learning for Semantic Segmentation and Disparity Estimation

Previous chapters explore our contribution to scene understanding using classical approaches. This chapter demonstrates how the disparity and segmentation predictions can be inferred using multi-task learning in an end-to-end process and obtain a competitive performance compared to the state-of-the-art of both task while using less parameters.

5.1 Introduction

Disparity estimation and semantic segmentation are fundamental problems in computer vision. The goal of disparity estimation is to find the pixel correspondences from a pair of images. On the other hand, semantic segmentation assigns class labels to each pixel in the image. Both areas, individually, have been intensively investigated, and lately, deep convolutional neural networks (CNN) have been the dominant solution for both [205, 206].

These two methods are heavily used in scene understanding, autonomous driving, and robotics. However, executing the state-of-the-art methods of both areas together [207, 208], essentially doubles the computation load, which may be prohibitive. Therefore, a method that combines both tasks into a single model can potentially reduce computation, and thus allow it to be embedded in portable devices or executed in real-

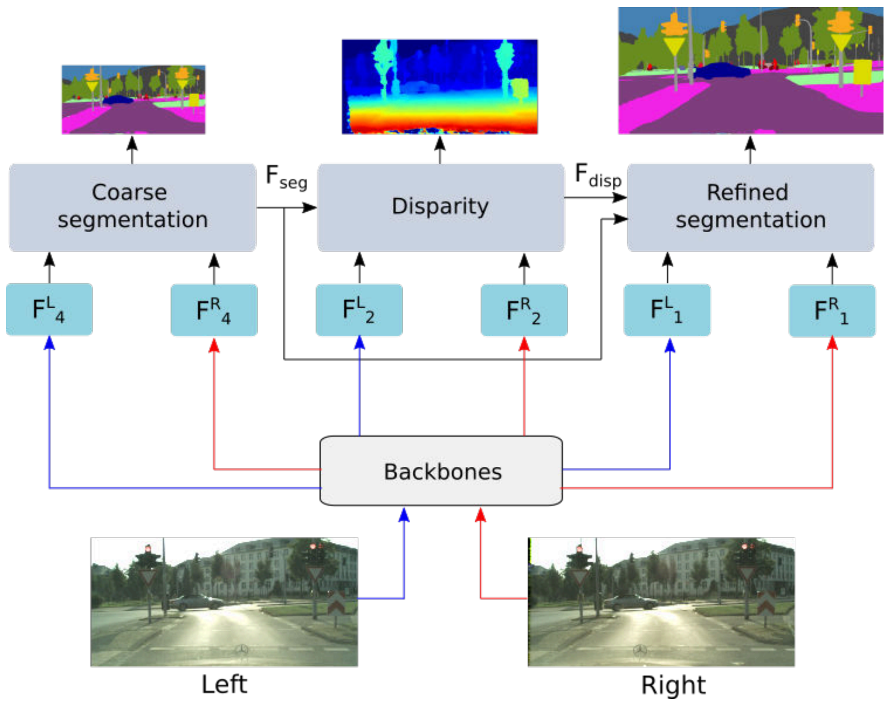


Figure 5.1: Network overview. Stereo pair features are extracted from the backbone and used for a multi-scale, multi-task prediction. Task-specific features F_{seg} and F_{disp} are shared among the decoders to progressively refine each task.

time[1]. This method is also called multi-task learning [62], an area that is least explored in the joint segmentation and disparity estimation field [8, 209].

In the literature, one can find many methods for disparity estimation and segmentation [205, 206]. The general modern is an encoder-decoder CNN, where the encoder extracts feature information by encoding the dimension of the input image, and the decoder decodes these descriptions to predict the disparity map or the semantic segmentation. For disparity estimation, two main approaches exist. The first approach treats disparity estimation as a regression problem, using solely 2D convolutions [48]. The second approach obtains a cost volume by extracting 3D features and discretizing the disparity [49, 50]. For semantic segmentation, the standard approach is to use a pre-trained network as a backbone (encoder) and add specialized layers (decoder) to obtain the segmentation [35]. These layers extract multi-scale information from the input image to improve the segmentation [208, 210], or to find some context using attention layers [211].

Compared to the extensive literature that approaches these tasks separately, there are few works about solving them together using data-driven techniques. This is be-

[1]This is not explored in the chapter because is out of the scope.

cause segmenting an image and finding the disparity map are complex tasks per se and require a lot of training examples to learn meaningful representations of each task with an end-to-end approach. Another issue is the lack of datasets that have both ground truths and stereo pairs; Cityscapes[2] [212], KITTI [213] and the Trim-Bot Garden Dataset [214] are among the few existing datasets. Because of this, most of the multi-task methods only estimate a monocular depth which is an ill-posed problem from the geometry perspective [79]. Among the approaches that solve both tasks jointly, some focus on using individual decoders for each task [209, 8, 77] and others on improving the energy function [209]. Similar to disparity and semantic segmentation multi-task learning, other works use shared encoders. For example, [215] uses a pre-trained semantic segmentation network to leverage semantic information and improve monocular depth. On the contrary, [216] uses a single decoder that can predict either the semantic segmentation or the depth of an image given as input a class token.

Different from the methods above, we argue that exploiting the feature information from different tasks and combining them progressively improves the learning of semantic segmentation and disparity using a stereo input. More specifically, we propose to use a backbone as an encoder to extract feature information at different scales from the left and right images and have 3 specialized branches as decoders. The first one extracts a coarse segmentation using high-level features from the backbone. The second branch outputs a disparity map by extracting the correlation between the left and right features and combining them with intermediate features from the coarse segmentation branch. The third branch obtains a refined segmentation by combining low-level features from the backbone and the coarse segmentation and disparity branches. Here, an attention mechanism is used, so the network can select the most useful features from the coarse segmentation branch and disparity branch [217]. This progressive multi-scale learning allows the segmentation and disparity information to refine each other. Low-level information from the input image was also extracted by using a convolution with a larger field of view and concatenated to each branch before outputting the prediction. This allows the network to obtain a prediction with almost the same dimension as the input. The main contributions of this chapter are:

- An end-to-end method that successfully learns the semantic segmentation and disparity map together from a stereo pair. To achieve this, it learns task-specific features that are shared along with the two tasks progressively.

[2]The disparity ground truth is obtain from SGM stereo.

- It improves the state-of-the-art for joint disparity and semantic segmentation multi-task learning with competitive results against other methods that solve each task individually.
- A network capable of solving both tasks jointly using less than $1/3$ of the parameters that the previous works use to solve only one task. This reduced number of parameters is useful for systems that have limited resources like the areas of autonomous driving and robotics.

5.2 Methodology

In this section, we introduce the proposed progressive multi-task architecture and elaborate on how it can use feature information from one task to improve other tasks using a coarse-to-fine structure with a stereo input.

5.2.1 Network architecture

Before describing the core parts of the network, an overview and description of the layers will be given; a detailed description of the network architecture can be found in Appendix A. The proposed method consists of a backbone and a decoder with 3 specialized branches which can be seen in Figure 5.1. The first branch obtains a coarse segmentation. It outputs task-specific features F_{seg}, which are used as input for the second branch. The second branch outputs the disparity of the stereo pair and its own task-specific features F_{disp}. The third branch uses these features to output a refined segmentation. The segmentation and disparity outputs are from the left image. Each convolution used in the network has a kernel size of 3×3 followed by a ReLU transformation and batch normalization unless stated otherwise. Following [50, 218], after concatenating any 2 or more features, they are processed with a series of top-down/bottom-up convolutions known as an hourglass (encoder-decoder) to learn more context information. The hourglass is composed of 3 convolutions and 3 deconvolutions with residual connections. The disparity and coarse segmentation branches have two encoder-decoder blocks. One is used to process the information obtained by the incoming features, either from the backbone or other branches; the output of this block is used as a task-specific feature. The second encoder-decoder processes the information further to obtain the final prediction. The segmentation branches use Cross-entropy and Lovasz [219] losses, while the disparity branch uses L1 loss.

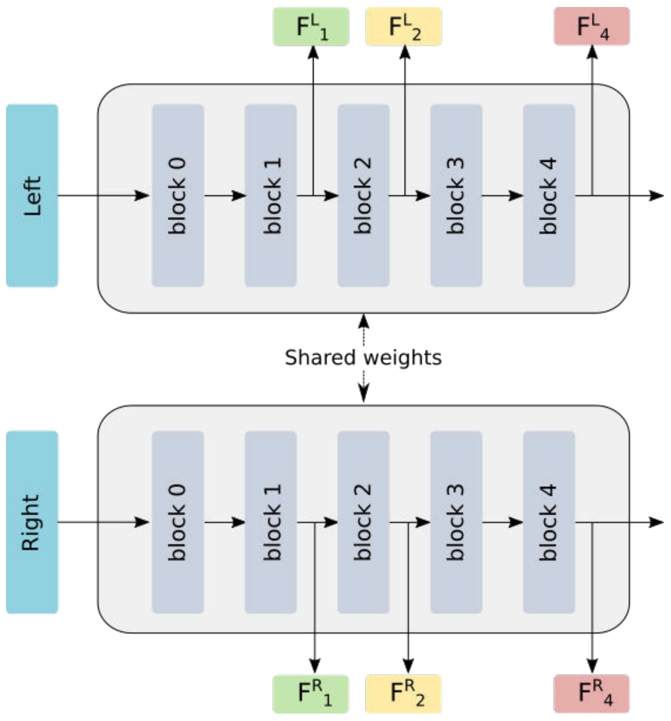


Figure 5.2: Backbone structure. Each block reduces the dimension of its input in half. The dimension of blocks 0, 1, 2, 3, 4 are $1/2$, $1/4$, $1/8$, $1/16$, $1/32$, respectively. The features F_1, F_2 and F_4 are used in the branches.

5.2.2 Backbone feature extraction

Two Densenet121 [34] with shared weights are used as backbones to obtain high-level and low-level features from the left and right images. The shared weights approach reduces the number of parameters of the network [220]. The backbones are pre-trained with ImageNet [34]. Each backbone is divided into 5 blocks to extract the features. Because the input is from stereo, the features from each view must be extracted. They will be denoted as F_i^L and F_i^R in the rest of the chapter, where i is the block from where it was extracted and L and R indicate if the feature belongs to the left or right image, as seen in Figure 5.2.

5.2.3 Coarse segmentation branch

The first branch extracts a coarse segmentation as seen in Figure 5.3, similar to [221]. Since this output is also optimized, it can be seen as an auxiliary loss [37]. The input of this branch is the concatenated feature maps F_4^L and F_4^R, followed by a sequence of convolutional layers. Block 4 from the backbone outputs features with $1/32$ of the size of the original image, causing a loss of detailed information. Following [222], to

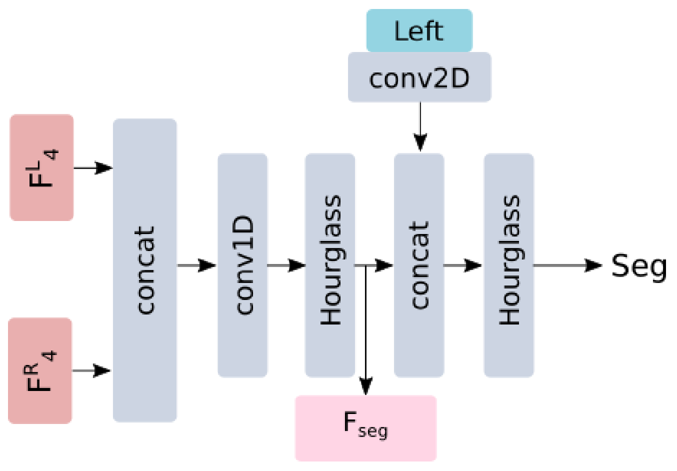


Figure 5.3: Coarse segmentation branch. It receives as input F_4^L and F_4^R. Before predicting the coarse segmentation, the left image is convolved and concatenated to add low-level information.

reduce this problem, low-level features are obtained by processing the left image with a 5×5 convolution (larger receptive field). Then, the output is concatenated with the rest of the features in the branch (Figure 5.3). The concatenated features are convolved once more to output the final segmentation. Softmax is used at each pixel to obtain the probability of each class. This branch extracts task-specific features F_{seg}, which will be concatenated with the disparity and refined segmentation branches. In this way, the network explicitly passes the features learned by this branch for progressive learning and refinement.

5.2.4 Disparity branch

The second branch computes the disparity map from the stereo images (Figure 5.4). The inputs of the branch are F_2^L and F_2^R. The correlation between these two features is computed using a correlation layer [48], which takes a block of the features from F_2^L and convolves it in a neighborhood window around F_2^R. Because the window size is limited, it does not capture correlations between distant features. Therefore, Spatial Pyramid Pooling (SPP) [223] is used on the two feature planes, before extracting their correlations, to obtain multi-scale information, and thus, find distant correlations. The depth of the correlation block is reduced with a 1×1 convolution. Then, it is concatenated with the features F_{seg}, which were computed by the coarse segmentation branch. In this way, the features learned by the segmentation are included to add more information for the disparity estimation. As the next step, low-level feature information is

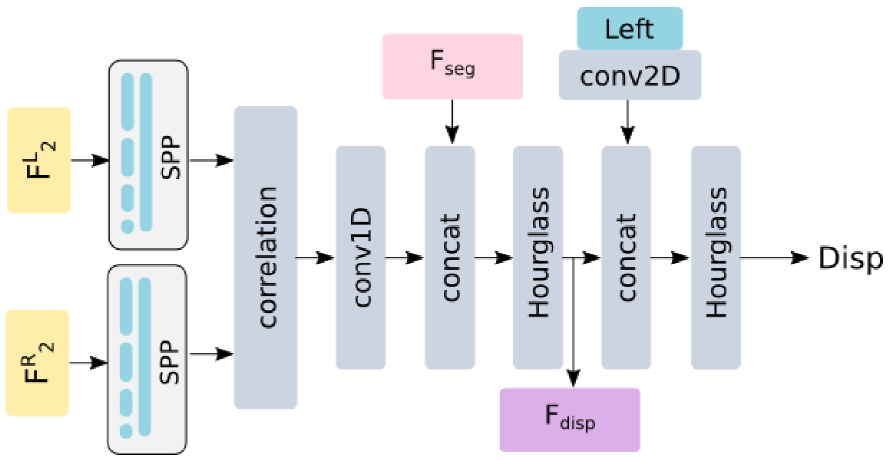


Figure 5.4: Disparity branch. It extracts multi-scale information from backbone blocks F_2^L and F_2^R using Spatial Pyramid Pooling (SPP). This information is processed further and concatenated with the features from the coarse segmentation.

added to the network using the same approach as the previous branch. The output is followed by an encoder-decoder block to obtain the disparity. Similar to the previous decoder, this branch outputs an intermediate task-specific feature F_{disp}, which is used by the refined segmentation branch.

5.2.5 Refined segmentation branch

The third decoder computes the refined segmentation (Figure 5.5). Here, SPP is used on the backbone features F_1^L and F_1^R, and their outputs are concatenated. The concatenation is followed by an encoder-decoder block and then split in two. One part is concatenated with F_{seg} and the other with F_{disp}. These features carry task-specific and high-level information from previous branches. We use a 1×1 convolution with sigmoid activation to get the attention map. The attention focuses on selecting important features from each task (see Figure 5.6). The attention is multiplied with the output of the convolved features F_1^L and F_4^R, which carries low-level information. An encoder-decoder block with an output dimension equal to the number of labels, followed by a softmax, is used to obtain the refined segmentation.

5.3 Experiments

The proposed method is evaluated using Cityscapes [212] and the TrimBot Garden [214] datasets. Some samples can be found in Figure 5.7 and 5.9 respectively. The

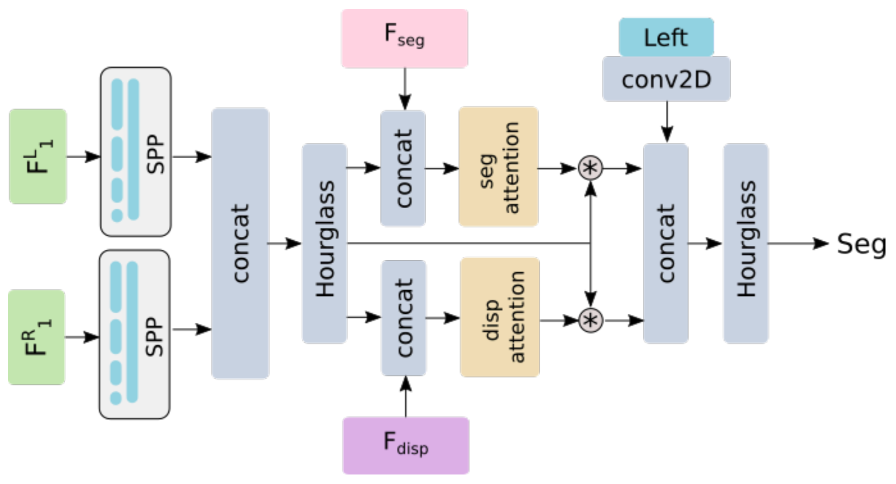


Figure 5.5: Refined segmentation branch. It concatenates the multi-scale feature information from F_1^L and F_1^R. The features from the previous segmentation (branch 1) and disparity (branch 2) are also added to the network using an attention layer.

datasets are described below as well as the details of the model implementation and training process (see also Appendix A). In Section 5.3.3, the results are compared with those of the state of the art. An ablation study is then carried out to analyze the parts of the model separately. Finally, the efficiency and effectiveness of the different methods are compared jointly.

5.3.1 Datasets

Cityscapes [212]: It is an urban scene understanding dataset that contains 19 classes such as person, wall, car truck, and bus. Some examples can be found in Figure 5.7. The classes are grouped into 7 categories: flat, nature, object, sky, construction, human, and vehicle. The disparity ground truth is a sparse map obtained by SGM stereo. It has 5,000 high resolution 2048×1024 px images, from which 2,975 are used for training, 500 for validation, and 1,525 for testing. The dataset does not provide the semantic segmentation ground truth for the test set. Therefore, to evaluate the performance of the network using this set of images, the segmented outputs needs to be uploaded to the official Cityscapes website. In the experiments, the mean of class-wise Intersection over Union (mIoU) is used as the evaluation metric for segmentation and D-1 error for disparity estimation. The latter calculates the percentage of pixelwise disparity errors below a threshold, which is usually set to 3 pixels of difference between the ground truth and the estimated disparity [209].

TrimBot Garden [214]: It consists of images from a synthetic garden under 4

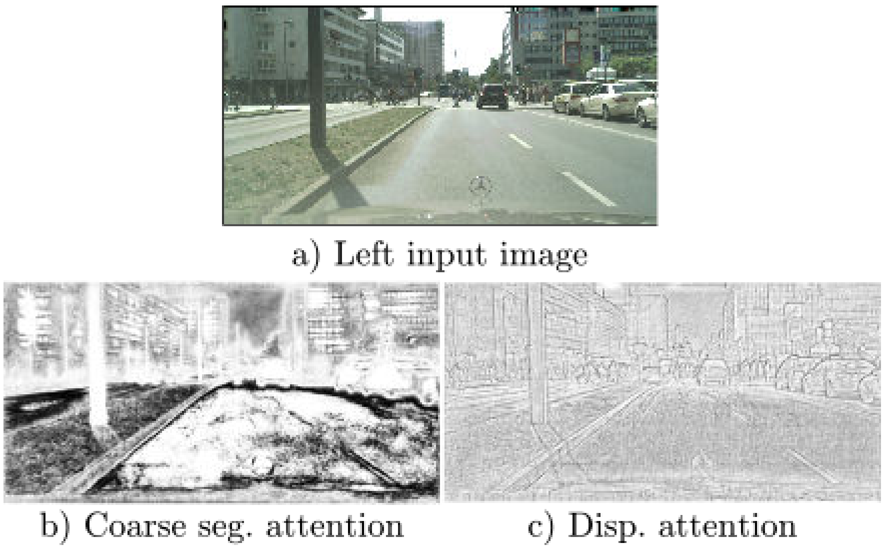


Figure 5.6: Attention map output. The intensity of the attention is in grayscale, with white being the areas with the greatest attention. The attention obtained for the coarse segmentation focuses on the inner part of each segment, where the segmentation is more confident. On the other hand, the attention for the disparity branch uses the features at the region edges, where the disparity calculation is most certain.

weather conditions, as seen in Figure 5.9. There are 9 pixel-wise segmented classes and their disparity maps. The dataset consists of 10k stereo pairs for training and 2.5k images for testing. Each image has a resolution of 640×480 px. Following the dataset metrics, pixel-wise accuracy was used to evaluate the semantic segmentation. The dataset evaluates the quality of the 3D reconstruction instead of finding the error of the 3D depth or disparity. Therefore, our disparity predictions were transformed into depth maps and then back-projected into the 3D space using the camera parameters provided by them. The dataset takes into consideration two other metrics, completeness, and 3D accuracy. The completeness is calculated by considering a predicted point as correct if it differs from the ground truth by $\leq 0.05m$, similar to D-1 error. The 3D accuracy is the distance d in meters, such that 90% of the reconstruction is within d of the ground truth mesh.

5.3.2 Implementation details

The implementation was built using the public library PyTorch [224]. Adam is used as optimizer with default hyperparameters: learning rate 1e$-$3, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 1\text{e}{-8}$. The final loss consists of the sum of the losses of the 3 branches.

$\mathcal{L} = \mathcal{L}seg_{cr} + \mathcal{L}disp + \mathcal{L}seg_{ref}$. The training was done end-to-end and the gradients of each branch loss are propagated across the whole network. For data augmentation, we did random cropping of 512×512 px for Cityscapes and 480×480 px for the TrimBot dataset, and random resizing between 0.7 and 1.5 only for Cityscapes. We did not use random resizing in the TrimBot dataset because it provides many more (and more varied) training examples. We also adjusted the brightness, contrast, saturation, and added Gaussian blur with a standard deviation that varies between 0.25 and 1.15. To handle the class imbalance, class uniform sampling [225] was used. For our experiments, a maximum of 100K iterations was set for Cityscapes and 70K iterations for the TrimBot Garden dataset. To evaluate the test set of Cityscapes, the training and validation sets were used for training. Due to limited physical memory on the GPU cards, the batch size was set to 16 during training. We also synchronized the batch normalization layer of all the GPUs for stable learning performance. Appendix A shows a more extensive evaluation of the hyper-parameters.

5.3.3 Model analysis

Cityscapes: The proposed network uses as the main backbone Densenet121 [34], a lighter backbone with 8.0M parameters compared to the usual DeepLab v3 used for semantic segmentation and multi-task learning, which has around 64.2M parameters [225]. After adding the specialized decoders for the disparity and semantic segmentation, the number of parameters of our network remains lower than the DeepLab v3 backbone with only 18.0M parameters. To show the capability of our progressive decoders, the backbone was not modified. Therefore, the feature at the coarsest level is $1/32$ smaller than the input size, compared with DeepLab v3 backbone which has a bigger receptive field of $1/16$. An example of the segmentation and disparity estimation of our network can be seen in Figure 5.7.

Table 5.1 compares our method with the state of the art in the Cityscapes ranking. For the comparison, the work with the highest score in segmentation, Panoptic-DeepLab [210], was considered, as well as the multi-task approach with the highest score for both tasks from Kendall *et al.* [209], which will be called MTU in our experiments. Given that both networks use Deeplab as backbone, two versions of Deeplab v3 using Resnet-101 and Resnet-50 were included as their base networks. The previous methods use deeper and heavier networks, which gives a boost to their performance. Therefore, we also compare our network with the method that achieves the

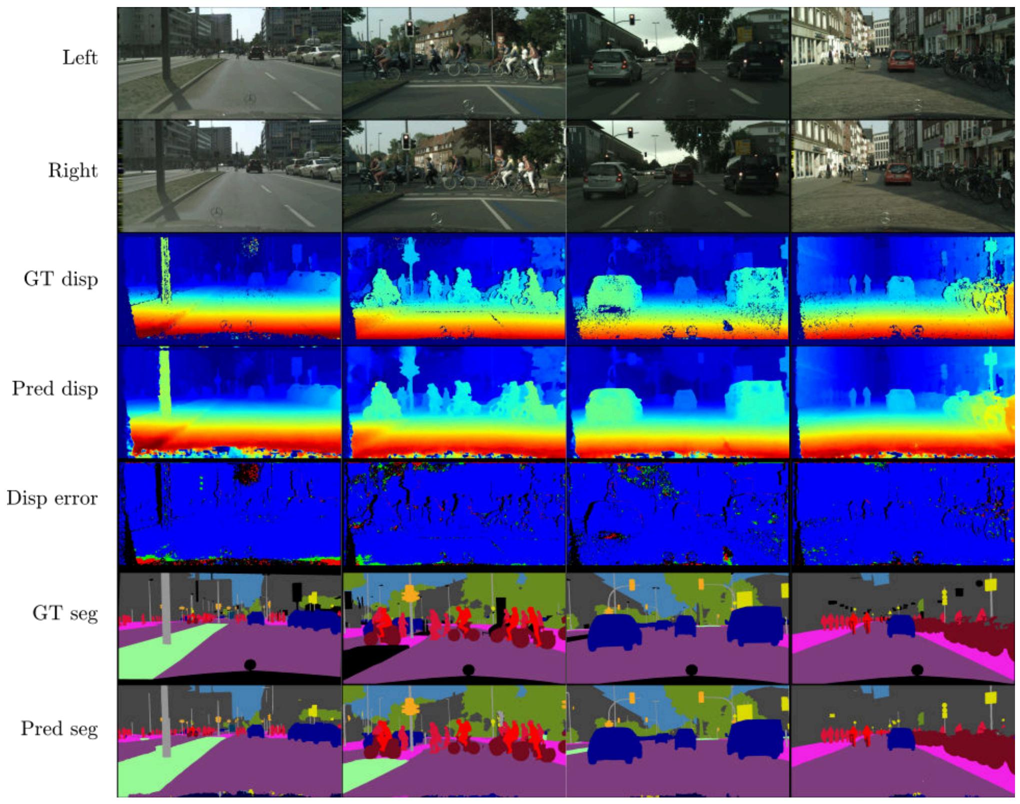


Figure 5.7: Cityscapes results on the validation set. The stereo pair input is shown in the first two rows. The ground truth and predicted disparities are shown in row 3 and 4. The error of the predicted disparity (row 5) encodes the error e in 3 colors: blue ($e < 3$), green ($3 <= e < 6$), red ($e \geq 6$). The black color in the disparity and segmentation images means that the ground truth has no disparity or has no class assigned to that pixel, respectively.

best segmentation using the DenseNet architecture as backbone, which is Ladder-style DenseNets [226].

Compared to MTU [209], our network gets lower disparity error, better IoU per category, and is only 2.5% worse on IoU per class using less than a third of the parameters. To assess whether this difference is due to the backbone, training process, or loss function, the following experiments were done (see Table 5.2): 1) We changed the MTU backbone for the one used by our method (Densenet121), keeping its decoder and its uncertainty loss, and 2) We used their loss with our architecture. The same learning process described in Section 5.3.2 was used for the experiments. The results show that our implementation significantly outperforms the segmentation and disparity obtained by MTU when it is evaluated and trained under the same conditions. The results were also worse when our network used their loss function. Therefore, we believe that the 2.5% improvement in the IoU per class obtained in Table 5.1 can be

Method	Backbone	Train input size	Params	Seg mIoU class		Seg mIoU category	Disp RMSE
				Eval	Test	Test	Test
Multi-task methods							
Ours	Densenet 121	512×512	**18.0M**	77.6	76.0	**91.0**	**3.28**
MTU [209]	Deeplab v3	512×512	64.2M	-	78.5	89.9	5.88
Semantic segmentation methods							
Panoptic [210]	Deeplab v3	2049×1025	46.7M	**81.5**	**84.5**	**92.9**	N/A
Ladder-style [226]	Densenet 121	2048×1024	**8.2M**	62.32	-	-	N/A
Ladder-style [226]	Densenet 169	2048×1024	15.6M	75.75	74.3	89.7	N/A
Deeplabv3 [41]	Xception-71	513×513	46.7M	79.55	82.1	-	N/A
Deeplabv3 [225]	Resnet 101	768×768	64.2M	79.2	-	-	N/A
Deeplabv3 [225]	Resnet 50	768×768	45.1M	77.8	-	-	N/A

Table 5.1: Cityscapes Dataset comparison table. The reported results are the class and category segmentation IoU, and disparity error.

Method	Backbone	Loss	Seg	Disp
			mIoU class	D-1 error
Ours	DN121	Ours	**77.6**	**0.040**
MTU	DN121	MTU	59.6	0.347
Ours	DN121	MTU	72.2	0.067

Table 5.2: Comparison with MTU [209] on Cityscapes evaluation set using the same backbone as ours. The comparison shows that the good performance of MTU from table 5.1 is due to the usage of a bigger backbone. If MTU is trained using the same backbone as ours their performance decreases. The table also reports the result of our network when trained with the loss proposed by MTU. This comparison supports the claim that, under similar conditions, our method performs better.

exclusively attributed to the use of a backbone with many more parameters[3]. The improvement provided by our method, regardless of the use of a specific backbone, can also be observed if our results are compared with those obtained with the approaches that use DenseNet (such as Ladder-style). Only the methods that use a backbone with many more parameters and larger images achieve an improvement in their results, such as Panoptic or Deeplab v3. However, this greater number of parameters and image sizes leads to less efficient solutions, and in these cases, to solve only a single task.

TrimBot Garden: For these experiments, the same architecture and training process were used as for Cityscapes. The results are compared with the best-performing methods for semantic segmentation and depth estimation, DTIS [227] and HAB [214], respectively. DTIS uses a network similar to FuseNet [228], which has as input an RGB image and a depth map, and outputs both the semantic segmentation and a refined version of the depth map. HAB uses ELAS stereo [229] to produce a dense point cloud, and DeepLab v3 [36] to obtain the segmentation. The resulting point cloud is

[3]We would like to demonstrate this claim directly but we do not have access to the necessary GPU resources.

denoised with class-specific filters based on the 3D geometry.

Table 5.3 shows that our method obtains a segmentation accuracy similar to DTIS and outperforms HAB. To evaluate the disparity output of our method with the metrics of the dataset, it was transformed into a depth map and back-projected into 3D as a point cloud. Disparity networks have the common problem of smoothing the edges. Therefore, the edges that have high gradient values were removed. Points that were labeled as sky and all the points that have ≤ 25 neighbors in a radius of $0.05m$ were also removed. The dataset evaluates the depth based on how well it reconstructs the 3D garden, using completeness and 3D accuracy as metrics. Our network achieves the best score in both. It is 3.8% more complete than the best result and the reconstruction accuracy is almost $0.01m$ better. It is important to highlight that, unlike HAB, our method obtains both results at the same time without applying any heavy point cloud filtering or using 3D features. Compared to DTIS, our network uses 10 times fewer parameters and it does not receive any previous disparity map as input. Figure 5.8 shows the 3D reconstruction obtained for the entire garden and Figure 5.9 the segmentation and disparity results from different views. The plot was obtained using the official code of the TrimBot dataset **??**.

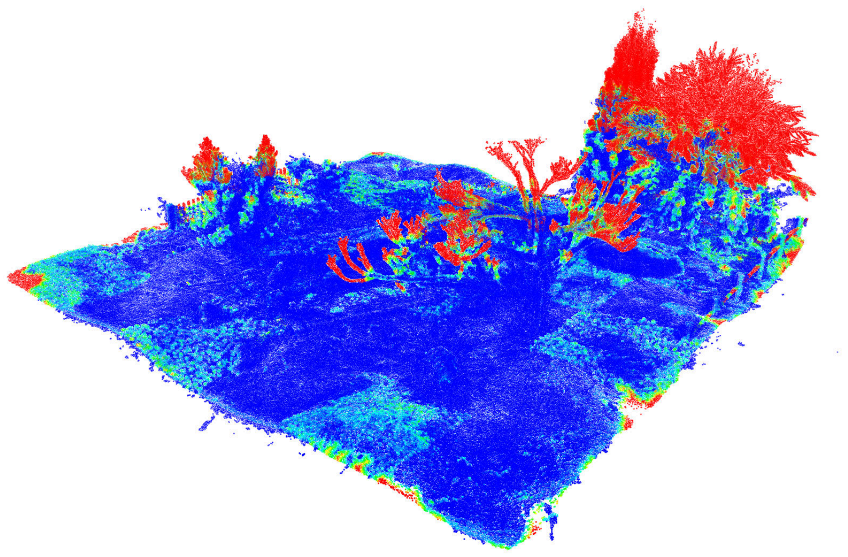


Figure 5.8: 3D reconstruction. Distances (0-1 m): Cold colors indicate well-reconstructed segments. Hot colors indicate missing parts (completeness).

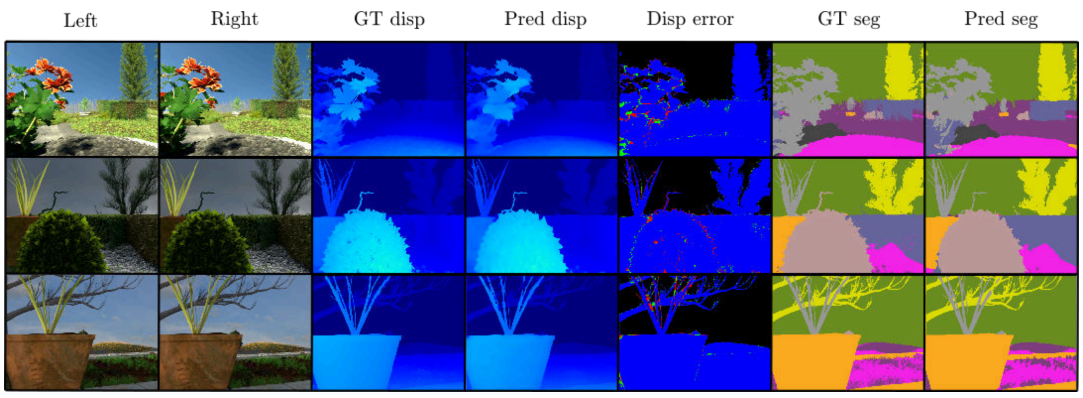


Figure 5.9: TrimBot Garden predictions on the test set. The color encoding used for the disparity predictions and errors are the same as those used for Cityscapes in Figure 5.7.

Method	Param.	Seg acc.	3D reconst.	
			Acc. (m)	Comp.
Ours	**18.0M**	**91.9**	**0.061**	**77.8**
DTIS [227]	207.8M	**91.9**	0.122	66.2
HAB [214]	64.2M	79.0†	0.069	74.0

Table 5.3: TrimBot dataset comparison. The segmentation accuracy (Seg acc.) and completeness (comp.) are in percentages. For the 3D accuracy, the lower the value the better. HAB† only reports the 3D segmentation accuracy.

5.3.4 Ablation study

Choosing output tasks

The number of outputs and the decision of having two outputs for the semantic segmentation and one for the disparity were made as follows. We trained multiple variants of our network. They were trained on the Cityscapes dataset using a maximum of 30K iterations and following the same configuration as described in Section 5.3.2. First, the network was trained to output one task at a time only once, as seen in the first two rows of Table 5.4. In these two experiments, the first task shares its features with the second task. As a second step, we tested if adding an extra output (refined output) of one task and receiving the task-specific features of the other outputs will obtain a better performance. Here, several configurations were tried where the tasks were output in a different order, as can be seen in the last four rows from Table 5.4. Due to limited physical memory on the GPU cards, we could not train the network with four outputs. The experiment shows that the best performance is obtained when the network outputs

Networks	Features from backbone			Coarse disparity	Final disparity	Coarse segmentation	Final segmentation
	F_4	F_2	F_1	D-1 error	D-1 error	IoU	IoU
Network 1	disp (R)	seg (R)		-	0.102	-	68.3
Network 2	seg (R)	disp (R)		-	0.094	-	67.8
Network 3	disp (C)	seg (R)	disp (R)	0.093	0.078	-	70.0
Network 4	disp (C)	disp (R)	seg (R)	0.093	0.088	-	69.1
Network 5	seg (C)	disp (R)	seg (R)	-	**0.070**	67.2	**72.4**
Network 6	seg (C)	seg (R)	disp (R)	-	0.075	67.1	69.3

Table 5.4: Results using different features from the backbone to predict a task. The maximum number of predictions a network can have is three. In the table, *disp* stands for disparity sub-network and *seg* for semantic segmentation sub-network. When the network outputs one task twice, the first output is considered as Coarse output, denoted by the letter (C) and the last output is considered as the Final output (R).

first a coarse segmentation. Then, the task-specific features are shared with the disparity sub-network, and finally, the task-specific features of both tasks are shared with a third sub-network, which is in charge of outputting the refined segmentation.

The table also shows two interesting results. 1) If a network predicts a refined version of a previous task, the second output will obtain a better score than a network that predicts that task only once. 2) The networks that have a refined output of one task and receive as input the features of its coarse version and the other task, will perform better than the networks whose refined sub-network only receive the coarse features. For example, Network 3 and 4 predict a refined disparity. Network 4 has a D-1 error of 0.088, meanwhile, Network 3, where the last sub-network receives both subtasks as input, obtains a lower D-1 error of 0.078. Similarly, Networks 5 and 6 predict a refined semantic segmentation. Network 5, whose refined branch receives both tasks, has an IoU of 72.4%, which is higher than Network 6 by 3.1%.

Sharing task-specific features

A series of experiments were conducted to test the effectiveness of sharing task-specific features among the branches of the proposed network. Recall that it concatenates feature information F_{seg} from the coarse segmentation branch with the disparity, and then uses the segmentation and disparity features, F_{seg} and F_{disp}, to refine the segmentation. Therefore, 3 variations of our proposed network were trained, and then tested on the validation set of Cityscapes: 1) All the connections between the task-specific features and branches were removed, with exception of the features that are used from the backbone, 2) The connection between the features from the coarse segmentation (F_{seg}) and the disparity decoder was kept, but the connections of the disparity and the coarse segmentation features (F_{seg} and F_{disp}) with the refined segmentation branch were removed, and 3) The connection F_{seg} to the disparity branch was removed and the connections F_{seg} and F_{disp} with the refined segmentation branch were kept. Table 5.5 shows the

F_{seg} $\downarrow$ Disparity branch	$F_{seg} \wedge F_{disp}$ $\downarrow$ Refined seg. branch	Coarse seg. IoU	Refined seg. IoU	Disp D-1 error
✗	✗	73.3	64.8	0.064
✓	✗	73.3	64.7	0.051
✗	✓	72.5	75.7	0.060
✓	✓	**74.6**	**77.6**	**0.040**

Table 5.5: Task-specific feature sharing effect. The first two columns show if the features F_{disp} and F_{seg} were concatenated to the disparity or refined segmentation branches. Coarse seg. and Refined seg. IoU show the results for coarse and refined segmentation.

results of each experiment in rows 1, 2, and 3, respectively; row 4 shows the original model with all the connections.

The experiments show that by having the connection between the coarse segmentation features F_{seg} and the disparity branch, the disparity error gets reduced, from 0.064 to 0.051 (results from the first and second row). The lowest disparity error, 0.040, is obtained when F_{seg} and F_{disp} are shared with the refined segmentation branch (fourth row). This result points out that, even though this last connection does not affect the disparity prediction at inference time, the progressive connection between task-specific features does help in the training process to learn better descriptors.

These experiments also demonstrate that the refined segmentation is improved when it receives the task-specific features F_{seg} and F_{disp}. The improvement ranges from 64.8% to 75.7% when the disparity branch does not receive F_{seg}, to 77.6% when it does. This also shows that if the disparity branch receives information from the coarse segmentation task, it can learn more meaningful features that can be passed to the refined segmentation branch.

Finally, the experiments show that the refined segmentation performs worse than the coarse segmentation (first two rows of Table 5.5), when it does not receive features F_{seg} and F_{disp}. This is because it can only use the features at the beginning of the backbone (low-level features), which are not enough for such a complex task as semantic segmentation.

5.3.5 Effectiveness vs. efficiency

It is important to note that the precision of the algorithm and its efficiency (measured in the number of parameters) are, quite often, opposite objectives, since attempting to improve one of them usually implies a deterioration in the other. From this point of view, this task can be seen as a Multi-objective Optimization Problem (MOP) in which two functions are optimized simultaneously.

The means commonly employed to evaluate this type of problem is the use of the concept of non-dominance: One solution is said to dominate another if, and only if, it is better or equal in each objective function and, at least, strictly better in one of them. The best solutions (there may be more than one) are those that are non-dominated. The strategies within this set can be considered the best without having to define any order among them [230].

Figure 5.10 compares the semantic segmentation precision and the efficiency of the algorithms evaluated assuming a MOP scenario. In addition, the Pareto frontier is marked with the non-dominated results, in which four combinations are found to be non-dominated for the Cityscapes dataset (Our proposal, Ladder D121, Ladder D169, and Panoptic) and two for the Garden dataset (Our proposal and DTIS). For both datasets, our solution is the multi-tasking approach with the best combination of efficiency and precision. The rest of the non-dominated results are optimized for a single task, or they are multi-task solutions but much less efficient.

5.3.6 Speed of the model

We computed the float-point operations (FLOPs)[4] of our method and compared it with DeepLab v3 with the Xception-71 backbone using an input size of $[2048 \times 1024]$. DeepLab v3 has 2.77 TFLOPs and obtains a result of 82.1% mIoU on the Cityscapes test set. Meanwhile, our method uses 3.1 TFLOPs and obtains a performance of 76.0%. Although our method uses more floating-point operations and has slightly lower performance, we should consider that, unlike DeepLab v3, it has two encoders, one for the left and another for the right image, this means that the number of flops will be doubled for the encoder. Also, our network has multiple sub-networks that not only compute the semantic segmentation but also predict the disparity. Moreover, it is important to highlight that disparity estimation networks usually have more flops than semantic segmentation ones. For example, a well-known disparity network PSM has 8.21 TFLOPs using the same input size, while our disparity sub-network only needs 1.2 TFLOPs. Although our network is slower than DeepLab v3, it presents a good trade-off between speed and practical utility, as running both tasks separately will use more flops.

[4]The values were computed using `https://github.com/sovrasov/flops-counter.pytorch`

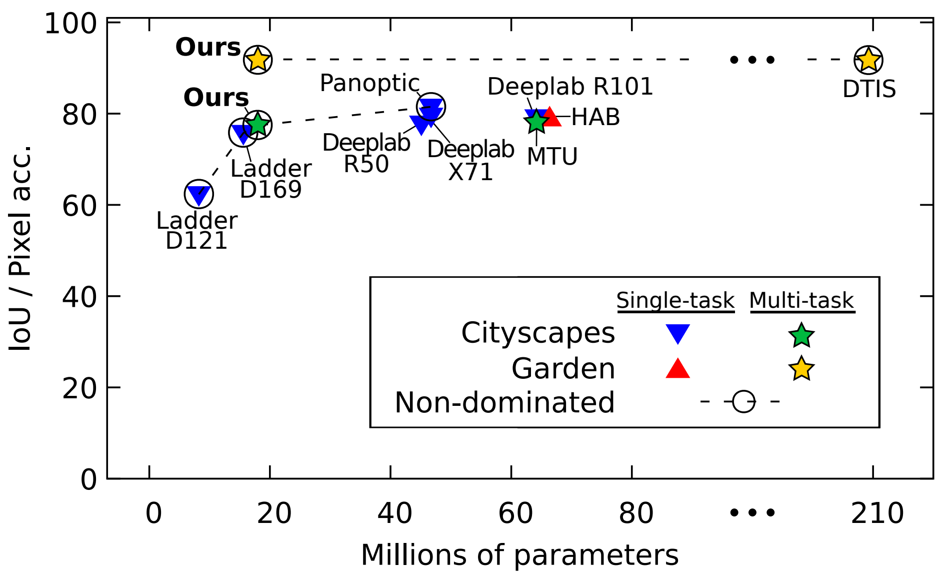


Figure 5.10: Analysis of efficiency (measured in the number of parameters) and effectiveness (considering IoU for the Cityscapes dataset and Pixel Accuracy for TrimBot Garden) as a Multi-objective Optimization Problem (MOP). Non-dominated elements and multi-task approaches are highlighted.

5.4 Conclusions

An end-to-end method based on multi-task learning that successfully learns the semantic segmentation and disparity map together from a stereo pair has been presented.

It has been experimentally demonstrated that using separate decoders for each task is not sufficient to achieve good performance, and that sharing the knowledge of segmentation and disparity tasks between them, in a progressive fashion, improves the learning and results on both tasks. Since the features of each task are obtained at different resolutions, these task-specific features also provide multi-scale information for coarse-to-fine learning. In addition, sharing the learned features also helps to reduce the complexity of the network and thus increase its performance, as demonstrated previously in [60, 61].

On the other hand, Standley *et al.* [61] showed that smaller networks tend to have a lower capacity to deal with multitask problems. However, our network manages to outperform the state of the art of multi-task learning on the Cityscapes and TrimBot Garden datasets, and obtains competitive results against the methods that solve each task individually using $1/3$ of the parameters. The reduced number of parameters allows its application on systems that have limited resources. For instance, it can run inference with 2048×1024 px images on a single GTX-1080Ti and half the resolution on a GTX-1060.

An area of improvement would be to reduce the FLOPs of the network by implementing more efficient layers like separable convolutions in the bottlenecks of the network.

Chapter 6

3D Segmentation

A robot or agent not only uses cameras to collect information, but also other sensors that capture the 3D information of its surroundings. This chapter presents a method to semantically segment and classify unordered point clouds using a two-headed attention layer to process the geometric information and latent features separately. The research reported in this chapter was previously published in [3].

6.1 Introduction

Robotics, autonomous driving, and related areas rely heavily on information captured by 3D sensors like RGB-D cameras, stereo cameras, and LiDARs. This information provides to the agent (robots or cars) the 3D location of their surroundings, which can be processed and used in tasks like scene understanding, path planning, navigation, among others [133, 2, 115]. One way to find the objects in the 3D space and their location is through point cloud segmentation. A point cloud is a set of points in 3D, usually unordered and sparse; some regions can be densely populated and others empty. This type of non-grid structured data makes it difficult to be used with convolution operators with the same efficiency as their 2D counterpart.

Various approaches have been proposed to handle such data. Some approaches project the 3D raw data into a regular structure (e.g. voxels) where 3D convolutions can be used [86, 87, 88, 89, 90, 91]. Other approaches use multilayer perceptrons (MLP) to process point clouds directly [11, 93, 10]. A third approach is to project the points to an intermediate grid structure where 2D convolutions can be used [98, 99, 100]. Lately, with the success of transformers and attention mechanisms in the area of natural language processing (NLP) [101], these methods have started to show dominance in this area [103, 104].

This chapter proposes a multi-head attention layer called Geometric-Latent Attention (Ge-Latto) to segment and label subsets of the point cloud. Ge-Latto is a two-headed local attention layer that evaluates a patch inside the point cloud and tries to find good relationships between the neighbor points. Unlike other works that combine all the features indiscriminately [105, 96], each attention head focuses on a specific type of feature. One head is in charge of finding good 3D geometric relations and the other in finding relationships among the latent features of the network. Similar to [98], we use an encoder-decoder network with residual connections. Each layer of the encoder sub-samples the input points, groups the points into neighborhoods, and uses our Ge-Latto layer to find local-spatial relationships from the latent and geometric features of neighbor points. The neighbors are found using radius neighborhoods instead of k-nearest-neighbors (kNN). The network increases this radius in each layer to increase the field of view and find relationships in bigger neighborhoods. In the decoder part, we up-sample the points using tri-linear interpolation. To ensure that in each sampled layer the network learns useful features, we add auxiliary losses similar to PSPNet [37] and RetinaNet [231]. In other words, the network predicts the segmentation for each sample size as seen in Figure 6.1. Our approach is also invariant to permutation because all the layers are shared MLPs.

The main contributions of this chapter are: **a)** A novel two-headed attention layer that is able to combine efficiently the geometric and latent information of unordered point clouds with variable densities for semantic segmentation. **b)** A pyramid-based encoder-decoder architecture with auxiliary losses to leverage feature patterns at different resolutions. **c)** State of the art performance on the complex dataset S3DIS, in not only area 5, but also in k-fold cross-validation, as well as competitive results on the ShapeNetPart and ModelNet40 datasets.

6.2 Methodology

Ge-Latto extracts two types of information from the point cloud: The geometric information is obtained from the Cartesian coordinates of the points and the latent feature information is learned by the network each time the point cloud is sub-sampled. The first part of this subsection describes the network architecture and sampling strategy. The second part describes the two-headed attention layer and explains how the latent and geometric information from each sampling is used.

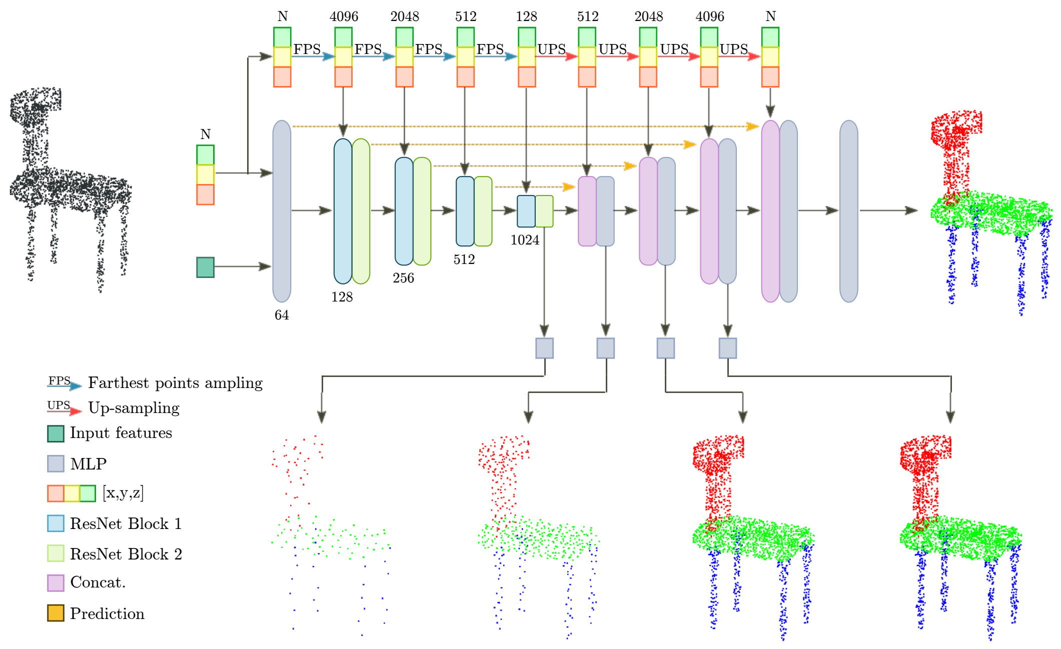


Figure 6.1: Overview of our network. The encoder-decoder architecture receives as input *xyz* coordinates and *RGB* (Input features). The numbers above the *xyz* coordinates represent the number of points that were sampled using Farthest Point Sampling (FPS). The numbers bellow each ResNet Block 1 show the feature dimension D of each layer. The encoder consists of ResNet blocks, which have our Ge-Latto layer (see Figure 6.3). The decoder consists of up-sampling layers which are concatenated with their respective encoder features using residual connections and combined with an MLP. The network has also 4 auxiliary outputs at multiple scales. One output comes from the last encoder layer and the other outputs plus the main output are obtained from the decoder layers.

6.2.1 Network and sampling

The network has an encoder-decoder architecture and receives as input the *xyz* coordinates and RGB color. Those features are projected to a higher dimension using a shared MLP layer[1] (see Figure 6.1). The encoder reduces the number of points and extracts high-level features from a neighborhood of points. For this, each layer subsamples the number of points of its input. Therefore $N_l > N_{l+1}$ where N is the number of points and l is the layer. The encoder has 4 layers that are designed like bottleneck ResNet blocks [33] with Ge-Latto replacing the 2D convolutions. Using Thomas *et al.* [10] configuration, the input features of a ResNet block are processed by an MLP layer followed by batch normalization and ReLU. The other MLPs of the block are only followed by batch normalization (see Figure 6.3 for details).

[1]In the chapter, we use the word MLP to refer to a shared MLP layer with 1 hidden dimension.

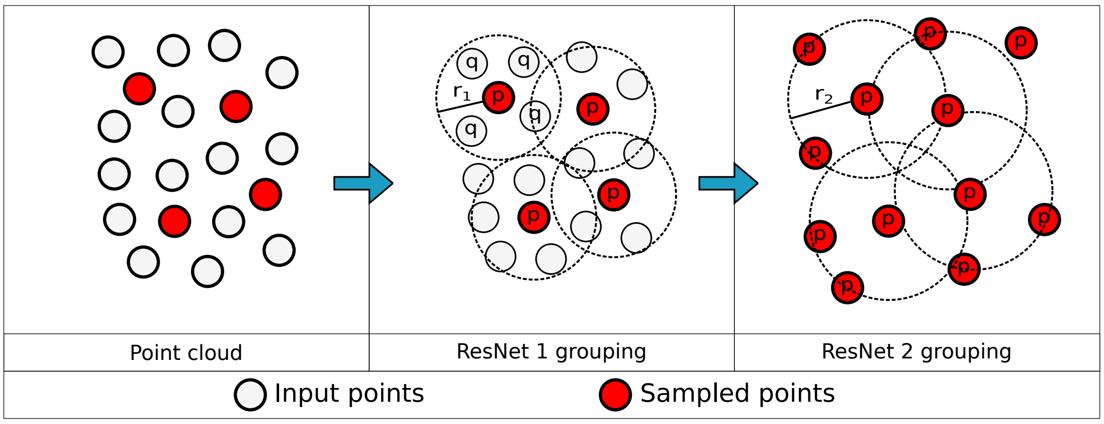


Figure 6.2: Clustering process inside ResNet blocks. The first image shows the input points of the layer. The grouping criteria of Block 1 and 2 are shown in the second and third image. Block 1 groups the input points using the sampled points as centers with a radius r_1, whereas Block 2 does the grouping on the sampled points with a bigger radius r_2.

Given the sparse nature of a point cloud, the choice of the sampling method is not trivial. The sampled points have to represent a group of points and be beneficial for the information “aggregation” of its neighbors. Here, we chose Farthest Point Sampling (FPS) because it outputs a more uniform-like distribution which is a desired property for a semantic segmentation network [10].

The next step groups each point p_i from the input set $\mathcal{P}_l$ (of size N_l) with their neighbors to find local-spatial relationships. The points can be either grouped by kNN or radius neighbors. We use the latter because it is more robust with non-uniform sampling settings like point clouds [10]. Therefore, for each representative point $p_i \in \mathcal{P}_l$, K points inside the given radius are randomly picked. We use $\mathcal{Q}_i$ to represent the grouped neighboring points of p_i in the rest of the chapter. It is important to note that $\mathcal{Q}_i \subseteq \mathcal{P}_{l-1}$, the only exception is in the second ResNet block, where $\mathcal{Q}_i \subseteq \mathcal{P}_l$ because no sampling is carried out; Figure 6.3 shows an example of this. The network also increases the receptive field by doubling the size of the radius at every layer.

For segmentation, the decoder up-samples the number of points until it recovers the size of the input of the network. The up-sampling of the features is done via tri-linear interpolation following Lin *et al.* [98]. The interpolated features are concatenated with the features from the corresponding encoder stage thanks to the residual connections (see Figure 6.1). The final and auxiliary outputs of the decoder are feature vectors for each point in the input point set. An MLP is used to map these features to the final logits, whose feature dimension is the number of classes. The size of the auxiliary out-

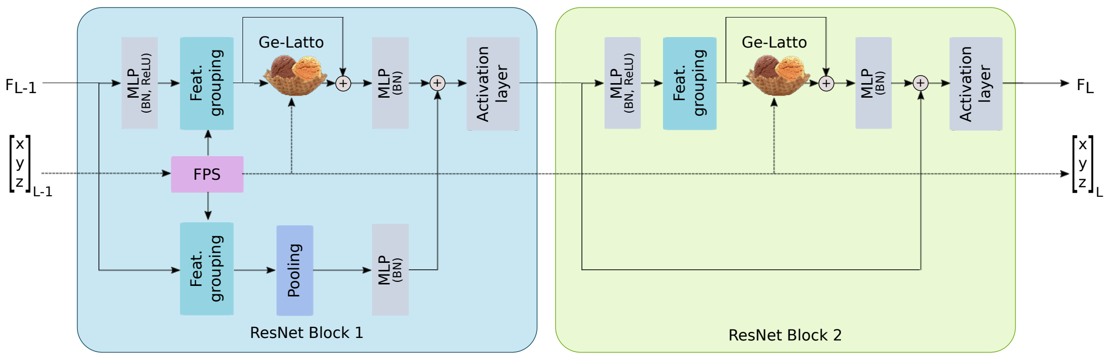


Figure 6.3: ResNet Blocks. The first block sub-samples the point cloud and finds nearest neighbors inside a radius between the sampled points and the input points. Because of the sampling, the residual connection has a maxpooling layer to match the input with the output size. The function of the second block is similar to the first one, but without sub-sampling the points.

puts corresponds to the number of points their respective layers have. The network has 4 auxiliary outputs, one for the last encoder layer, and three for the following decoder layers (see Figure 6.1). The auxiliary outputs are used by the auxiliary losses, which help optimize the learning process [37]. For classification, global average pooling is used over the last encoder features to get a global feature vector of the point cloud. This feature is passed to an MLP to obtain the classification logits.

6.2.2 Two-headed attention

We claim that our two-headed attention layer finds better features by performing the local aggregation of the geometric and latent information separately (Figure 6.4). The geometric features that are used are the absolute position of the representative points $p_i \in \mathbb{R}^3$, the K neighbor points $\mathcal{Q}_i \in \mathbb{R}^{K\times 3}$, and the relative position of the neighbors $\mathcal{Q}_i - \mathcal{K}p_i \in \mathbb{R}^{K\times 3}$, where the operator $\mathcal{K}$ replicates the vector K times. The latent information are the features learned by the hidden layers of the network. Each Ge-Latto layer uses those that belong to each centroid or representative point $r_i \in \mathbb{R}^D$, the features of the neighbors $\mathcal{S}_i \in \mathbb{R}^{K\times D}$, and the difference between the K neighbors and centroids features $\mathcal{S}_i - \mathcal{K}r_i \in \mathbb{R}^{K\times D}$, where D is the dimensionality of the features, which is the number of feature planes shown in Figure 6.1. The feature values are mapped linearly using an MLP layer f_i. The combined geometric and latent features are represented by $\mathcal{G}_i \in \mathbb{R}^{K\times D}$ and $\mathcal{H}_i \in \mathbb{R}^{K\times D}$ respectively, and are computed as follows (see Figure 6.4): First, the latent features r_i and $\mathcal{S}_i - \mathcal{K}r_i$ are transformed by MLPs and combined by vector addition: $\mathcal{H}_i = f_r(\mathcal{K}r_i) + f_{rs}(\mathcal{S}_i - \mathcal{K}r_i)$.

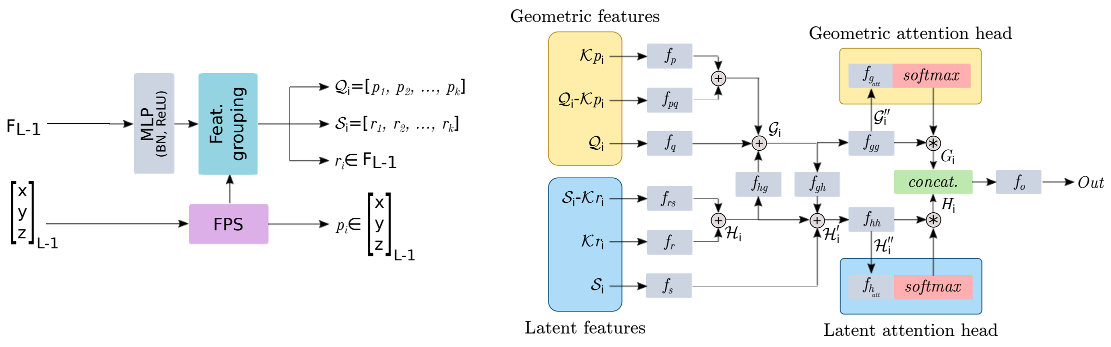


Figure 6.4: The two-headed Ge-Latto layer process the local-attention for the geometric and latent features individually and then combines them using f_i MLP layers. These features are obtained from the feature grouping and FPS blocks (See Figure 6.3). The FPS block outputs the sampled points p_i. The feature grouping block outputs the latent feature r_i that corresponds to the point p_i, the K neighbor points Q_i of p_i, and their latent features S_i.

Then, the geometric features are combined. From Eq. 6.1, $f_p(\mathcal{K}p_i)$ and $f_q(\mathcal{Q}_i)$ encode the global geometric context in 3D space of the representative points and its neighbors. Meanwhile $f_{pq}(\mathcal{Q}_i - \mathcal{K}p_i)$ represents the local geometric context. We augment the geometric context by adding and projecting the latent feature $\mathcal{H}_i$.

$$\mathcal{G}_i = f_p(\mathcal{K}p_i) + f_{pq}(\mathcal{Q}_i - \mathcal{K}p_i) + f_q(\mathcal{Q}_i) + f_{hg}(\mathcal{H}_i) \tag{6.1}$$

In the same way, the latent features are combined and augmented by adding the projected geometric feature $\mathcal{G}_i$. As Eq. 6.1 encodes the geometric context, Eq. 6.2 encodes the latent context. $f_r(\mathcal{K}r_i)$ and $f_s(\mathcal{S}_i)$ represent the global latent information and $f_{rs}(\mathcal{S}_i - \mathcal{K}r_i)$ represents the local latent information.

$$\mathcal{H}_i' = \mathcal{H}_i + f_s(\mathcal{S}_i) + f_{gh}(\mathcal{G}_i) = f_r(\mathcal{K}r_i) + f_{rs}(\mathcal{S}_i - \mathcal{K}r_i) + f_s(\mathcal{S}_i) + f_{gh}(\mathcal{G}_i) \tag{6.2}$$

The resultant features $\mathcal{G}_i$ and $\mathcal{H}_i'$ are each projected by another MLP layer: $\mathcal{G}_i'' = f_{gg}(\mathcal{G}_i)$ and $\mathcal{H}_i'' = f_{hh}(\mathcal{H}_i')$. Then, self-attention is used to combine the features inside the neighborhood patch (Eq. 6.3 and Eq. 6.4). The attention part consists of an MLP layer followed by a normalization function (Softmax) ϕ to obtain the weights of the neighbor features. Here, vector attention is used instead of scalar attention. This allows the network to “attend” to individual feature channels [105]. The dimension of the attention weights, the geometric features $\mathcal{G}_i''$ and latent features $\mathcal{H}_i''$ is $[K \times D]$. Finally, to aggregate the local features, each neighbor feature $g_k \in \mathcal{G}_i''$ and $h_k \in \mathcal{H}_i''$ is multiplied element-wise by its respective weight and then all the neighbors k are summed. The outputs G_i and H_i have dimension D.

$$G_i = \sum_{k=1}^{K} (\phi(f_{g_{att}}(g_k)) \odot g_k) \tag{6.3}$$

$$H_i = \sum_{k=1}^{K} (\phi(f_{h_{att}}(h_k)) \odot h_k) \tag{6.4}$$

The output O_i of the layer is obtained by concatenating and projecting the geometric and latent features: $O_i = f_o([G_i; H_i])$, where $[G_i; H_i] \in \mathbb{R}^{2D}$ and $f_o : \mathbb{R}^{2D} \mapsto \mathbb{R}^{D}$.

In the transformers literature [101, 102], the geometric features $f_p(\mathcal{K}p_i)$ and $f_{pq}(\mathcal{Q}_i - \mathcal{K}p_i)$, from Eq. 6.1, can be seen as absolute and relative positional encodings, respectively. Therefore, Eq. 6.1 provides information about the absolute position of the points, and the relative position of the neighbor points with respect to the representative (centroid) points. Similarly, $f_{rs}(\mathcal{S}_i - \mathcal{K}r_i)$ from Eq. 6.2 can be seen as the *key* and *query* components in the transformers settings, where instead of using dot product as similarity function to obtain the relationship between two vectors, the values are subtracted.

Each head (geometric or latent) in our attention mechanism can be considered as a multi-head attention layer with feature dimension (channels) $D' = 1$ for each head or number of heads $n = D$, where D is the dimensionality of the features in each layer. This is demonstrated as follows. Considering Eq. 6.5, the multi-head equation of a feature vector $H_i \in \mathbb{R}^D$ proposed by Vaswani *et al.* [101].

$$\begin{aligned} H_i &= \mathrm{Concat}(\mathrm{head_{i,1}}, \mathrm{head_{i,2}}, \mathrm{head_{i,3}}, \ldots, \mathrm{head_{i,n}}) \\ \mathrm{head_{i,n}} &= \sum_{k=1}^{K} (\mathcal{M}\phi(Similarity_k) \odot Value_k) \end{aligned} \tag{6.5}$$

where :

$$\begin{aligned} \mathcal{M} &= \text{Replicates the vector } D' \text{ times} \\ \phi &= \text{Normalization function} \\ dim(Similarity_k) &= 1 \\ dim(Value_k) &= D' \\ dim(\mathrm{head_n}) &= D' \\ dim(H_i) &= D = nD' \\ K &= \text{Neighborhood size} \end{aligned}$$

Eq. 6.5 shows that there is one weight ϕ per head n, and each ϕ multiplies D' feature channels. Therefore, **the feature $H_i \in \mathbb{R}^D$ has n weights ϕ and nD' feature channels**. In the case where the number of heads n is D, D' would have the value of 1. There would be one weight ϕ per head, and each weight would multiply one feature channel. **This would give a vector H_i with D weights ϕ and D feature channels**, which are the dimensions of our geometric and latent head features, as seen in Eq. 6.6 (latent feature equation).

$$H_i = \sum_{k=1}^{K} (\phi(f_{h_{att}}(h_k)) \odot h_k) \tag{6.6}$$

Where the dimension of $f_{h_{att}}(h_k)$, h_k and H_i is D.

Geometric and Latent Attention Scores: To illustrate the attention scores learned by our Ge-Latto layer in each encoder step, an input point, that was not discarded by the sample process, was picked. Because the attention score of each point has a dimension D, where D is the dimensionality of a given layer, we randomly picked a value $d \in D$ per attention score to be shown in Figure 6.5 and Figure 6.6 for the geometric and latent heads, respectively.

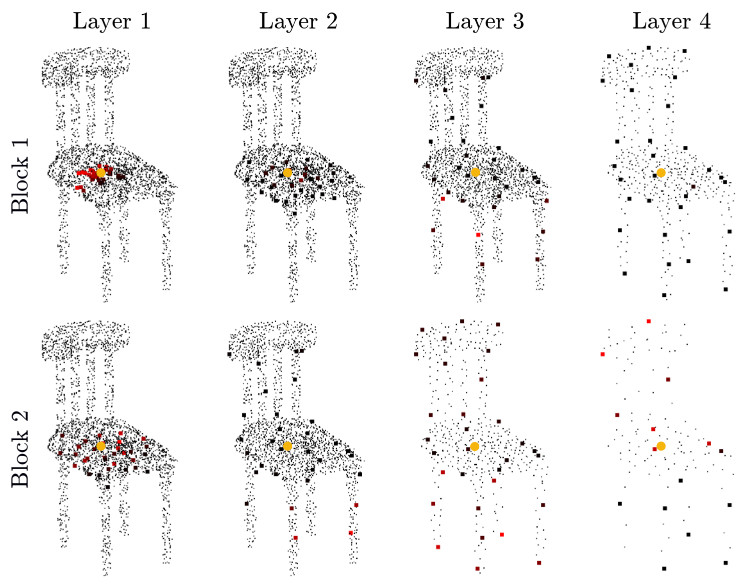


Figure 6.5: Learned geometric attention scores of a point p_i(in yellow). The attention scores are represented in red, the stronger the intensity, the higher the score. The small black points are the sampled points at a given layer, the bigger points are the picked neighbor points inside a radius. The image shows the attention scores of the two ResNet Blocks at every encoder layer.

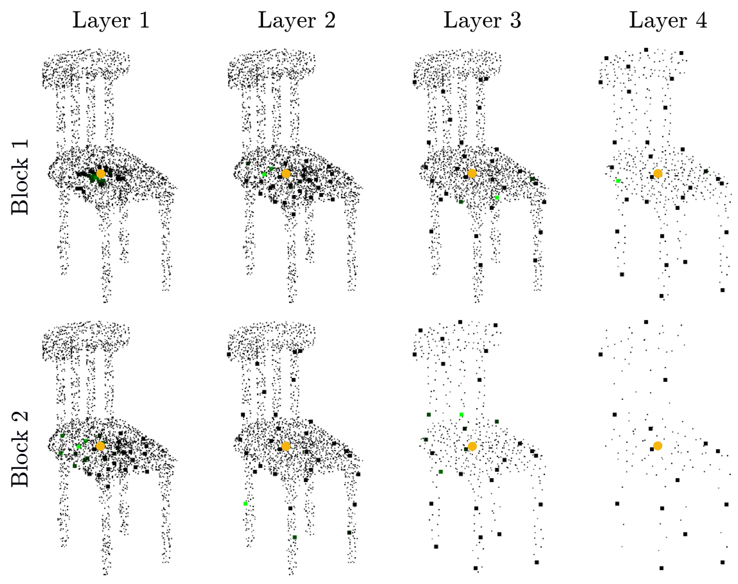


Figure 6.6: Learned latent attention scores of a point p_i (in yellow). Similar to the learned geometric attention example (Figure 6.5), the attention scores are represented in green and the bigger black dots are the points that are picked inside the neighbor radius.

To observe how our network captures global and local relationships, all the attention scores of each head were grouped in Figure 6.7. The figure shows that, for the chosen point, the latent head focuses more on closer points, meanwhile, the geometric head not only pays attention to the local points but also to more distant points, such as those of the back of the chair and legs.

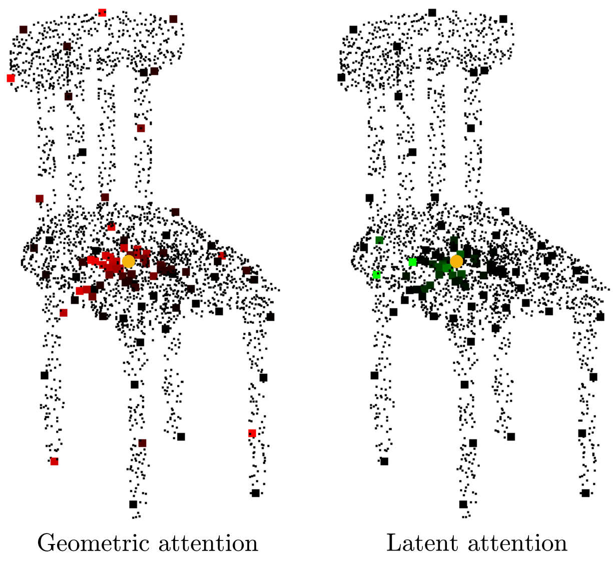


Figure 6.7: Geometric and Latent attention scores grouped.

6.3 Experiments

Our method was evaluated using 3 datasets: ShapeNetPart [232] for 3D object part segmentation, Stanford Large-Scale 3D Indoor Spaces (S3DIS) [233] for 3D scene segmentation, and ModelNet40 [234] for 3D shape classification.

Implementation details: The implementation was built using the public library PyTorch [224]. We use Adam as optimizer with learning rate 1e$-$4 and the default hyperparameters: $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\varepsilon = 1\text{e}{-9}$. Cross-entropy with label smoothing is used as the loss function for all the outputs. The final loss consists of the sum of 4 auxiliary losses and the main loss: $\mathcal{L} = \alpha_1 * \mathcal{L}_{aux_1} + \alpha_2 * \mathcal{L}_{aux_2} + \alpha_3 * \mathcal{L}_{aux_3} + \alpha_4 * \mathcal{L}_{aux_4} + \mathcal{L}_{main}$. The influence of the auxiliary losses is weighted by α_i because we are only interested in the final prediction, which is optimized by the main loss. Following the results of our ablation study, all the $\alpha_i = 0.4$ in the experiments. The encoder consists of one layer with size N, to process the input features, followed by 4 layers with sizes: 4096, 2048, 512, and 128; as seen in Figure 6.1. The radius (receptive field) of the first encoder layer with ResNet blocks is $0.10m$ and it doubles at every layer. The number of neighbors is 32 for all the layers except for the last one which is 16. This is because the last layer has fewer points than the rest. All the MLP layers from the ResNet blocks (see Figure 6.3) are followed by batch normalization and ReLU. The output layer before the prediction consists of an MLP layer with batch normalization and ReLU followed by a dropout with a probability of 0.5. All the experiments were done using a single RTX2080Ti with a batch size of 2. The data augmentation consists of scaling, flipping, rotating, and perturbing the points. For S3DIS, the color was augmented by switching the RGB channels and adding noise.

6.3.1 Scene segmentation

The S3DIS [233] dataset was used to test the network for scene segmentation. The dataset consists of six real large-scale indoor areas from three different buildings. Each area has rooms whose points are labeled with 13 classes (e.g. ceiling, floor, chair) and have color information. The number of points in one room varies between 0.5 million to 2.5 million, depending on its size. Because the number of points of each room is large, each room was split into blocks of size $[2m \times 2m \times height]$. For training and testing, 6,144 points were randomly sampled and used as input. However, for testing, 6,144 points are randomly sampled until all the points inside a block are labeled. All the points are only sampled once. However, because the total number of points inside a

Method	mIoU	mAcc	ceiling	floor	wall	beam	column	window	door	table	chair	sofa	bookcase	board	clutter
PointNet [11]	41.1	49.0	88.8	97.3	69.8	**0.1**	3.9	46.3	10.8	59.0	52.6	5.9	40.3	26.4	33.2
SegCloud [235]	48.9	57.4	90.1	96.1	69.9	0.0	18.4	38.4	23.1	70.4	75.9	40.9	58.4	13.0	41.6
FPConv [98]	62.7	68.9	**94.6**	98.5	80.9	0.0	19.1	60.1	48.9	80.6	88.0	53.2	68.4	68.2	54.9
MinkowskiNet [236]	65.3	71.7	91.8	98.7	**86.2**	0.0	**34.1**	48.9	62.4	81.6	89.8	47.2	74.9	74.4	58.6
KPConv [10]	67.1	72.8	92.8	97.3	82.4	0.0	23.9	58.0	**69.0**	81.5	91.0	75.4	**75.3**	66.7	58.9
PCT [103]	61.3	67.6	92.5	98.4	80.6	0.0	19.4	61.6	48.0	76.6	85.2	46.2	67.7	67.9	52.3
Bilateral [104]	65.4	73.1	92.9	97.9	82.3	0.0	23.1	**65.5**	64.9	78.5	87.5	61.4	70.7	68.7	57.2
Ge-Latto (ours)	**69.2**	**75.9**	94.5	**99.2**	84.0	0.0	24.5	56.3	68.9	**84.2**	**92.4**	**82.8**	70.9	**76.9**	**64.6**

Table 6.1: S3DIS Area 5 results. The reported metrics are the mean class segmentation (mIoU), mean of class-wise accuracy (mAcc), and IoU for each class.

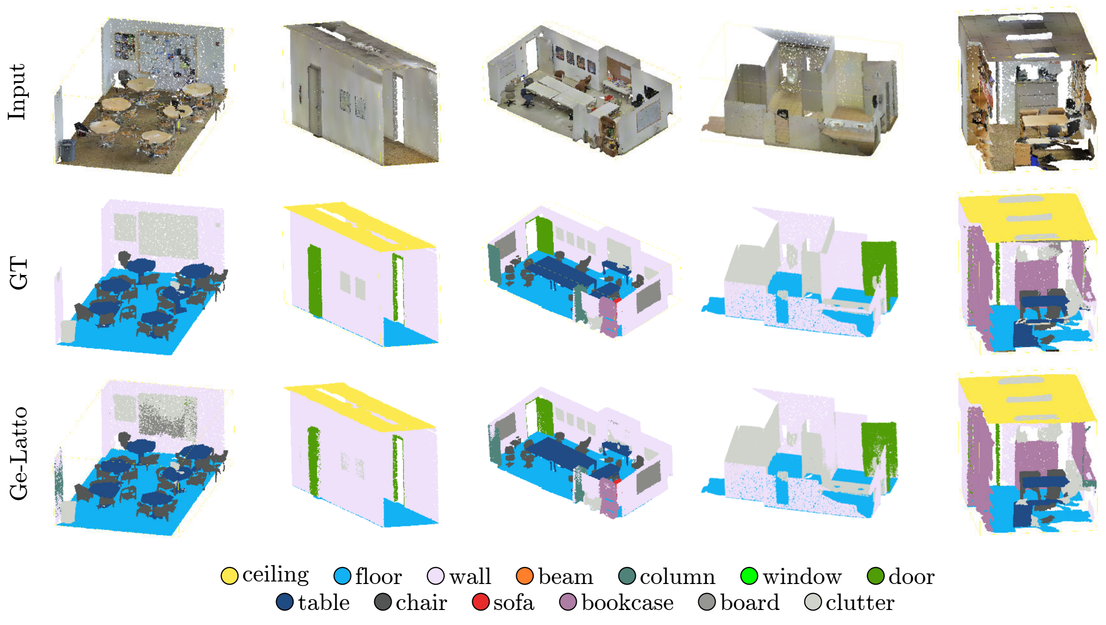
Method	OA	mIoU	mAcc	ceiling	floor	wall	beam	column	window	door	table	chair	sofa	bookcase	board	clutter
PointNet [11]	78.5	47.6	66.2	88.0	88.7	69.3	42.4	23.1	47.5	51.6	54.1	42.0	9.6	38.2	29.4	35.2
PointCNN [96]	88.1	65.4	**88.1**	94.8	**97.3**	75.8	63.3	51.7	58.4	57.2	71.6	69.1	39.1	61.2	52.2	58.6
RandLA-Net [237]	88.0	70.0	82.0	93.1	96.1	80.6	62.4	48.0	64.4	69.4	69.4	76.4	60.0	64.2	65.9	60.1
KPConv [10]	-	70.6	79.1	93.6	92.4	**83.1**	63.9	**54.3**	**66.1**	**76.6**	64.0	57.8	**74.9**	**69.3**	61.3	60.3
Bilateral [104]	88.9	**72.2**	83.1	93.3	96.8	81.6	61.9	49.5	65.4	73.3	72.0	**83.7**	67.5	64.3	67.0	**62.4**
Ge-Latto (ours)	**89.7**	71.4	81.3	**95.3**	95.1	82.3	**69.2**	51.9	64.8	73.3	**77.3**	59.6	71.1	63.0	**67.4**	57.9

Table 6.2: S3DIS dataset k-fold cross-validation comparison table.

block is hardly a multiple of 6,144, the last 6,144 sampled points will contained points already drawn. In that case, the Softmax outputs are summed and the highest value is used as the predicted label. The evaluation metrics used are mean class-wise intersection over union (mIoU), mean of class-wise accuracy (mAcc), and overall accuracy over all points (OA).

Following the standard evaluation process, the dataset was evaluated in two ways: 1) Area 5 is used as test set and the network is trained using the other areas. 2) 6-fold cross-validation. Ge-Latto outperforms prior models in both evaluations. On area 5, it is 2.1% better than KPConv [10] in mIoU (Table 6.1), the qualitative results are shown in Figure 6.8. Meanwhile, on the k-fold cross-validation, it obtains the best OA (89.7%), surpassing the previous state of the art of Qiu *et al.* [104] and obtaining better IoU in more objects (Table 6.2).

Table 6.1 not only shows the overall comparison between methods, but also the IoU of each class. We can observe that our method works better in classes like floor, table, chair, sofa, board, and clutter. This result is also supported by comparing visually the segmentations of our method and the state-of-the-art KPConv; the comparison is shown in Figure 6.9. For example, KPConv struggles to segment the legs of the tables while our method labels them correctly. Also, KPConv misses the class of the pinboard (clutter class), whereas our method manages to segment most of the pinboard correctly. The other class our network labels the pinboard is the class board. Finally, Figure 6.9 also shows why the performance of the class wall of our network is slightly lower than KPConv. This is because the network sometimes confuses a wall with the class column when the wall is thin.



Input
GT
Ge-Latto
● ceiling ● floor ● wall ● beam ● column ● window ● door
● table ● chair ● sofa ● bookcase ● board ● clutter

Figure 6.8: S3DIS results.

Method	ModelNet40	ShapeNetPart	
	OA	cat. mIoU	inst. mIoU
PointNet [11]	89.2	80.4	83.7
PointNet++ [93]	91.9	81.9	85.1
SO-Net [94]	90.9	81.0	84.9
KPConv [10]	92.7	**85.1**	**86.4**
PCT [103]	**93.2**	-	**86.4**
Ge-Latto (ours)	91.1	84.2	84.5
Ge-Latto (ours fine-tuned)	**93.2**	-	-

Table 6.3: ModelNet40 and ShapeNet comparison table.

6.3.2 Object part segmentation

The performance of the network in object part segmentation is measured using the ShapeNetPart [232] dataset. This dataset is a collection of 16,681 3D point clouds with 16 categories, each with 2 to 6 part labels. The standard train/test splits provided by the dataset is used. Category mean intersection over union (cat. mIoU) and instance mIoU are used as evaluation metrics. For training, 4096 points are randomly picked and used as input. For testing, the total number of points in a point cloud is used. The evaluation shows that our model is only 0.9% behind in cat. mIoU from the current state of the art. Some of the wrong classifications are caused by noisy data, where some object components (e.g. rocket, motorbike, table) are wrongly labeled in the ground truth which is penalized by the metric. Figure 6.10 shows some qualitative results.

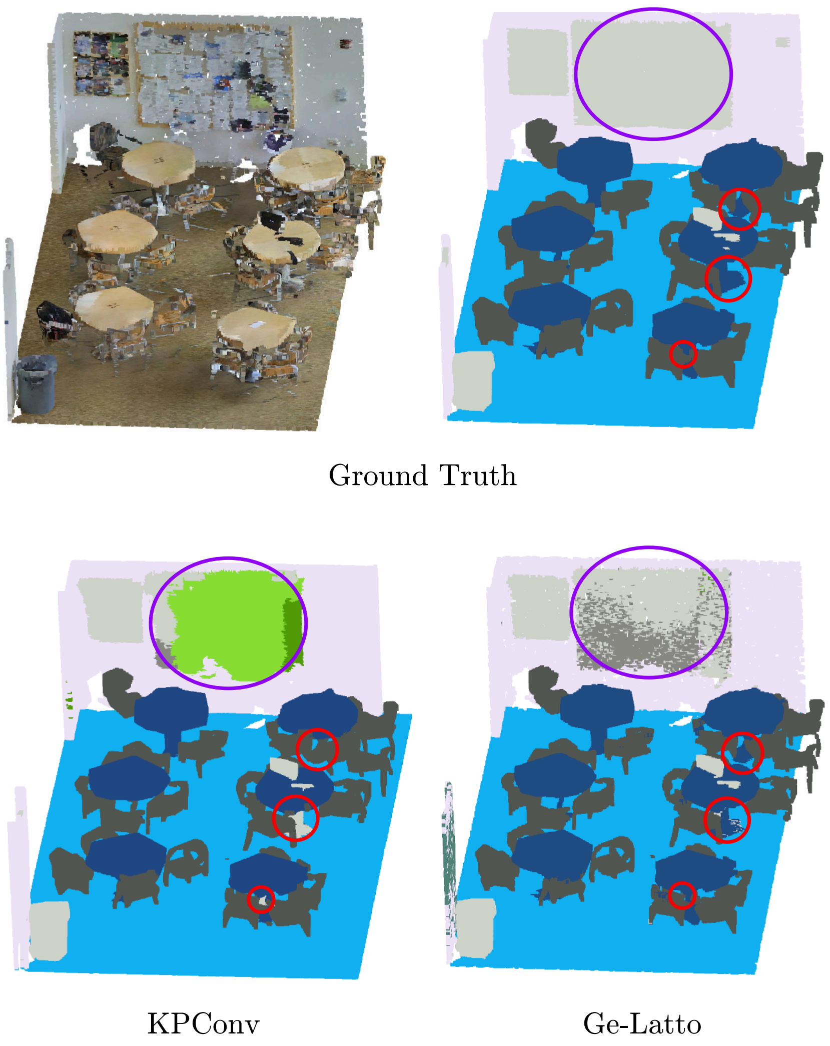


Figure 6.9: Visual comparison between KPConv and Ge-Latto outputs of Lobby_1 Area 5. The image shows that our method can segment each part of the table better than KP-Conv (red circles). Also, our method is able to segment most of the pinboard correctly, whereas KPConv totally misses the correct class (purple circle).

6.3.3 Shape classification

The ModelNet40 [234] dataset is used to study the performance of our network for shape classification. The dataset consists of 12,311 3D meshes and their normal vectors classified into 40 categories. For the experiments, the standard training and validation split is used the data is processed similar to Section 6.3.2.For training, 7,168 points are randomly picked as input, and for testing all the points are used. The evaluation metric is overall accuracy. Table 6.3 shows that our method achieves a competitive result of 91.1% when using the same hyper-parameters and network structure as ShapeNet. If the parameters are fine-tuned, the performance goes up to 93.2%, matching the current

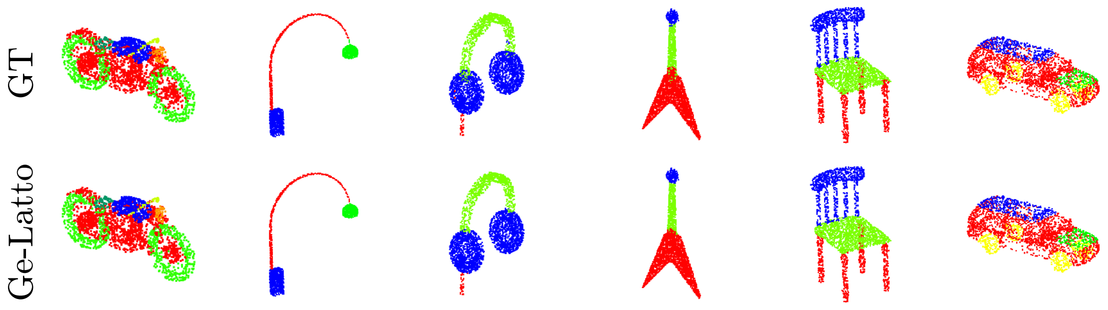


Figure 6.10: Our ShapeNetPart segmentation results.

Auxiliary losses			Attention heads			Number of neighbors			Features per multi-head		
Aux. weight α	mIoU	mAcc	Attention heads	mIoU	mAcc	k	mIoU	mAcc	N Feat.	mIoU	mAcc
$\alpha = 0.0$	67.4	73.2	Only geometric	66.3	72.0	8	64.5	71.8	1	**69.2**	**75.9**
$\alpha = 0.2$	68.5	74.4	Only features	66.5	72.3	16	66.4	72.1	2	68.2	74.3
$\alpha = 0.4$	**69.2**	**75.9**	Both	**69.2**	**75.9**	32	**69.2**	**75.9**	4	68.0	74.2
$\alpha = 0.6$	68.6	74.6	MLP+pooling	63.5	69.2	64	-	-	8	68.0	74.1
$\alpha = 0.8$	68.3	74.3									
$\alpha = 1.0$	68.1	74.0									

Table 6.4: S3DIS ablation study experiments.

best result. More details are given in the ablation study.

6.3.4 Ablation study

Auxiliary losses. The auxiliary losses provide a boost in performance to the network. Here, we consider that all weights α_i of the auxiliary losses have the same value and vary them from 0 to 1. Table 6.4 shows that the best performance is obtained with $\alpha_i = 0.4$, being 1.8% (in terms of mIoU) better than the network trained without auxiliary losses.

We also explored different values for each alpha using grid search. The experiment consists of varying the alpha of one auxiliary loss, from 0 to 1 with increments of 0.2, and fixing the other alphas to 0.4 (the best value found before); the process is repeated for the 4 auxiliary losses. The values that the varying alpha can take are [0, 0.2, 0.6, 0.4, 0.8, 1], where 0 means that we do not minimize the loss for that output. We trained 24 variations of our network (4 auxiliary losses with 6 values for alpha). Each network was initialized with the weights from our best model, for the S3DIS area 5 dataset, and trained for 50 epochs. This experiment showed that there is no improvement when we vary the alphas individually and that the best value for this parameter is 0.4. However, we still observe that the use of auxiliary losses improves the performance of the network. As seen in Table 4 in the main paper, in the ablation study, when the network is trained without auxiliary losses, it obtains an IoU of 67.4%.

Meanwhile, when the auxiliary losses are added, the performance increases to 69.2%.

Two-headed attention. Our proposed attention layer is evaluated using different variants: only with the geometric head, only with the feature head, both heads, and no heads (baseline). For the last one, the geometric and latent features are concatenated and then processed by an MLP+pooling layer, this replaces the attention mechanism. The experiments reported in Table 6.4 show that only one head is enough to improve the baseline, and that the combination of both heads helps the network in the segmentation task.

Number of neighbors. Three networks with different neighborhood size were trained to find the number of neighbors that provide enough local information. The results from Table 6.4 show that a when $k \leq 16$, the number of neighbors might not be enough to provide a correct representation of the local context; $k = 64$ could not fit in the GPU memory.

Features per multi-head. As showed in Eq. 6.5 and Eq. 6.6, each head (geometric and latent) of our attention layer is a special case of a multi-head attention with number of heads $n = D$ or feature dimension $D' = 1$ per head. For this experiment, the number of features D' per head was varied. More features per head means less heads (number of heads $n = D/D'$). In other words, D' features will be weighted by the same attention score. The results from Table 6.4 demonstrate that the more features a head has (less number of heads), the less flexible the network becomes. However, reducing the number of heads allows the network to be lighter, because it has to compute only D/D' attention scores.

Point cloud classification. The model presented in Section 6.3 had the same network parameters for all the datasets to show that the proposed method can obtain competitive results without optimizing the hyper-parameters on each dataset. If the hyper-parameters are adjusted for a specific dataset, the performance of the network improves. In this section, the number of sub-sampling points per layer were modified following [238], where they mainly focus on point cloud classification. By sampling the points using the following values per layer, $N \rightarrow 1024 \rightarrow 512 \rightarrow 256 \rightarrow 64$, instead of $N \rightarrow 4096 \rightarrow 2048 \rightarrow 512 \rightarrow 128$, the overall accuracy of our network increases from 91.1% to 93.2%, matching the state-of-the-art result. This seems to be caused by the number of sampled points and the global average pooling at the end of the encoder layer. Considering that the ModelNet40 dataset has objects with similar parts, like plants with flower pots, if one region of the object is bigger than others, this region will have more sampled points. Because the features of the sampled points at the last

Method	Parameters	mIoU
MinkowskiNet	21.7M	65.3
KPConv	25.8M	67.1
PCT	**2.88M**	61.3
FPConv	17.6M	62.7
Ge-Latto (ours)	15.3M	**69.2**

Table 6.5: Model parameters comparison table. The parameters are in millions and the metric is for the S3DIS dataset.

Number of points	6K	10K	20K	200K
Inference batch size	5	3	3	3
Inference time	100ms	200ms	210ms	300ms
Training batch size	2	-	-	-
Training time	160ms	-	-	-

Table 6.6: Training and inference time.

layer of the encoder are averaged, if there are more points representing a specific area, the averaged features will have a tendency to represent the wrong part of the object.

Model Size and Speed. Table 6.5 shows that our method achieves the state-of-the-art in the S3DIS dataset with fewer parameters than the previous methods. Our network has only 15.3M parameters, whereas KPConv has 25.8M, MinkowskiNet 21.7M, and FPConv 17.6M. The only network that has fewer parameters is PCT, 2.88M. However, we are 8% better in point cloud semantic segmentation and obtain a similar performance in ModelNet40. It is important to highlight that KPConv, FPConv, and our method have similar network architectures with the only difference that their proposed layers are replaced by our Ge-latto layer. This shows that the performance improvement is obtained by our layer and not by the architecture.

We also tested the training and inference times using different numbers of points and batch sizes. The results are reported in Table 6.6. The table shows that it takes around 160ms to train 6144 points with a batch size of 2. At inference time, our network can analyze 6144 points with a batch size of 5 in 100ms, 20K points with a batch size of 3 in 210ms, and 200K points with a batch size of 3 in 300ms. All the tests were done using an NVIDIA RTX2080ti. These experiments show that our network is suitable for applications that need a lighter network.

Work flow video. The video of our proposed method can be found at: `https://youtu.be/mjsttn3C89g`.

6.4 Conclusions

This chapter proposes a novel two-headed attention mechanism capable of combining the geometric and latent information of neighbor points to learn richer features. This, combined with the leverage provided by the auxiliary losses, allow our network to work with real data and obtain the state of the art in the complex dataset S3DIS for 3D point cloud semantic segmentation. It also gets competitive results in the ShapeNetPart and ModelNet40 datasets.

As future work, this method can be implemented on forests and gardens to segment key elements in the environment like trees or plants. Also, our method can be trained and tested on our synthetic dataset of roses to segment bushes. However, this dataset is quite simple for point cloud segmentation. It is because the projected points of one image into the 3D world consist of the rose bush and part of the ground without any other object around it. Therefore, given that the network can easily segment plane sections, see Floor and Ceiling results from Table 6.1, it would be easy for the network to segment the rose bush from the ground.

Chapter 7

Conclusions

This thesis addressed the computer vision tasks of scene understanding, segmentation, and localization. The overall contributions of this thesis are methods that work efficiently and robustly in real environments and can be applied to real-time tasks, not only using 2D images but also for 3D data. The thesis also demonstrated that these properties make them useful for their application in autonomous driving, robotics, and agriculture.

The specific contributions are the following: A combination of end-to-end semantic segmentation and block matching disparity learning to obtain the 3D structure of objects, as well as the complex morphology of rose bushes [1] (Chapter 3). The second contribution extends the previous work. It not only obtains the reconstruction of the desired object but also finds key locations (cutting targets) and provides visual feedback. The feedback updates the positions of the key locations to counter the noise of the workflow. To the best of our knowledge, this is the first closed-loop rose pruning system [2] (Chapter 4). The third contribution presents an end-to-end solution using multitask learning to obtain the semantic segmentation and disparity estimation jointly, progressively and at different scales. Our method achieves competitive performance compared to the current approaches that predict each task individually, using only $1/3$ of the parameters (Chapter 5). Finally, our last contribution proposes a two-headed attention layer to aggregate the information of a patch of points inside a point cloud, using the geometric and latent features separately to segment and classify unordered point clouds. Processing these two pieces of information separately creates more robust features and leads to a better performance than current methods [3].

7.1 Thesis accomplishments and discussion

We presented algorithms that can be used in scene understanding. As explained in previous chapters, the data to be analyzed can be acquired from different sensors. Therefore, the proposed methods cover the two most common types of inputs: 2D images Chapters 3, 4, and 5 and 3D point clouds Chapter 6. Also, our methods were tested in complex environments, such as street views, gardens, and offices. The data contain a certain amount of noise as well. The 2D images had objects at different scales, occluded, with class variation, and under different light conditions. In the case of the point clouds, they not only had the previously mentioned noise but also a non-uniform density. It means that some regions of the points were densely populated while others had fewer points. These experiments demonstrated that our methods are robust and efficient.

7.1.1 Semantic segmentation and disparity estimation for reconstruction

The strategy presented in Chapter 3 combines classical disparity estimation and end-to-end semantic segmentation to improve the output of each other. This method is applied to the plant phenotyping domain (agriculture) because plants present complex structures that can validate the robustness and efficiency of our method. Here, the scene is represented as the image of a plant in a garden, and our proposed approach locates the position of the plant with respect to the camera and reconstructs the plant in 3D with its branching structure. The disparity is obtained using the classical BM algorithm. The branches are found by segmenting them out from the rest of the scene using our proposed network (Section 3.3.1), which is a small encoder-decoder with residual connections CNN. Both methods aim for fast segmentation and disparity estimation. BM is chosen over its CNN counterpart because it is faster and the regions without correspondence can be estimated by our refinement process (Section 3.3.3). A CNN was used for semantic segmentation because a classical approach is slow and performs badly in an unconstrained scenario. Also, even though our network is small, it obtains competitive results compared to state-of-the-art methods. Similar to the disparity estimation, semantic segmentation can be improved further thanks to the refinement process (Section 3.3.3).

To improve the segmentation, the disparity is binarized based on a distance thresh-

old. The resultant image is used as a mask, where the segmented branches that are separated but share the same masked region, are joined. Then, the segmented branch is used as a mask to estimate the disparity in the regions where the BM algorithm did not find any correspondence. This proposed method shows that it can improve the initial segmentation and disparity estimation. Then, the skeleton of the plant is obtained by using 2D skeletonization on the segmented branches, as seen in Section 3.3.5. The skeleton is found in 2D because the skeletonization methods preserve the connectivity of regions and reduce the binary image into a one-pixel width. These properties make searching for parts of the plant that belong to a branch easier because the pixels of a branch will only have two neighbors. Compared to methods that find the skeleton and branches in 3D, this approach does not have to find an optimal distance to label neighbor points into the same group. To the best of our knowledge, this is the first method that works without assumptions about the environment.
One problem of using a binary segmented bush as input for the 2D skeletonization is that if two branches are parallel and close enough, they will be segmented together, which will make the skeletonization algorithm fuse both branches. An interesting future work would be to use a CNN to learn a more robust skeletonization. It is an interesting area to explore because the output of the network should preserve the skeletonization properties.

Our first work proposed a network capable of segmenting plants under real environments (noisy and object with similar texture and color). Chapter 4 shows that the same network is useful for real-time robot navigation. The method proposed in the chapter is a visual servoing pipeline for a robotic arm that uses a disparity map and our semantic segmentation network to locate the branches in the 3D space, reconstruct the plant and locate cutting points in the branches. This method had two stages. The first one is used to scan the whole bush and find the prior locations of the cutting points (Section 4.3.2). The second stage uses the pipeline as visual feedback to update the locations of the cutting targets while the robotic arm is moving towards them (Section 4.3.5).

Back-projecting 2D images to the 3D space creates noisy point clouds and the prior target locations might be few centimeters off from their real location when the target is far from the camera position. The proposed visual feedback helps to reduce this noise by updating the targets in real-time. Also, considering that the robot arm gets closer to the target after each iteration, the projection noise is reduced, and the semantic segmentation network improves its performance because the branches in the image become more clear.

One missing part of our pipeline is an automatic evaluation to check if a branch was cut after sending the clipping command. Therefore, an extension of this work would be to include a post-cutting evaluation.

Also, our pipeline assumes that the robot is in front of the rose bush and scans it only from one side. Although this is an acceptable assumption for rose bushes, other types of plants are denser and wider. Thus, our method might not find all the cutting points. To make our approach more general, a possible future work would be to scan the plants from different sides and fuse the views.

Another future work would be to include more specific gardening rules, like cutting the branches above the eye-buds and cutting dead branches. The former is an interesting problem for object localization because the eye-buds are smaller and occupy few pixels in the image. The latter can be solved by extending our 3D morphological method to find dead branches.

Another interesting extension would be to evaluate the social impact that these type of devices have. For example, how it might affect the work of gardeners and related fields and whether or not this type of technology would replace them or create new technical jobs where they can switch to [239]. Also, it is important to evaluate the risk factor of having autonomous machines with sharp tools mounted on them. This would require to answer questions like: When is it considered safe to use this machine [240]? What is the the protocol it should be followed to create a safe environment for the people working around the robot? What would be the security threats and vulnerabilities for these type of robots [241]?

7.1.2 Multitask learning

The previous contributions combine classic and end-to-end approaches in a pipeline to find the desired objects in the environment and locate their position. Chapter 5 provides an end-to-end solution to jointly learn the semantic segmentation and disparity estimation using multi-task learning. Current methods use one pre-trained network as encoder and two decoders to predict both tasks, but this is expensive. Our approach predicts each task only using one decoder, at different scales, and in a progressive fashion. First, in Section 5.2.3 a coarse segmentation is predicted, then the disparity map is obtained in Section in Section 5.2.4. Because segmenting an image is a more difficult task, the network outputs a refined semantic segmentation using the feature information from the layers that output each task (Section 5.2.5). These features are combined by two self-attention mechanisms, which are in charge of selecting important features from each task. Thanks to this process, our network obtains competitive results against methods that solve only one task using only $1/3$ of the parameters. One possible extension would be to check if adding more tasks improves the segmentation and disparity estimation.

Due to the limited resources, we used DenseNet121 instead of the usual backbones used in the state-of-the-art segmentation networks like ResNet101 or Xception-71. Perhaps, using them might have improved the performance of our method.

7.1.3 3D scene understanding

One of the aims of this thesis is to address the different problems in 3D scene understanding. One of these problems is the semantic segmentation of unordered point clouds. Chapter 6 proposes an efficient method that evaluates patches inside point clouds and finds good relationships between neighbor points. Unlike current works that only use the learned features to segment the point cloud, our approach uses the geometric information given by the location of each point, and the latent features learned by the network. Therefore, the influence of one point towards its neighbors is measured by their geometric and latent relationship. This influence is obtained using two self-attention mechanisms; one for each information type (see Section 6.2.2). At the time the experiment was done, we obtained the state-of-the-art in one of the most difficult real datasets S3DIS. Aside from the segmentation, another advantage of our network is that we can reduce the number of heads of the self-attention mechanisms to make the network lighter, with only a small reduction in the accuracy as a side effect.

One problem of our method, and the current point cloud segmentation approaches, is that the network is highly dependant on the sampling method and the number of neighbors chosen in a patch. Recalling that the sampling method is used to reduce the number of points at every step of the encoder layer, one interesting future work would be to explore better sampling methods than FPS or use an efficient attention mechanism to choose the more representative points. The same can be said for a better choice of neighbors.

Appendix A

Multitask Learning Extended Ablation Study

A.1 Training process evaluation

This section includes some additional details about the training process with Cityscapes, such as the influence of different hyper-parameters, backbones and data augmentation process. Table A.1 shows the results in terms of IoU for semantic segmentation and D-1 error for disparity estimation.

Data augmentation. To train semantic segmentation networks, a common practice is to randomly resize the images to increase the variety of the dataset. However, to the best of our knowledge, it is not used when training disparity networks. The main reason is because the disparity changes when the images are zoomed in or out. Therefore, we introduce this data augmentation by using simple geometry. The disparity values should be re-scaled based on the zooming scale factor. The new disparity would be $disp_{new} = scale * disp_{old}$. This augmentation reduced the disparity error by 0.018 and improved the IoU in 1.4%. Applying random changes in the brightness of the input image improved the segmentation by 1.5% and the disparity error by 0.008. Finally, the class uniform sampling improved the segmentation IoU by 0.9% and disparity error by 0.016.

Input size and batch size. Table A.1 shows that a bigger input image gives better overall performance. This is due to the following reasons. First, the input size is based on the cropping around the original image of 2048×1024 px. This means that the bigger the image, the more context the network sees, which translates into learning better relationships between objects. Therefore, a smaller cropping size will be restrictive, especially because of the big size of the images in the dataset. The second reason is related to the reduction in dimension due to the backbone topology. In the case of ResNet and DenseNet, the backbones are divided into blocks, and each block reduces by half its input size. This means that the feature dimensions of the blocks 1 to 5 are $1/2$, $1/4$, $1/8$, $1/16$ and $1/32$ of the input image. Therefore, if we use an input size of 256×256 px the output feature from block 5 will be only 8×8.

Similar to other works that use backbones with batch normalization layers [206, 35], our experiments show that small training batches lead to unstable batch-norm statistics and to a poor learning performance. Table A.1 shows a progressive improvement in the disparity and segmentation when the batch size is increased. When the batch size is only 4 and the input size is 256×256 px, a significantly worse result is obtained, 66% IoU for the the segmentation and 0.353 error for the disparity. Our best result is obtained when using a batch size of 16 and an input size of 512×512 px. This configuration obtains a disparity error of 0.04 and a segmentation IoU of 77.6%.

A.2 Network details

This section complements the network details described in Chapter 5. Table A.2 shows the split of the backbone and the dimension of each output feature F_i. Tables A.3, A.4, and A.5 show the layer settings, inputs, and outputs of the coarse segmentation, disparity estimation, and refined segmentation branches, respectively. Each layer has a stride of 1 except for the SPP where the stride is the same size as their pooling window.

Input size	Backbone	Batch size	Random cropping	Zoom in/out	Brightness variation	Class sampling	Seg cr. IoU	Seg ref. IoU	Disp D-1 error
256x256	densenet121	4	✓				63.2	66.0	0.353
256x256	densenet121	8	✓				64.7	68.3	0.345
256x512	densenet121	8	✓				67.3	71.3	0.185
512x512	densenet121	16	✓				70.1	73.2	0.082
512x512	densenet121	16	✓	✓			71.7	74.6	0.064
512x512	densenet121	16	✓	✓	✓		73.0	76.1	0.056
512x512	densenet121	16	✓	✓	✓	✓	**74.6**	**77.6**	**0.040**
256x512	densenet121	32	✓	✓	✓	✓	70.1	73.3	0.092
512x512	efficientnet-b2	16	✓	✓	✓	✓	72.2	74.7	0.090
512x512	mobilenet-v3	16	✓	✓	✓	✓	69.1	72.0	0.124
256x512	resnet101	8	✓	✓	✓	✓	60.4	64.0	0.453
256x512	resnet50	8	✓	✓	✓	✓	66.6	70.0	0.173

Table A.1: Results using different hyper-parameters on Cityscapes validation set. The result evaluates the coarse segmentation *Seg cr.*, refined segmentation *Seg ref.* and disparity error D1-error.

Input	Layer settings	Output	
		Name	Dim
Left	Densenet121	F_1^L	$\frac{H}{4}$ x $\frac{W}{4}$ x 128
		F_2^L	$\frac{H}{8}$ x $\frac{W}{8}$ x 256
		F_4^L	$\frac{H}{32}$ x $\frac{W}{32}$ x 1024
Right	Densenet121	F_1^R	$\frac{H}{4}$ x $\frac{W}{4}$ x 128
		F_2^R	$\frac{H}{8}$ x $\frac{W}{8}$ x 256
		F_4^R	$\frac{H}{32}$ x $\frac{W}{32}$ x 1024

Table A.2: Backbone settings. It shows the input image (Left and Right) for each backbone and the features that are extracted. The size of each feature is shown with respect to the input size (height H and width W). The last dimension of the output represents the number of channels.

Input	Layer settings	Output
Left	conv2D$(5 \times 5, 1)$	L_conv_0
F_4^L F_4^R	concat	cat_0
cat_0	upscale $\times 2$ conv1D$(1 \times 1, 64)$	conv1d_0
conv1d_0	hourglass$(3 \times 3, 32)$	F_{seg}
F_{seg}	resize to L_conv_0 dim	convdec_0
L_conv_0 convdec_0	concat	cat_1
cat_1	conv1D$(1 \times 1, 32)$	conv1d_1
conv1d_1	hourglass$(3 \times 3, 32)$	convdec_1
convdec_1	conv2D$(3 \times 3, n_labels)$	conv2d_0
conv2d_0	resize to Left dim	coarse_seg

Table A.3: Coarse segmentation branch modules. For the convolutional layers conv_1d and conv_2d, the layer settings $k \times k, f$ represent the kernel size k and the number of filters f.

Input	Layer settings	Output
Left	conv2D$(5 \times 5, 1)$	L_conv_1
F_2^L	$\begin{bmatrix} \text{avgPool}(32,32), \text{conv2D}(3 \times 3, 32) \\ \text{avgPool}(16,16), \text{conv2D}(3 \times 3, 32) \\ \text{avgPool}(8,8), \text{conv2D}(3 \times 3, 32) \\ F_2^L \end{bmatrix}$	SPP_2^L
F_2^R	$\begin{bmatrix} \text{avgPool}(32,32), \text{conv2D}(3 \times 3, 32) \\ \text{avgPool}(16,16), \text{conv2D}(3 \times 3, 32) \\ \text{avgPool}(8,8), \text{conv2D}(3 \times 3, 32) \\ F_2^R \end{bmatrix}$	SPP_2^R
SPP_2^L SPP_2^R	correlation_layer	corr
corr	conv1D$(1 \times 1, 128)$	conv1d_2
F_{seg}	hourglass$(3 \times 3, 128)$ resize to conv1d_2 dim	convdec_2
conv1d_2 convdec_2	concat	cat_3
cat_3	hourglass$(3 \times 3, 64)$	F_{disp}
F_{disp}	resize to L_conv_1 dim	convdec_3
L_conv_1 convdec_3	concat	cat_4
cat_4	conv1D$(1 \times 1, 64)$	conv1d_3
conv1d_3	hourglass$(3 \times 3, 64)$	convdec_4
convdec_4	deconv2D$(5 \times 5, 1)$	deconv2d_0
deconv2D_0	resize to Left dim	disp

Table A.4: Disparity branch modules. The brackets in the SPP module indicate that the input was processed in 4 different ways and the result of each process was concatenated.

Input	Layer settings	Output
Left	conv2D($5 \times 5, 1$)	L_conv_2
F_1^L	$\begin{bmatrix} \text{avgPool}(64,64), \text{conv2D}(3 \times 3, 32) \\ \text{avgPool}(32,32), \text{conv2D}(3 \times 3, 32) \\ \text{avgPool}(16,16), \text{conv2D}(3 \times 3, 32) \\ \text{avgPool}(8,8), \text{conv2D}(3 \times 3, 32) \\ F_1^R \end{bmatrix}$	SPP_1^L
F_1^R	$\begin{bmatrix} \text{avgPool}(64,64), \text{conv2D}(3 \times 3, 32) \\ \text{avgPool}(32,32), \text{conv2D}(3 \times 3, 32) \\ \text{avgPool}(16,16), \text{conv2D}(3 \times 3, 32) \\ \text{avgPool}(8,8), \text{conv2D}(3 \times 3, 32) \\ F_1^R \end{bmatrix}$	SPP_1^R
SPP_1^L SPP_1^R	concat	cat_5
cat_5	$1 \times 1, 128$	conv1d_4
F_{seg}	resize to conv1d_4 dim	F_{seg}
F_{seg} conv1d_4	concat	cat_6
cat_6	hourglass($3 \times 3, 64$)	convdec_5
convdec_5	conv1D($1 \times 1, 1$), sigmoid() resize to F_{seg} dim	seg_at
F_{disp}	resize to conv1d_4 dim	F_{disp}
F_{disp} conv1d_4	concat	cat_7
cat_7	hourglass($3 \times 3, 64$)	convdec_6
convdec_6	conv1D($1 \times 1, 1$), sigmoid() resize to F_{disp} dim	disp_at
seg_at$*F_{seg}$ disp_at$*F_{disp}$ conv_1_4	concat	cat_8
cat_8	hourglass($3 \times 3, 64$) resize to L_conv_2 dim	convdec_7
L_conv_2 convdec_7	concat	cat_9
cat_9	hourglass($3 \times 3, 32$)	convdec_8
convdec_8	conv2D($3 \times 3, n\_labels$)	conv2d_1
conv2D_1	resize to L_conv_2 dim	ref_seg

Table A.5: Refined segmentation branch modules. The brackets in the *SPP* module follow the same notation as in Table A.4.

Bibliography

[1] Hanz Cuevas-Velasquez, Antonio-Javier Gallego, and Robert B Fisher. Segmentation and 3d reconstruction of rose plants from stereoscopic images. *Computers and electronics in agriculture*, 2020.

[2] Hanz Cuevas-Velasquez, Antonio-Javier Gallego, Radim Tylecek, Jochen Hemming, Bart Van Tuijl, Angelo Mencarelli, and Robert B Fisher. Real-time stereo visual servoing for rose pruning with robotic arm. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7050–7056. IEEE, 2020.

[3] Hanz Cuevas-Velasquez, Antonio-Javier Gallego, and Robert B Fisher. Two heads are better than one: Geometric-latent attention for point cloud classification and segmentation. In *32th British Machine Vision Conference, BMVC 2021*, 2021.

[4] Muzammal Naseer, Salman Khan, and Fatih Porikli. Indoor scene understanding in 2.5/3d for autonomous agents: A survey. *IEEE access*, 7:1859–1887, 2018.

[5] Towaki Takikawa, David Acuna, Varun Jampani, and Sanja Fidler. Gated-scnn: Gated shape cnns for semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5229–5238, 2019.

[6] Kun Zhou, Xiangxi Meng, and Bo Cheng. Review of stereo matching algorithms based on deep learning. *Computational intelligence and neuroscience*, 2020, 2020.

[7] Pier Luigi Dovesi, Matteo Poggi, Lorenzo Andraghetti, Miquel Martí, Hedvig Kjellström, Alessandro Pieropan, and Stefano Mattoccia. Real-time semantic stereo matching. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10780–10787. IEEE, 2020.

[8] Guorun Yang, Hengshuang Zhao, Jianping Shi, Zhidong Deng, and Jiaya Jia. Segstereo: Exploiting semantic information for disparity estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 636–651, 2018.

[9] Matthias Ochs, Adrian Kretz, and Rudolf Mester. Sdnet: Semantically guided depth estimation network. In *German Conference on Pattern Recognition*, pages 288–302. Springer, 2019.

[10] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6411–6420, 2019.

[11] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[12] Cahit Gürel, M Hassan G Zadeh, and Abdulkadir Erden. Rose stem branch point detection and cutting point location for rose harvesting robot. In *The 17th International Conference on Machine Design and Production, UMTIK 2016*, 2016.

[13] Tom Botterill, Scott Paulin, Richard Green, Samuel Williams, Jessica Lin, Valerie Saxton, Steven Mills, XiaoQi Chen, and Sam Corbett-Davies. A robot system for pruning grape vines. *Journal of Field Robotics*, 34(6):1100–1122, 2017.

[14] Cahit GÜrel, M Hassan GM Zadeh, and Abdulkadir Erden. Development and implementation of rose stem tracing using a stereo vision camera system for rose harvesting robot. In *8th International Conference on Image Processing, Wavelet and Applications (IWW 2016)*, 2016.

[15] Mohit Gupta, Qi Yin, and Shree K Nayar. Structured light in sunlight. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 545–552, 2013.

[16] Martin Thoma. A survey of semantic segmentation. *arXiv preprint arXiv:1602.06541*, 2016.

[17] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[18] Mehmet Celenk. A color clustering technique for image segmentation. *Computer Vision, Graphics, and image processing*, 52(2):145–170, 1990.

[19] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5):603–619, 2002.

[20] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[21] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

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

[23] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[24] Gabriella Csurka, Christopher Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision, ECCV*, volume 1, pages 1–2. Prague, 2004.

[25] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.

[26] Leo Grady. Random walks for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 28(11):1768–1783, 2006.

[27] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.

[28] Jesús Angulo and Dominique Jeulin. Stochastic watershed segmentation. In *ISMM (1)*, pages 265–276, 2007.

[29] Shoaib Ahmed Siddiqui, Ahmad Salman, Muhammad Imran Malik, Faisal Shafait, Ajmal Mian, Mark R Shortis, and Euan S Harvey. Automatic fish species classification in underwater videos: exploiting pre-trained deep neural network models to compensate for limited labelled data. *ICES Journal of Marine Science*, 75(1):374–389, 2018.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[31] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[34] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[35] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[36] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[37] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[38] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528, 2015.

[39] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[40] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[41] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[42] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters–improve semantic segmentation by global convolutional network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4353–4361, 2017.

[43] Reoxiang Li, Bing Zeng, and Ming L Liou. A new three-step search algorithm for block motion estimation. *IEEE transactions on circuits and systems for video technology*, 4(4):438–442, 1994.

[44] Lai-Man Po and Wing-Chung Ma. A novel four-step search algorithm for fast block motion estimation. *IEEE transactions on circuits and systems for video technology*, 6(3):313–317, 1996.

[45] Jo Yew Tham, Surendra Ranganath, Maitreya Ranganath, and Ashraf A Kassim. A novel unrestricted center-biased diamond search algorithm for block motion estimation. *IEEE transactions on Circuits and Systems for Video Technology*, 8(4):369–377, 1998.

[46] Yu-Wen Huang, Ching-Yeh Chen, Chen-Han Tsai, Chun-Fu Shen, and Liang-Gee Chen. Survey on block matching motion estimation algorithms and architectures with new results. *Journal of VLSI signal processing systems for signal, image and video technology*, 42(3):297–320, 2006.

[47] Yun-Qing Shi and X Xia. A thresholding multiresolution block matching algorithm. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(2):437–440, 1997.

[48] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4040–4048, 2016.

[49] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. End-to-end learning of geometry and context for deep stereo regression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 66–75, 2017.

[50] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5410–5418, 2018.

[51] Lidong Yu, Yucheng Wang, Yuwei Wu, and Yunde Jia. Deep stereo matching with explicit cost aggregation sub-architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[52] Feihu Zhang, Victor Prisacariu, Ruigang Yang, and Philip HS Torr. Ga-net: Guided aggregation net for end-to-end stereo matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 185–194, 2019.

[53] Zequn Jie, Pengfei Wang, Yonggen Ling, Bo Zhao, Yunchao Wei, Jiashi Feng, and Wei Liu. Left-right comparative recurrent model for stereo matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3838–3846, 2018.

[54] Zhengfa Liang, Yiliu Feng, Yulan Guo, Hengzhu Liu, Linbo Qiao, Wei Chen, Li Zhou, and Jianfeng Zhang. Learning deep correspondence through prior and posterior feature constancy. *arXiv preprint arXiv:1712.01039*, 7(8), 2017.

[55] Jiahao Pang, Wenxiu Sun, Jimmy SJ Ren, Chengxi Yang, and Qiong Yan. Cascade residual learning: A two-stage convolutional neural network for stereo

matching. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 887–895, 2017.

[56] Tingman Yan, Yangzhou Gan, Zeyang Xia, and Qunfei Zhao. Segment-based disparity refinement with occlusion handling for stereo matching. *IEEE Transactions on Image Processing*, 28(8):3885–3897, 2019.

[57] Feihu Zhang, Xiaojuan Qi, Ruigang Yang, Victor Prisacariu, Benjamin Wah, and Philip Torr. Domain-invariant stereo matching networks. In *European Conference on Computer Vision*, pages 420–439. Springer, 2020.

[58] Junyuan Xie, Ross Girshick, and Ali Farhadi. Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks. In *European conference on computer vision*, pages 842–857. Springer, 2016.

[59] Zhe Ren, Junchi Yan, Bingbing Ni, Bin Liu, Xiaokang Yang, and Hongyuan Zha. Unsupervised deep learning for optical flow estimation. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[60] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3712–3722, 2018.

[61] Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In *International Conference on Machine Learning*, pages 9120–9132. PMLR, 2020.

[62] Yu Zhang and Qiang Yang. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114*, 2017.

[63] Carlo Dal Mutto, Fabio Dominio, Pietro Zanuttigh, and Stefano Mattoccia. *Stereo Vision and Scene Segmentation*. IntechOpen, 2012.

[64] Majid Ezzati. Fast image segmentation using stereo vision. 1995.

[65] Rui Fan and Ming Liu. Road damage detection based on unsupervised disparity map segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 21(11):4906–4911, 2019.

[66] Yinghua Shen, Chaohui Lü, and Pin Xu. Stereoscopic video object segmentation based on disparity map. In *2010 International Conference on Measuring Technology and Mechatronics Automation*, volume 3, pages 493–495. IEEE, 2010.

[67] Chenxi Zhang, Liang Wang, and Ruigang Yang. Semantic segmentation of urban scenes using dense depth maps. In *European Conference on Computer Vision*, pages 708–721. Springer, 2010.

[68] Noppon Lertchuwongsa, Michele Gouiffes, and Bertrand Zavidovique. Enhancing a disparity map by color segmentation. *Integrated Computer-Aided Engineering*, 19(4):381–397, 2012.

[69] Vlad-Cristian Miclea and Sergiu Nedevschi. Real-time semantic segmentation-based stereo reconstruction. *IEEE Transactions on Intelligent Transportation Systems*, 21(4):1514–1524, 2019.

[70] Sébastien Drouyer, Serge Beucher, Michel Bilodeau, Maxime Moreaud, and Loïc Sorbier. Sparse stereo disparity map densification using hierarchical image segmentation. In *International symposium on mathematical morphology and its applications to signal and image processing*, pages 172–184. Springer, 2017.

[71] Zhile Ren, Deqing Sun, Jan Kautz, and Erik Sudderth. Cascaded scene flow prediction using semantic segmentation. In *2017 International Conference on 3D Vision (3DV)*, pages 225–233. IEEE, 2017.

[72] Xiao Lin, Dalila Sánchez-Escobedo, Josep R Casas, and Montse Pardàs. Depth estimation and semantic segmentation from a single rgb image using a hybrid convolutional neural network. *Sensors*, 19(8):1795, 2019.

[73] Wujing Zhan, Xinqi Ou, Yunyi Yang, and Long Chen. Dsnet: Joint learning for scene segmentation and disparity estimation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2946–2952. IEEE, 2019.

[74] Thomas Leimkühler, Hans-Peter Seidel, and Tobias Ritschel. Minimal warping: Planning incremental novel-view synthesis. In *Computer Graphics Forum*, volume 36, pages 1–14. Wiley Online Library, 2017.

[75] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pages 2650–2658, 2015.

[76] Iasonas Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6129–6138, 2017.

[77] Vladimir Nekrasov, Thanuja Dharmasiri, Andrew Spek, Tom Drummond, Chunhua Shen, and Ian Reid. Real-time joint semantic segmentation and depth estimation using asymmetric annotations. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7101–7107. IEEE, 2019.

[78] Zhenyu Zhang, Zhen Cui, Chunyan Xu, Zequn Jie, Xiang Li, and Jian Yang. Joint task-recursive learning for semantic segmentation and depth estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 235–251, 2018.

[79] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[80] Mathieu Aubry, Ulrich Schlickewei, and Daniel Cremers. The wave kernel signature: A quantum mechanical approach to shape analysis. In *2011 IEEE international conference on computer vision workshops (ICCV workshops)*, pages 1626–1633. IEEE, 2011.

[81] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In *Computer graphics forum*, volume 28, pages 1383–1392. Wiley Online Library, 2009.

[82] Michael M Bronstein and Iasonas Kokkinos. Scale-invariant heat kernel signatures for non-rigid shape recognition. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1704–1711. IEEE, 2010.

[83] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Aligning point cloud views using persistent feature histograms. In *2008*

IEEE/RSJ international conference on intelligent robots and systems, pages 3384–3391. IEEE, 2008.

[84] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE international conference on robotics and automation*, pages 3212–3217. IEEE, 2009.

[85] Haibin Ling and David W Jacobs. Shape classification using the inner-distance. *IEEE transactions on pattern analysis and machine intelligence*, 29(2):286–299, 2007.

[86] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, 2018.

[87] Hsien-Yu Meng, Lin Gao, Yu-Kun Lai, and Dinesh Manocha. Vv-net: Voxel vae net with group convolutions for point cloud segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8500–8508, 2019.

[88] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3577–3586, 2017.

[89] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (TOG)*, 36(4):1–11, 2017.

[90] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 863–872, 2017.

[91] Tianjia Shao, Yin Yang, Yanlin Weng, Qiming Hou, and Kun Zhou. H-cnn: spatial hashing based cnn for 3d shape analysis. *IEEE transactions on visualization and computer graphics*, 26(7):2403–2416, 2018.

[92] Takumi Fujiwara, Shin Akatsuka, Ryosuke Kaneko, and Masataka Takagi. Construction method of voxel model and the application for agro-forestry. *Internet Journal of Society for Social Management Systems, Vol. sms17*, 2017.

[93] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017.

[94] Jiaxin Li, Ben M Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9397–9406, 2018.

[95] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019.

[96] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. *Advances in neural information processing systems*, 31:820–830, 2018.

[97] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent convolutions for dense prediction in 3d. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3887–3896, 2018.

[98] Yiqun Lin, Zizheng Yan, Haibin Huang, Dong Du, Ligang Liu, Shuguang Cui, and Xiaoguang Han. Fpconv: Learning local flattening for point convolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4293–4302, 2020.

[99] Jingwei Huang, Haotian Zhang, Li Yi, Thomas Funkhouser, Matthias Nießner, and Leonidas J Guibas. Texturenet: Consistent local parametrizations for learning from high-resolution signals on meshes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4440–4449, 2019.

[100] Yuqi Yang, Shilin Liu, Hao Pan, Yang Liu, and Xin Tong. Pfcnn: convolutional neural networks on 3d surfaces using parallel frames. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13578–13587, 2020.

[101] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[102] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.

[103] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu. Pct: Point cloud transformer. *arXiv preprint arXiv:2012.09688*, 2020.

[104] Shi Qiu, Saeed Anwar, and Nick Barnes. Semantic segmentation for real point cloud scenes via bilateral augmentation and adaptive fusion. *arXiv preprint arXiv:2103.07074*, 2021.

[105] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. Exploring self-attention for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10076–10085, 2020.

[106] Pasi Raumonen, Mikko Kaasalainen, Markku Åkerblom, Sanna Kaasalainen, Harri Kaartinen, Mikko Vastaranta, Markus Holopainen, Mathias Disney, and Philip Lewis. Fast automatic precision tree models from terrestrial laser scanner data. *Remote Sensing*, 5(2):491–520, 2013.

[107] Yifei Shi, Pinxin Long, Kai Xu, Hui Huang, and Yueshan Xiong. Data-driven contextual modeling for 3d scene understanding. *Comput. Graph.*, 55:55–67, 2016.

[108] Gabriel L Oliveira, Wolfram Burgard, and Thomas Brox. Efficient deep models for monocular road segmentation. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4885–4891. IEEE, 2016.

[109] Henrique Oliveira and Paulo Lobato Correia. Road surface crack detection: improved segmentation with pixel-based refinement. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pages 2026–2030. IEEE, 2017.

[110] Ziyu Jiang, Zhenhua He, Xueqin Huang, Zibin Yang, and Pearl Tan. Ce-peopleseg: Real-time people segmentation with 10% cpu usage for video conference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 914–922, 2021.

[111] Mengye Ren and Richard S Zemel. End-to-end instance segmentation with recurrent attention. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6656–6664, 2017.

[112] Zhang-Wei Hong, Chen Yu-Ming, Shih-Yang Su, Tzu-Yun Shann, Yi-Hsiang Chang, Hsuan-Kung Yang, Brian Hsi-Lin Ho, Chih-Chieh Tu, Yueh-Chuan Chang, Tsu-Ching Hsiao, et al. Virtual-to-real: Learning to control in visual semantic segmentation. *arXiv preprint arXiv:1802.00285*, 2018.

[113] Minjie Hua, Yibing Nan, and Shiguo Lian. Small obstacle avoidance based on rgb-d semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[114] Kentaro Wada, Kei Okada, and Masayuki Inaba. Joint learning of instance and semantic segmentation for robotic pick-and-place with heavy occlusions in clutter. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9558–9564. IEEE, 2019.

[115] Nicola Strisciuglio, Radim Tylecek, Michael Blaich, Nicolai Petkov, Peter Biber, Jochen Hemming, Eldert van Henten, Torsten Sattler, Marc Pollefeys, Theo Gevers, et al. Trimbot2020: an outdoor robot for automatic gardening. In *ISR 2018; 50th International Symposium on Robotics*, pages 1–6. VDE, 2018.

[116] AR Jimenez, R Ceres, and Jose L Pons. A survey of computer vision methods for locating fruit on trees. *Transactions of the ASAE*, 43(6):1911, 2000.

[117] GUILING GL SUN, RUOBIN RB WANG, CHENG QIAN, FAN ZHOU, and SIRUI SR WANG. Research and realization of crop instance segmentation based on yolact. In *2021 3rd International Conference on Management Science and Industrial Engineering*, pages 13–20, 2021.

[118] Helin Dutagaci, Pejman Rasti, Gilles Galopin, and David Rousseau. Rose-x: an annotated data set for evaluation of 3d plant organ segmentation methods. *Plant methods*, 16(1):1–14, 2020.

[119] G. Flandin, F. Chaumette, and E. Marchand. Eye-in-hand/eye-to-hand cooperation for visual servoing. In *Proc. ICRA*, volume 3, pages 2741–2746, 2000.

[120] Moslem Kazemi, Kamal Gupta, and Mehran Mehrandezh. Path-Planning for Visual Servoing: A Review and Issues. In *Visual Servoing via Advanced Numerical Methods*, pages 189–207. 2010.

[121] Pankaj Kumar, Jinhai Cai, and Stan Miklavcic. High-throughput 3d modelling of plants for phenotypic analysis. In *Proceedings of the 27th conference on image and vision computing New Zealand*, pages 301–306, 2012.

[122] Noha M Elfiky, Shayan A Akbar, Jianxin Sun, Johnny Park, and Avinash Kak. Automation of dormant pruning in specialty crop production: An adaptive framework for automatic reconstruction and modeling of apple trees. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 65–73, 2015.

[123] Dejan Kaljaca, Nikolaus Mayer, Bastiaan Vroegindeweij, Angelo Mencarelli, Eldert van Henten, and Thomas Brox. Automated boxwood topiary trimming with a robotic arm and integrated stereo vision. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[124] Nikolaus Correll, Nikos Arechiga, Adrienne Bolger, Mario Bollini, Ben Charrow, Adam Clayton, Felipe Dominguez, Kenneth Donahue, Samuel Dyar, Luke Johnson, et al. Indoor robot gardening: design and implementation. *Intelligent Service Robotics*, 3(4):219–232, 2010.

[125] Ruud Barth, Jochen Hemming, and Eldert J van Henten. Design of an eye-in-hand sensing and servo control framework for harvesting robotics in dense vegetation. *Biosystems Engineering*, 146:71–84, 2016.

[126] Duke M Bulanon, Colton Burr, Marina DeVlieg, Trevor Braddock, and Brice Allen. Development of a visual servo system for robotic fruit harvesting. *AgriEngineering*, 3(4):840–852, 2021.

[127] R Ceres, Jose L Pons, AR Jimenez, JM Martin, and L Calderon. Design and implementation of an aided fruit-harvesting robot (agribot). *Industrial Robot: An International Journal*, 1998.

[128] Biqing Li, Aizhen Zhou, Chongjun Yang, and Shiyong Zheng. The design and realization of fruit harvesting robot based on iot. In *2016 International Conference on Computer Engineering, Information Science & Application Technology (ICCIA 2016)*, pages 158–161. Atlantis Press, 2016.

[129] Yuki Onishi, Takeshi Yoshida, Hiroki Kurita, Takanori Fukao, Hiromu Arihara, and Ayako Iwai. An automated fruit harvesting robot by using deep learning. *Robomech Journal*, 6(1):1–8, 2019.

[130] Zhao De-An, Lv Jidong, Ji Wei, Zhang Ying, and Chen Yu. Design and control of an apple harvesting robot. *Biosystems engineering*, 110(2):112–122, 2011.

[131] J Hemming, CW Bac, BAJ van Tuijl, Ruud Barth, Jan Bontsema, EJ Pekkeriet, and Eldert Van Henten. A robot for harvesting sweet-pepper in greenhouses. 2014.

[132] Thiago Teixeira Santos, Luciano Vieira Koenigkan, Jayme Garcia Arnal Barbedo, and Gustavo Costa Rodrigues. 3d plant modeling: localization, mapping and segmentation for plant phenotyping using a single hand-held camera. In *European Conference on Computer Vision*, pages 247–263. Springer, 2014.

[133] Hanz Cuevas-Velasquez, Nanbo Li, Radim Tylecek, Marcelo Saval-Calvo, and Robert B Fisher. Hybrid multi-camera visual servoing to moving target. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1132–1137. IEEE, 2018.

[134] Dominik Honegger, Torsten Sattler, and Marc Pollefeys. Embedded real-time multi-baseline stereo. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5245–5250. IEEE, 2017.

[135] Nicola Strisciuglio, Radim Tylecek, Nicolai Petkov, Peter Bieber, Jochen Hemming, Eldert van Henten, Torsten Sattler, Marc Pollefeys, Theo Gevers, Thomas Brox, and Robert B. Fisher. Trimbot2020: an outdoor robot for automatic gardening. In *50th International Symposium on Robotics*. VDE Verlag GmbH - Berlin - Offenbach, 2018.

[136] Can Pu, Nanbo Li, Radim Tylecek, and Bob Fisher. Dugma: Dynamic uncertainty-based gaussian mixture alignment. In *3DV 2018*, 2018.

[137] Johannes L Schönberger, Marc Pollefeys, Andreas Geiger, and Torsten Sattler. Semantic visual localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6896–6906, 2018.

[138] Jorge Calvo-Zaragoza and Antonio-Javier Gallego. A selectional auto-encoder approach for document image binarization. *Pattern Recognition*, 86:37 – 47, 2019.

[139] Jan Hackenberg, Christopher Morhart, Jonathan Sheppard, Heinrich Spiecker, and Mathias Disney. Highly accurate tree models derived from terrestrial laser scan data: A method description. *Forests*, 5(5):1069–1105, 2014.

[140] Yotam Livny, Feilong Yan, Matt Olson, Baoquan Chen, Hao Zhang, and Jihad El-Sana. Automatic reconstruction of tree skeletal structures from point clouds. *ACM Transactions on Graphics (TOG)*, 29(6):151, 2010.

[141] David Belton, Simon Moncrieff, and Jane Chapman. Processing tree point clouds using gaussian mixture models. *Proceedings of the ISPRS annals of the photogrammetry, remote sensing and spatial information sciences, Antalya, Turkey*, pages 11–13, 2013.

[142] Hui Huang, Shihao Wu, Daniel Cohen-Or, Minglun Gong, Hao Zhang, Guiqing Li, and Baoquan Chen. L1-medial skeleton of point cloud. *ACM Trans. Graph.*, 32(4):65–1, 2013.

[143] Amy Tabb and Henry Medeiros. A robotic vision system to measure tree traits. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6005–6012. IEEE, 2017.

[144] Liying Zheng, Jingtao Zhang, and Qianyu Wang. Mean-shift-based color segmentation of images containing green vegetation. *Computers and Electronics in Agriculture*, 65(1):93–98, 2009.

[145] Liying Zheng, Daming Shi, and Jingtao Zhang. Segmentation of green vegetation of crop canopy images based on mean shift and fisher linear discriminant. *Pattern Recognition Letters*, 31(9):920–925, 2010.

[146] R Barth, J IJsselmuiden, J Hemming, and EJ Van Henten. Data synthesis methods for semantic segmentation in agriculture: A capsicum annuum dataset. *Computers and Electronics in Agriculture*, 144:284–296, 2018.

[147] R Barth, J IJsselmuiden, J Hemming, and EJ Van Henten. Synthetic bootstrapping of convolutional neural networks for semantic plant part segmentation. *Computers and Electronics in Agriculture*, 2017.

[148] Takahiro Isokane, Fumio Okura, Ayaka Ide, Yasuyuki Matsushita, and Yasushi Yagi. Probabilistic plant modeling via multi-view image-to-image translation. *arXiv preprint arXiv:1804.09404*, 2018.

[149] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks.

[150] Anthony Paproki, Xavier Sirault, Scott Berry, Robert Furbank, and Jurgen Fripp. A novel mesh processing based technique for 3d plant analysis. *BMC plant biology*, 12(1):63, 2012.

[151] Kyle Simek, Ravishankar Palanivelu, and Kobus Barnard. Branching gaussian processes with applications to spatiotemporal reconstruction of 3d trees. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conference on Computer Vision – ECCV 2016*, pages 177–193, Cham, 2016. Springer International Publishing.

[152] William Gélard, Michel Devy, Ariane Herbulot, and Philippe Burger. Model-based segmentation of 3d point clouds for phenotyping sunflower plants. In *12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP 2017)*, volume 4, pages 459–467, 2017.

[153] Guillem Alenya, Babette Dellen, Sergi Foix, and Carme Torras. Robotized plant probing: Leaf segmentation utilizing time-of-flight data. *IEEE Robotics & Automation Magazine*, 20(3):50–59, 2013.

[154] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, June 2015.

[155] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing.

[156] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015.

[157] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[158] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *JMLR W&CP*, 37, 2015.

[159] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, January 2014.

[160] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. *Journal of Machine Learning Research (JMLR) W&CP*, 15:315–323, 2011.

[161] James Bergstra and Yoshua Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[162] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[163] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[164] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances In Neural Information Processing Systems*, pages 1–9, 2012.

[165] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. In *British Machine Vision Conference*, pages 1–11, 2014.

[166] Kurt Konolige. Small vision systems: Hardware and implementation. In Yoshiaki Shirai and Shigeo Hirose, editors, *Robotics Research*, pages 203–212, London, 1998. Springer London.

[167] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341, Feb 2008.

[168] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical

flow, and scene flow estimation. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. arXiv:1512.02134.

[169] Morgan Quigley, Josh Faust, Tully Foote, and Jeremy Leibs. Ros: an open-source robot operating system.

[170] Paul Furgale, Joern Rehder, and Roland Siegwart. Unified temporal and spatial calibration for multi-sensor systems. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1280–1286. IEEE, 2013.

[171] TY Zhang and Ching Y Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984.

[172] Zicheng Guo and Richard W Hall. Parallel thinning with two-subiteration algorithms. *Communications of the ACM*, 32(3):359–373, 1989.

[173] Ta-Chih Lee, Rangasami L Kashyap, and Chong-Nam Chu. Building skeleton models via 3-d medial surface axis thinning algorithms. *CVGIP: Graphical Models and Image Processing*, 56(6):462–478, 1994.

[174] Harry Blum. A transformation for extracting new descriptors of shape. In Weiant Wathen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, Cambridge, 1967.

[175] Nicola Strisciuglio, George Azzopardi, and Nicolai Petkov. Robust inhibition-augmented operator for delineation of curvilinear structures. *Ieee transactions on image processing*, 2019.

[176] M. T. Orchard and C. A. Bouman. Color quantization of images. *IEEE Transactions on Signal Processing*, 39(12):2677–2690, Dec 1991.

[177] L.G. Shapiro and G.C. Stockman. *Computer Vision*. Prentice Hall, 2001.

[178] Nicola Strisciuglio, George Azzopardi, and Nicolai Petkov. Brain-inspired robust delineation operator. In Laura Leal-Taixé and Stefan Roth, editors, *Computer Vision – ECCV 2018 Workshops*, pages 555–565, Cham, 2019. Springer International Publishing.

[179] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings IJCAI*, volume 2 of *IJCAI'95*, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[180] Luai Al Shalabi, Zyad Shaaban, and Basel Kasasbeh. Data Mining: A Preprocessing Engine. *Journal of Computer Science*, 2(9):735–739, 2006.

[181] Tinku Acharya and Ajoy K. Ray. *Image Processing - Principles and Applications*. Wiley-Interscience, New York, NY, USA, 2005.

[182] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017.

[183] Gong Cheng and Junwei Han. A survey on object detection in optical remote sensing images. *{ISPRS} Journal of Photogrammetry and Remote Sensing*, 117:11 – 28, 2016.

[184] Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[185] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2366–2374, Cambridge, MA, USA, 2014. MIT Press.

[186] Qin Zou, Yu Cao, Qingquan Li, Qingzhou Mao, and Song Wang. Cracktree: Automatic crack detection from pavement images. *Pattern Recognition Letters*, 33(3):227–238, 2012.

[187] Srinivas C Turaga, Kevin L Briggman, Moritz Helmstaedter, Winfried Denk, and H Sebastian Seung. Maximin affinity learning of image segmentation. *arXiv preprint arXiv:0911.5372*, 2009.

[188] Li Bangyu and Xu Fang. Outdoor grass target recognition based on gabor filter. In *2017 IEEE 7th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 432–435. IEEE, 2017.

[189] AO Adeodu, IA Daniyan, TS Ebimoghan, and SO Akinola. Development of an embedded obstacle avoidance and path planning autonomous solar grass cutting robot for semi-structured outdoor environment. In *2018 IEEE 7th International Conference on Adaptive Science & Technology (ICAST)*, pages 1–11. IEEE, 2018.

[190] Bosch. Accessed on 19/07/2019. [online]. Available: `https://www.bosch-garden.com/gb/en/garden-tools/indego-home.jsp`.

[191] Agrobot. Accessed on 19/07/2019. [online]. Available: `http://agrobot.com/`.

[192] Rabiul Hossen Rafi, Shuva Das, Nawsher Ahmed, Iftekhar Hossain, and SM Taslim Reza. Design & implementation of a line following robot for irrigation based application. In *2016 19th International Conference on Computer and Information Technology (ICCIT)*, pages 480–483. IEEE, 2016.

[193] Scott Paulin, Tom Botterill, Jessica Lin, X Chen, and R Green. A comparison of sampling-based path planners for a grape vine pruning robot arm. In *2015 6th International Conference on Automation, Robotics and Applications (ICARA)*, pages 98–103. IEEE, 2015.

[194] Scott Paulin, Tom Botterill, X Chen, and Richard Green. A specialised collision detector for grape vines.

[195] Tom Botterill, Richard Green, and Steven Mills. Finding a vine's structure by bottom-up parsing of cane edges. In *2013 28th International Conference on Image and Vision Computing New Zealand (IVCNZ 2013)*, pages 112–117. IEEE, 2013.

[196] Peilin Li, Sang-heon Lee, and Hung-Yao Hsu. Review on fruit harvesting method for potential use of automatic fruit harvesting systems. *Procedia Engineering*, 23:351–366, 2011.

[197] C Wouter Bac, Eldert J van Henten, Jochen Hemming, and Yael Edan. Harvesting robots for high-value crops: State-of-the-art review and challenges ahead. *Journal of Field Robotics*, 31(6):888–911, 2014.

[198] Kinova Robotics. Accessed on 19/07/2019. [online]. Available: `http://www.kinovarobotics.com`.

[199] Mitchell Wills. Accessed on 19/07/2019. [online]. Available: `http://wiki.ros.org/epos_hardware`.

[200] Jérôme Maye, Paul Furgale, and Roland Siegwart. Self-supervised calibration for robotic systems. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 473–480. IEEE, 2013.

[201] Eddy Ilg, Tonmoy Saikia, Margret Keuper, and Thomas Brox. Occlusions, motion and depth boundaries with a generic network for disparity, optical flow or scene flow estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 614–630, 2018.

[202] Jorge Calvo-Zaragoza and Antonio-Javier Gallego. A selectional auto-encoder approach for document image binarization. *Pattern Recognition*, 86:37 – 47, 2019.

[203] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise.

[204] Sachin Chitta, Ioan Sucan, and Steve Cousins. Moveit![ros topics]. *IEEE Robotics Automation Magazine - IEEE ROBOT AUTOMAT*, 19:18–19, 03 2012.

[205] Hamid Laga. A survey on deep learning architectures for image-based depth reconstruction. *arXiv preprint arXiv:1906.06113*, 2019.

[206] Shijie Hao, Yuan Zhou, and Yanrong Guo. A brief survey on semantic segmentation with deep learning. *Neurocomputing*, 2020.

[207] Zhelun Shen, Yuchao Dai, and Zhibo Rao. Msmd-net: Deep stereo matching with multi-scale and multi-dimension cost volume. *arXiv preprint arXiv:2006.12797*, 2020.

[208] Andrew Tao, Karan Sapra, and Bryan Catanzaro. Hierarchical multi-scale attention for semantic segmentation. *arXiv preprint arXiv:2005.10821*, 2020.

[209] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018.

[210] Bowen Cheng, Maxwell D Collins, Yukun Zhu, Ting Liu, Thomas S Huang, Hartwig Adam, and Liang-Chieh Chen. Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12475–12485, 2020.

[211] Yuhui Yuan, Xilin Chen, and Jingdong Wang. Object-contextual representations for semantic segmentation. *arXiv preprint arXiv:1909.11065*, 2019.

[212] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[213] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[214] Radim Tylecek, Torsten Sattler, Hoang-An Le, Thomas Brox, Marc Pollefeys, Robert B. Fisher, and Theo Gevers. The second workshop on 3d reconstruction meets semantics: Challenge results discussion. In Laura Leal-Taixé and Stefan Roth, editors, *ECCV 2018 Workshops*, pages 631–644, Cham, 2019. Springer International Publishing.

[215] Vitor Guizilini, Rui Hou, Jie Li, Rares Ambrus, and Adrien Gaidon. Semantically-guided representation learning for self-supervised monocular depth. *arXiv preprint arXiv:2002.12319*, 2020.

[216] Po-Yi Chen, Alexander H Liu, Yen-Cheng Liu, and Yu-Chiang Frank Wang. Towards scene understanding: Unsupervised monocular depth estimation with semantic-aware representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2624–2632, 2019.

[217] Heliang Zheng, Jianlong Fu, Tao Mei, and Jiebo Luo. Learning multi-attention convolutional neural network for fine-grained image recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 5209–5217, 2017.

[218] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[219] Maxim Berman, Amal Rannen Triki, and Matthew B Blaschko. The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-

union measure in neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4413–4421, 2018.

[220] Gregory Koch. Siamese neural networks for one-shot image recognition. 2015.

[221] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1925–1934, 2017.

[222] Rowel Atienza. Fast disparity estimation using dense networks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3207–3212. IEEE, 2018.

[223] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[224] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[225] Sungha Choi, Joanne T Kim, and Jaegul Choo. Cars can't fly up in the sky: Improving urban-scene segmentation via height-driven attention networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9373–9383, 2020.

[226] Ivan Kreso, Sinisa Segvic, and Josip Krapac. Ladder-style densenets for semantic segmentation of large natural images. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 238–245, 2017.

[227] Marcela Carvalho, Maxime Ferrera, Alexandre Boulch, Julien Moras, Bertrand Le Saux, and Pauline Trouvé-Peloux. Technical report: Co-learning of geometry and semantics for online 3d mapping. *arXiv preprint arXiv:1911.01082*, 2019.

[228] Caner Hazirbas, Lingni Ma, Csaba Domokos, and Daniel Cremers. Fusenet: Incorporating depth into semantic segmentation via fusion-based cnn architecture. In *Asian conference on computer vision*, pages 213–228. Springer, 2016.

[229] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient large-scale stereo matching. In *Asian conference on computer vision*, pages 25–38. Springer, 2010.

[230] K. Miettinen. *Nonlinear multiobjective optimization*. Kluwer Academic Publishers, Boston, 1999.

[231] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[232] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[233] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1534–1543, 2016.

[234] Li Yi, Vladimir G Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics (ToG)*, 35(6):1–12, 2016.

[235] Lyne Tchapmi, Christopher Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. In *2017 international conference on 3D vision (3DV)*, pages 537–547. IEEE, 2017.

[236] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019.

[237] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11108–11117, 2020.

[238] Tiange Xiang, Chaoyi Zhang, Yang Song, Jianhui Yu, and Weidong Cai. Walk in the cloud: Learning curves for point clouds shape analysis. *arXiv preprint arXiv:2105.01288*, 2021.

[239] Juan P Vasconez, George A Kantor, and Fernando A Auat Cheein. Human–robot interaction in agriculture: A survey and current challenges. *Biosystems engineering*, 179:35–48, 2019.

[240] Paul Baxter, Grzegorz Cielniak, Marc Hanheide, and Pål From. Safe human-robot interaction in agriculture. In *Companion of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, pages 59–60, 2018.

[241] Jean-Paul A Yaacoub, Hassan N Noura, Ola Salman, and Ali Chehab. Robotics cyber security: vulnerabilities, attacks, countermeasures, and recommendations. *International Journal of Information Security*, pages 1–44, 2021.