

AMERICAN UNIVERSITY OF BEIRUT

Mapping through a Stereo Rig: Identifying
Ground Patches and Obstacles

by

Mahmoud Mohamad Ali Hamandi

A thesis

submitted in partial fulfillment of the requirements
for the degree of Master of Engineering
to the Department of Mechanical Engineering
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
April 2017

AMERICAN UNIVERSITY OF BEIRUT

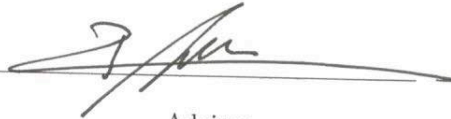
Mapping through a Stereo Rig: Identifying
Ground Patches and Obstacles

by

Mahmoud Mohamad Ali Hamandi

Approved by:

Dr. Daniel Asmar, Associate Professor
Mechanical Engineering



Advisor

Dr. Elie Shammass, Assistant Professor
Mechanical Engineering



Member of Committee

Dr. Naseem Daher, Assistant Professor
Electrical and Computer Engineering



Member of Committee

Date of thesis defense: April 13, 2017

AMERICAN UNIVERSITY OF BEIRUT

THESIS, DISSERTATION, PROJECT
RELEASE FORM

Student Name: Hamandi Mahmoud Mohamad Ali
Last First Middle

Master's Thesis Master's Project Doctoral Dissertation

I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

I authorize the American University of Beirut, **three years after the date of submitting my thesis, dissertation, or project**, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.


Signature

19-April-2017
Date

Acknowledgements

First of all I am thankful for Almighty Allah for providing me with the opportunity to start my masters, and taught me the skills I needed to finish it, whilst providing me with the needed companionship, resources and power when I needed it the most.

I am deeply grateful to my advisor and mentor Professor Daniel Asmar, for guiding me throughout my masters years, providing me with his insight whenever I am stuck and bearing with me throughout. Along with his academic advice, one cannot forget the motivation and support he insisted on providing us with.

I would also like to thank both Professor Elie Shamma and Professor Naseem Daher for pushing me forward with their on point questions and comments.

I thank my fellow labmates, who have provided me their support throughout my year and a half, in a place I sincerely call home. I thank Ali Harakeh for his work upon which I built this thesis. I thank Salah Bazy for being our big brother in the lab, and providing us with all the guidance and help he had provided. I would like to thank Bilal Hammoud for all the coffee breaks, and companionship. Lastly, I cannot thank enough my best companions, Abed AlRahman AlMakdah, and Rahaf Rahal, I could not have imaged my stay in the lab without both of you.

Finally, I am infinitely thankful for my parents and brothers for their ever lasting physical and mental support, and for pushing me to be the best I can in this life and the hereafter. May I always make you proud.

An Abstract of the Thesis of

Mahmoud Mohamad Ali Hamandi for Master of Engineering
Major: Mechanical Engineering

Title: Mapping through a Stereo Rig: Identifying Ground Patches and Obstacles

Mapping the space about an autonomous agent is one of the preliminary steps for any sort of navigation or task planning. In this work, stereo rig output is employed to accomplish this task, where ground patches, near ground obstacles and background are delineated in one of the stereo images, in a self-supervised method. In a first method, free space is estimated using a novel technique: first depth information extracted from a stereo camera is used to identify, with high certainty, regions that belong to ground or obstacle clutter. Next, interactive graph cuts is used to propagate the initial segmentation across the entire image to yield an estimate of the location of free space in each image, while considering all obstacles and background as one object. This novel method is proved to outperform state of the art in the literature; however, to mitigate the effect of local noise in the disparity image, and reduce the runtime of the system, the self supervised classifier is used to train a ground segmentation deep learning algorithm, which is fine-tuned online when the system is faced with new ground. The system detects the change in ground using a novel image clustering technique, and by comparing its output against a weak classifier.

Contents

Acknowledgements	iv
Abstract	v
List of Figures	viii
List of Tables	ix
1 Introduction	1
2 Literature Review	3
3 Background	5
3.1 v -disparity and u -disparity transformations	5
3.2 Positive Naïve Bayes framework	6
3.3 Convolutional Neural Networks	7
3.3.1 Convolutional Layers	8
3.3.2 Multilayer Perceptron	10
3.3.3 Backpropagation	11
4 System Details	13
4.1 Free Space estimation using Graph Cuts	13
4.1.1 v -disparity and Training data	14
4.1.2 u -disparity and identifying obstacles	15
4.1.3 Ground Classification and filtering	19
4.1.4 Interactive Graph Cuts	19
4.1.5 Unary Terms	20
4.1.6 Binary Terms	21
4.1.7 Seeds	21
4.2 Deep Learning Techniques	21
4.2.1 Deep Clustering	22
4.2.2 Adaptive Learning	27

5 Experiments and Results	29
5.1 Dataset	29
5.1.1 Easy Dataset	29
5.1.2 Moderate Dataset	29
5.1.3 Difficult Dataset	30
5.2 Benchmarking	30
5.3 DCN	30
5.4 Analysis	34
5.4.1 FSGC obstacle constraint and runtime	34
5.4.2 Free Parameters	37
5.4.3 Runtime	39
5.4.4 Comparision	40
6 Conclusion and Future Work	45
Bibliography	47

List of Figures

3.1	transformation from (u, v, d) to (v, d, s) and (u, d, l)	6
3.2	example showing a convolutional operation, with a 2x2 kernel, and 1 pixel stride	9
3.3	example showing maxpooling, with a 2x2 filter, and 2 pixels stride	9
3.4	example showing a) a fully connected two layer perceptron network, b) a locally connected two layer perceptron network.	11
4.1	diagram of Free Space estimation using Graph Cuts algorithm	13
4.2	image showing result of u -diaprity algorithm from the literature	15
4.3	details of my obstacle detection algorithm	18
4.4	interactive graph cuts for a 3x3 image	20
4.5	Deep Clustering Ground Segmentation	23
4.6	DCN training Data	26
4.7	Adaptive Learning Segmentation system	28
5.1	DCN first iteration	31
5.2	DCN fourth iteration	32
5.3	DCN seventh iteration	33
5.4	FSGC precision comparison graph	36
5.5	ROC for varying γ and κ	38
5.6	Precision vs Recall system comparison	41
5.7	final FSGC segmentation examples	43
5.8	segmentation comparision examples	44

List of Tables

4.1	DCNs CNN architecture	24
5.1	FSGC comparision table	35
5.2	runtime comparison table	39
5.3	performance comparison table	40

Chapter 1

Introduction

With the era of autonomous vehicles about us, scene understanding is becoming ever more important. For autonomous navigation, one of the most fundamental scene understanding tasks is to delineate free space from obstacles; thereby allowing path planning for the autonomous agent.

Supervised training algorithms are thriving in the field of segmentation and object detection [1], [2], [3], and offer a very appealing solution to many computer vision problems. However, because of the variety of environments a vehicle might travel in, another appealing alternative is the self-supervised approach, where one sensor guides another, or one classification method supervises a second method, without any human interaction. The absence of both protracted training time and supervised human input renders this last approach more suitable for ground segmentation.

Although other sensors are shown to be effective in the literature, I focus on ground segmentation using stereo cameras because of the large bandwidth of data they offer, their relatively small size, low power consumption, and relatively low prices. However, it should be noted that the proposed system can work with a monocular camera, paired with a device providing depth information, such as LIDARs. In addition, most systems in the literature only exploit stereo disparity data, and disregard intensity information. However, the extent of the range that a stereo camera can capture is limited by the baseline between its stereo pair. On the other hand intensity information is reliable for objects that extend to infinity in a scene; therefore, integrating the intensity information into my solution can expand the detection span beyond what the stereo data can accomplish alone.

It is in this spirit that my system is designed; my intent is to segment smooth and coherent free space patches from obstacles and background in one of the stereo image pairs, supervised by training data extracted from the stereo sensor input. My approach is multi-tiered; starting with a series of classifiers to identify, in each image, a number of pixels that represent free space and others that represent obstacles. Interactive Graph Cuts (IGC) [4] is then implemented, using the labeled pixels in the first stage as seeds, to find a smooth free space segmentation. Then

a deep learning algorithm is trained with the acquired segmentation to perform its own semantic ground segmentation. The trained network is fine-tuned when required, based on metrics calculated by two proposed techniques; the first represents the change in image appearance, and the other the depreciated network performance.

The main contributions in this thesis are as follows:

- A novel ground segmentation algorithm that employs IGC, with unary terms and seeds designed specifically for the application.
- A deep learning technique that clusters images based on their color representation, and detects new representations in new images.
- An adaptive learning technique based on the decay of network performance.
- A variant of a deep learning ground segmentation technique, that is fine-tuned based on the change in ground appearance, or the decay in network performance.
- Evaluation of the proposed techniques, and comparison against the most relevant self-supervised techniques in the literature.

The remainder of this thesis is structured as follows: Chapter 2 summarizes the most relevant ground segmentation techniques in the literature, with emphasis on self-supervised methods. Chapter 3 provides background information to provide context for my work. Chapter 4 provides detailed steps of the first part of my system, referred to as Free Space estimation using Graph Cuts (FSGC), along with the obstacle detection algorithm, and finally the deep learning techniques employed for ground segmentation, and the consequent deep clustering and adaptive learning techniques. Chapter 5 describes the experiments that were conducted to validate the mentioned systems, along with analysis and comparison of my results with those from three other algorithms from the literature. Finally, Chapter 6 concludes this thesis, and presents possible future work.

Chapter 2

Literature Review

Previous work on ground segmentation and free space estimation employ a variety of sensors, such as LIDARs, and radars. 3D LIDARs were used in Sugar *et al.* [5] in a semi-supervised free space estimation algorithm, where a human input constitutes the training data, and is used to learn the occupancy grid of the environment. Dahlkamp *et al.* [6] employed a 2D LIDAR to extract ground training data and a color-based classifier is learned to segment ground in a monocular image. Millela *et al.* [7] use a radar to extract ground training patches, and then classify ground in a monocular image; unfortunately, their approach fails when segmenting distant ground patches, mistakenly segmenting some of them as obstacles.

Stereo cameras constitute a very attractive alternative sensor for free space estimation, because of their relatively low cost, and the high bandwidth of information they offer. Labayrade *et al.* [8] used a stereo camera to identify ground patches, by extracting the ground in the v -disparity space. The transformation to the v -disparity space consists of calculating the frequency of each disparity at each v (meaning image row height) location. Labayrade noticed that a relatively flat ground plane shows up as a salient slanted line in the v -disparity image, and was able to identify ground pixels based on their closeness to this line in the v -disparity space.

Later, Harakeh *et al.* [9], [10] further studied the structure in the v -disparity to learn a probabilistic model of the ground, by using Bayesian linear regression to model the ground plane in both structured and unstructured environments. While learning the parameters of the model, the authors identify free-space training data, which is used to train a support vector classifier. Harakeh *et al.* classify ground with high precision; however, their system can only classify pixels featuring reliable disparity, and labels all others as obstacles, thereby risking many false negatives for free space.

Kim *et al.* [11] and Vernaza *et al.* [12] assumed the ground to be the largest plane around the vehicle. While this assumption works well on flat scenes, it fails on off-road scenes. Oniga *et al.* [13] used the u -disparity space to identify obstacles.

Similar to the v -disparity, the u -disparity consists of calculating the frequency of each disparity at each u (meaning image column) location. Oniga *et al.* then filter the u -disparity to identify obstacles in the scene. Although their method is mathematically sound, it fails to generalize to the dataset I used, providing weak obstacle classification, and a subsequent weaker ground segmentation.

Recently, Brust *et al.* [14] proposed a new technique to train a convolutional neural network for pixel-wise ground segmentation. The proposed technique processes each patch of the image through a convolutional neural network(CNN) architecture to classify its central pixel. This method has the advantage that each pixel is a training item on its own, thus for a given labeled image, one would have training examples equal to its pixel count. In addition, Brust *et al.* noticed that the pixel position is a strong prior for the labeling, and as such incorporated the spatial prior as an input to the fully connected layers of the CNN.

Although Brust *et al.* [14] showed promising results, their algorithm requires supervised human interaction in the form of hand labeled training data. As a solution, Sanberg *et al.* [15] extracted a self-supervised weak classifier to train the network. In [15] they proved that online training of the network with the weak classifier can outperform both the weak classification and the offline trained network. However, their method requires online training of the network for each input image, after extracting its weak classifier, thus increasing the system's runtime. In addition, the weak classifier they used is The Stixel World proposed by Badino *et al.* [16], which requires more than two seconds to extract on a *CPU*, also increasing runtime per image.

In summary, the weaknesses of the state-of-the-art are either required human supervision, extensive processing time, or processing only disparity data while disregarding intensity information, all of which I intend to avoid in this work.

Chapter 3

Background

In this chapter I present some of the background information necessary to understand the remainder of this work. Mainly I will focus on the extraction of *u-disparity* and *v-disparity* maps from a *u,v-disparity* map, as well as explaining the Positive Naïve Bayes classifier, and presenting a brief explanation about Convolutional Neural Networks.

3.1 *v-disparity* and *u-disparity* transformations

At the beginning of each segmentation iteration, each captured stereo pair is processed and a corresponding disparity map is produced; then, a 3D point cloud corresponding to this disparity map is estimated, using the cameras' focal lengths, the baseline distance between the two stereo cameras, and the disparity value at each pixel location. The assumption here is that both cameras possess comparable intrinsic parameters and that they are calibrated offline. Furthermore, since my system also relies on appearance, the intensity values in one of two images is also recorded.

Once the disparity map (u, v, d) is obtained, I calculate the *v-* and *u-*disparity maps, which represent the frequencies (s) and (l) of occurrence of each disparity value at each row v and column u , respectively. Figure 3.1 shows an example of how this mapping occurs.

Mathematically, the transformation H from disparity space (u, v, d) to *v-disparity* space (v, d, s) is formulated as:

$$H(u, v = i, d = \Delta) = (i, \Delta, s = \sum_{u=\min}^{\max} d == \Delta). \quad (3.1)$$

Similarly, the transformation Q from disparity space (u, v, d) to *u-disparity* space (u, d, l) is formulated as:

$$Q(u = i, v, d = \Delta) = (i, \Delta, l = \sum_{v=\min}^{\max} d == \Delta), \quad (3.2)$$

where d stands for disparity, and Δ is a specific disparity value.

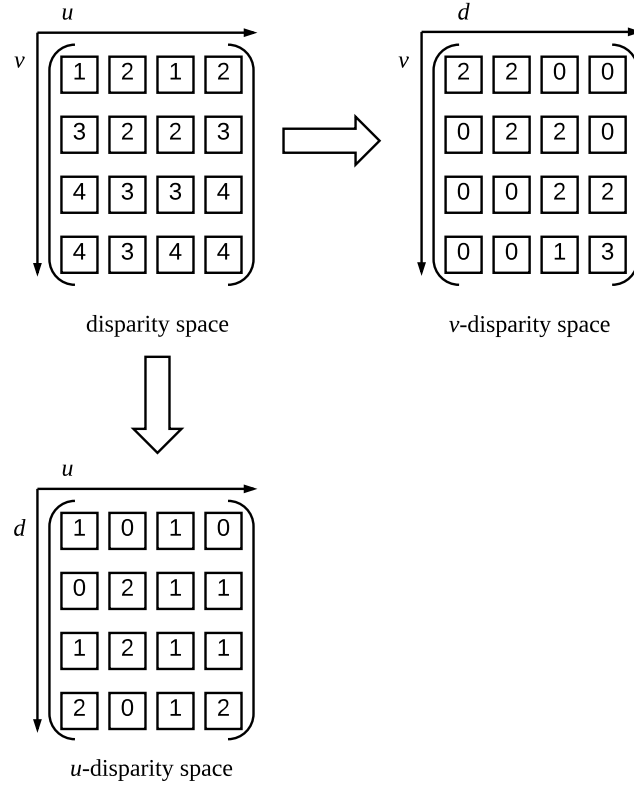


Figure 3.1: transformation from (u, v, d) to (v, d, s) and (u, d, l)

3.2 Positive Naïve Bayes framework

The Positive Naïve Bayes (PNB) [17] is a classification method that estimates the probability $P_{i=1 \rightarrow N}(label_i | featureVector)$, then chooses the label with the highest conditional probability. The probabilities are calculated using Bayes rule as:

$$P(label_i | FV) = \eta P(label_i) P(FV | label_i), \quad (3.3)$$

where FV is the feature vector and η is a normalizing factor, which is ignored in PNB as it is unchanging between labels. By assuming independence between the features, the joint probability can be reduced to a product, and (3.3) becomes:

$$P(label_i | FV) = \eta P(label_i) \prod_{j=1}^M P(f_j | label_i). \quad (3.4)$$

It is assumed here that each component of the feature vector lies in a strictly positive feature space such that $f_j \in [1, \dots, K] \forall j \in [1, \dots, M]$, where K is the total

number of discrete values a feature can take and M is the total number of features in the feature vector. Let the functions $C(f_i, S)$ and $C(S)$ represent two counting functions, where the first one is used to calculate the number of occurrences of feature i in set S , and the second one is used to calculate the number of elements in S . In the system at hand, the two labels include free space and obstacle. PNB estimates $P(f_i|free)$ as:

$$P(f_i|free) = \frac{\zeta + C(f_i, TP)}{\zeta \times M \times K + C(TP)}, \quad (3.5)$$

where TP are the training pixels, and ζ is a smoothing parameter. In case no training data for obstacles exist, (3.5) cannot be used to find $P(f_i|obstacle)$, and must be estimated from the law of total probability as:

$$P(f_i) = P(f_i|obstacle)P(obstacle) + P(f_i|free)P(free). \quad (3.6)$$

Then:

$$P(f_i|obstacle) = \frac{P(f_i) - P(f_i|free)P(free)}{P(obstacle)}. \quad (3.7)$$

Using the notation of UD being the unlabeled pixels, the probability of a feature is then expressed as $P(f_i) = C(f_i, UD)/C(UD)$. Finally, substituting (3.5) into (3.7), and applying Laplace smoothing as done in [18] yields:

$$P(f_i|obstacle) = \frac{1 + Q}{M \times K + (1 - P(free))C(UD)}, \quad (3.8)$$

where

$$Q = \max(0, C(f_i, UD) - P(free)P(f_i|free)C(UD)). \quad (3.9)$$

The max function was added by He *et al.* [18] to ensure a non negative probability. The estimation of the prior $P(free)$ warrants a comment; while one could use the resulting classification of each frame as a prior to the subsequent frame, this heuristic risks propagating erroneous classifications in scenes where the shape of the ground plane changes quickly. It was therefore decided to initialize each new frame with priors of equal values between free and occupied (*i.e.* $P(free) = P(occupied) = 0.5$).

Once both $P(f_i|obstacle)$ and $P(f_i|free)$ are calculated, (3.3) is used to determine the state of each pixel as being an obstacle or free.

3.3 Convolutional Neural Networks

In this section I will briefly explain some of the building blocks of a Convolutional Neural Network (CNN), its operation, and training process. This chapter is not intended to provide a comprehensive study of CNNs.

CNNs, as we know them today, were introduced by LeCun *et al.* in 1989 in [19]

and [20], as hand written digit classification networks, which can be trained to classify an image after extracting its features. Later, in 1998, LeCun *et al.* [21] presented another network architecture known as LeNet, trained to classify hand written letters. As such, a CNN architecture has three main components: the convolutional layers responsible of feature extraction, the connected layers known as Multilayer Perceptron (MLP) responsible of decision making, and the backpropagation algorithm used for training.

In this work I am only interested in simple CNNs, where no internal branches exist in the network, nor outputs can be input at a layer other than the first layer, as opposed to Directed Acyclic Graphs (DAGs).

3.3.1 Convolutional Layers

Convolutional operators have been used in many computer vision applications for feature extraction [22], as a matrix operation between a Kernel K of dimensions $m \times n$, and an input image as shown in Fig.3.2. Mathematically the operation can be written as follows:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n) \quad (3.10)$$

where $S(i, j)$ is the output of the convolution operation at pixel (i, j) , and S is referred to as a feature map.

Classical convolutional operations in computer vision have fixed kernels, thus performing specific tasks such as blurring, edge detection, or smoothing. In CNN, features change from one application to the other, thus the kernel K is initialized at random, and then learned in the process. Each convolutional layer might have multiple kernels, producing multiple feature maps. Each feature map can be followed by a maxpooling layer, where only the maximum value within a rectangular neighborhood is reported, as shown in Fig.3.3. Also, each feature map can be followed by an activation function. The order of the maxpooling layer and the activation layer can vary from one architecture to another. Finally, the number of convolutional layers, number of feature maps per layer, kernel size and parameters, maxpooling size and parameters, and activation functions are specific for each architecture and application.

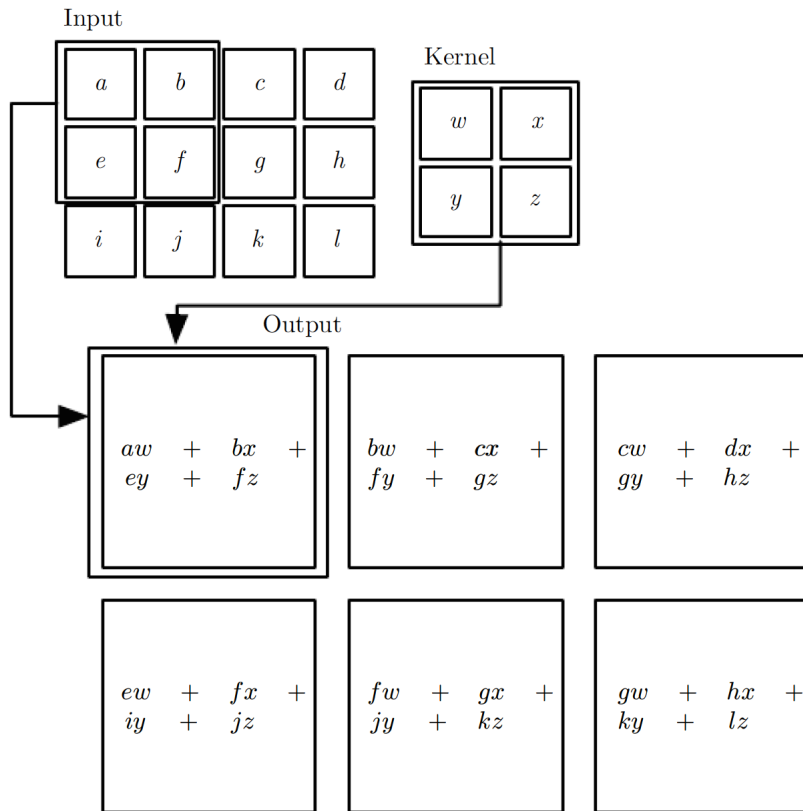


Figure 3.2: Convolutional operation example, with a 2x2 kernel, and 1 pixel stride; this image is excerpted from [23]

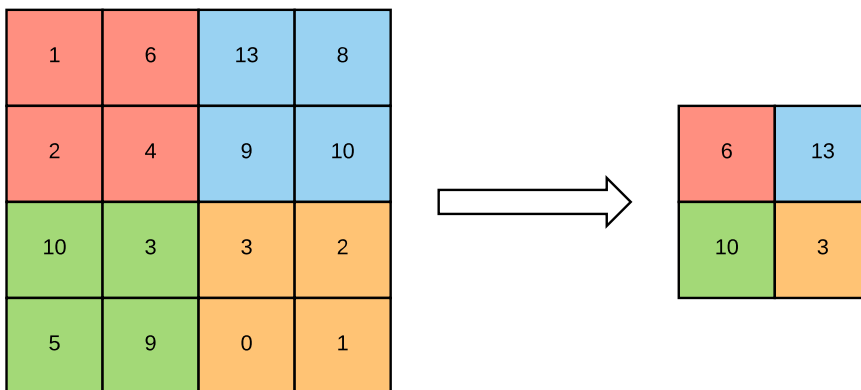


Figure 3.3: example showing maxpooling, with a 2x2 filter, and 2 pixels stride

3.3.2 Multilayer Perceptron

Multilayer Perceptrons are the basic building blocks in any deep learning algorithm [23], where multiple layers of perceptrons are connected, each one with its preceding layer. The depth of the network is proportional to the number of hidden layers, situated between the first layer, named the input layer, and the last layer named the output layer. In a CNN architecture, an MLP follows the convolutional layers whose output is considered as input to the MLP. MLPs are also named Feedforward networks as the output of one layer is only processed by the layers that follow, without any feedback loops. Each perceptron has the output of some or all the perceptrons in the preceding layer as input, and its output can be formulated as $y_j = f(\sum_i w_{ij}x_i + b_j)$, where $f(x)$ is the activation function between the current and preceding layer, y_j is the output of perceptron j in this layer with bias b_j , w_{ij} are the weights connecting perceptron j in the current layer with each corresponding perceptron i in the preceding layer, with output x_i .

Each perceptron can be fully connected to its previous layer as shown in Fig.3.4a, or locally connected to n of its preceding neighbors as shown in Fig.3.4b, with the connection option being decided by the architecture and application.

Some of the activation functions used in the literature are:

- Linear unit $f(x) = x$
- Sigmoid function $f(x) = \frac{1}{1+e^x}$
- Softmax function $f(x) = \frac{e^{x_i}}{\sum_j e^{x_j}}$, where the summation is over all perceptrons of a layer.
- Hyperbolic Tangent $f(x) = \tanh(x)$
- Rectified Linear Unit (ReLU) $f(x) = \max(0, x)$

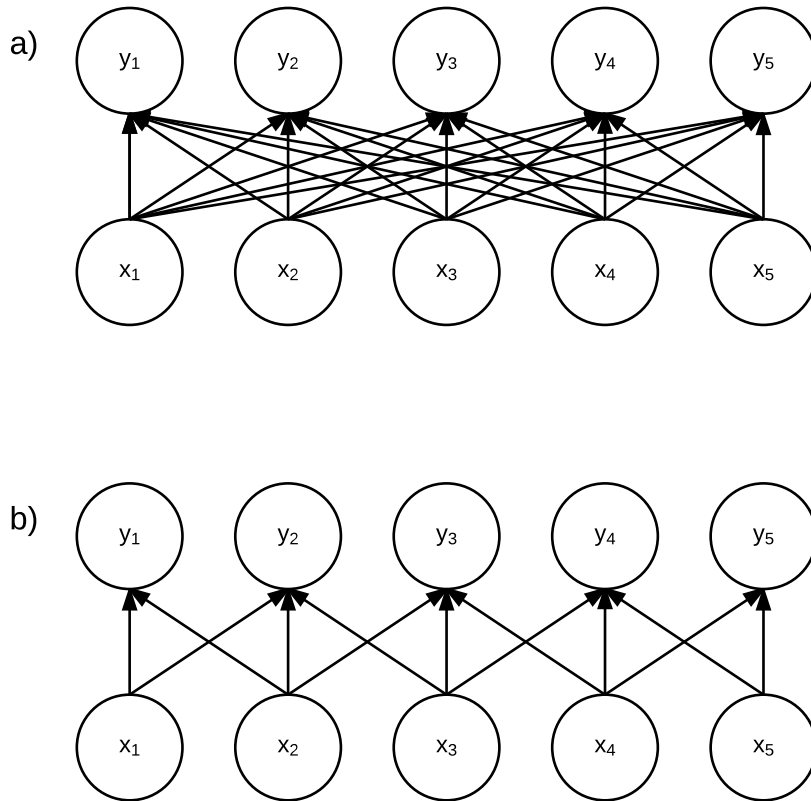


Figure 3.4: example showing a) a fully connected two layer perceptron network, b) a locally connected two layer perceptron network.

3.3.3 Backpropagation

Training of a CNN is similar to many supervised machine learning algorithms, where at each iteration, variable parameters are changed in the direction that reduces the error between the output of the network and the training label. In CNNs, and MLPs in general, this is known as backpropagation since the error from the last layer is propagated to the first layer to correct all parameters of the network, as opposed to forward propagation where the input is propagated to the last layer.

Mathematically, given a cost function C , input x , and desired output y , after each forward propagation iteration, the network calculates the output of the network \hat{y} , based on the input x and the current weights and biases. Then it calculates $C(y, \hat{y})$, and $\nabla_{w_k} C$ for all weights in the network, and $\nabla_{b_j} C$ for all biases in the

network. The gradient computation is a simple element-wise matrix multiplication [23]. Then once the gradients are computed, each weight or bias is updated as follows:

$$\begin{cases} w_k \leftarrow w_k - \alpha \nabla_{w_k} C \\ b_j \leftarrow b_j - \alpha \nabla_{b_j} C \end{cases} \quad (3.11)$$

where alpha is the learning rate, set manually. This approach is usually referred to as gradient descent in optimization theory.

Usually with large datasets, a stochastic gradient descent algorithm is applied, as it is difficult to calculate the cost for all inputs. As such, the training algorithm stochastically picks a batch of training examples from the dataset at each iteration, and optimizes the network accordingly, reducing the runtime of each iteration.

Although gradient descent and stochastic gradient descent are used in many deep learning techniques, there exists other training algorithms that will not be covered here.

Chapter 4

System Details

4.1 Free Space estimation using Graph Cuts

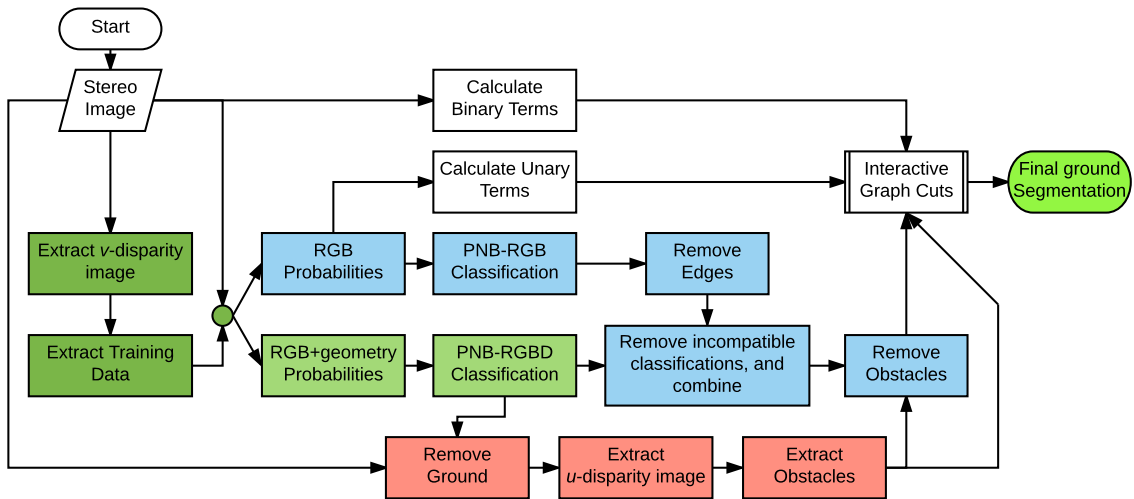


Figure 4.1: image showing a diagram summarizing the steps of FSGC

Figure 4.1 presents a breakdown of the system proposed as FSGC. First, for each input image, I extract a corresponding v -disparity image, which is then processed to extract ground training data, in a manner similar to the method proposed in Harakeh *et al.* [10], in which a small percentage of pixels, possessing reliable disparity values, are adopted as training data. Then, classification is conducted via three stages: two stages that rely on a Positive Naïve Bayes (PNB) classifier, and one that relies on (IGC) [4]. The first PNB classifier is trained with the color values of the ground training data and then, once trained, it is used to label the remaining pixels in the image based on their color. The second PNB classifier is applied to all pixels for which the disparity calculation was reliable, only this time both color and geometry are included in the feature vector of the

PNB. In addition, using what is known as a u -disparity mapping, the resulting segmentation from both PNB classifications is further processed, and accordingly, pixels identified as obstacles have their labels corrected. Finally, IGC is used to yield a final smooth classification for the entire image.

4.1.1 v -disparity and Training data

The classification of ground pixels in the v -disparity space is done as follows: first, the frequency of each disparity occurrence at each v -value is computed. Next, the v -disparity image is processed to extract the line that represents, in the v -disparity space, the ground plane. In order to attenuate vertical lines, the v -disparity image is filtered using a horizontal Sobel edge detector. Then, the image is binarized, thereby resulting in a line representing ground, in addition to noise, which is removed using a binary area opening operation.

At this point in my solution, the segmentation of the ground plane is treated as a stochastic process, where ground pixels in the v -disparity image are modeled as belonging to a Gaussian distribution with the ground line, determined above, as its mean. Although in perfectly planar roads, pixels belonging to ground fit very well to a straight line in the v -disparity space, in off-road scenes the ground plane is not flat and pixels belonging to it no longer fit well to a line; rather, a second degree polynomial with additive Gaussian noise is a better fit. Accordingly, the disparity of points on the ground plane is expressed as:

$$d = w_0 + w_1\nu + w_2\nu^2 + \varepsilon = \mathbf{w}^T \phi^*(\nu) + \varepsilon. \quad (4.1)$$

Then by consequence:

$$P(d|\nu, \mathbf{w}, \beta) = \mathcal{N}(d; \mathbf{w}^T \phi^*(\nu), \beta^{-1}). \quad (4.2)$$

By assuming the pixels to be independent, the joint probability of all disparity values of ground pixels can be reduced to the product of the conditional probabilities of all pixels. Then, as shown in (4.3), the probability of the vector \mathbf{d} of all disparities that belong to ground, conditioned on their corresponding row values—concatenated into the vector \mathbf{v} —, and parameters of the modeled line from (4.1), I get:

$$P(\mathbf{d}|\mathbf{v}, \mathbf{w}, \beta) = \prod_{n=1}^N \mathcal{N}(d_n; \mathbf{w}^T \phi^*(\nu_n), \beta^{-1}), \quad (4.3)$$

where $\mathbf{w} = [w_0 w_1 w_2]$ represents the parameters of the ground line in the v -disparity space, ε is a Gaussian random variable with mean zero, and precision β , $\phi^*(\nu_n) = [\nu_n^0 \nu_n^1 \nu_n^2]$, and (d_n, ν_n) are the v -d values of each pixel in the image. A prior distribution on \mathbf{w} is assumed as: $P(\mathbf{w}|\alpha) = \mathcal{N}(0, \alpha^{-1}I)$.

In [10], the authors proposed a method to learn the two parameters α and β , and consequently compute the Gaussian distribution of the v -disparity line. Once the distribution is computed, all pixels that are within a confidence interval of the distributions mean are considered as good training data for ground segmentation.

4.1.2 u -disparity and identifying obstacles

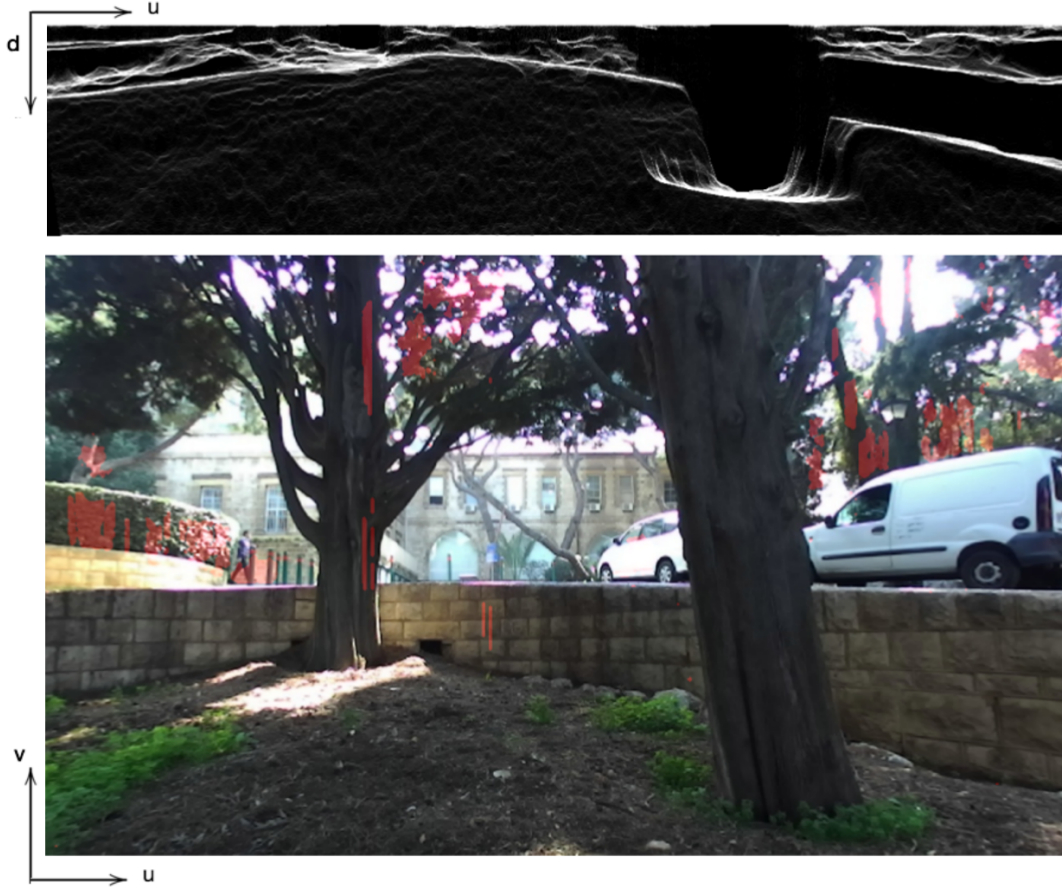


Figure 4.2: an example showing a colored image with the final filtered obstacles in red overlaid (bottom) using Oniga *et al.* [13]; the corresponding raw u -disparity image(top)

Obstacles in outdoor scenes can be difficult to detect [24]. However, in u -disparity space such obstacles show up as salient bright regions (See Fig. 4.2 top); knowledge of their location can significantly assist in the overall segmentation and delineation of free space. Oniga *et al.* [13] suggest the following method to adaptively filter the u -disparity image for all obstacles that are above a certain height y_{3dmin} . By replacing the canonical expression for depth in a stereo rig $Z = b \times f/d$ into the denominator of the equation of a pinhole camera model $y_{min}/f = y_{3dmin}/Z$, a threshold y_{min} , which is dependent on the disparity, is obtained:

$$y_{min} = \frac{y_{3dmin}}{b}d. \quad (4.4)$$

Here, b is the baseline between the stereo pair, f is the focal length, d is the disparity, Z is the depth, y_{3dmin} is the min 3D obstacle height. This filtering stage

transforms the u -disparity image into a binary image and removes all obstacles smaller than y_{3dmin} . In the u -disparity images, obstacles are reflected by clusters of points, which are in some cases dense and large and in other cases sparse and small. My experiments revealed that during the ensuing graph cuts stage, a decrease in classification performance is observed when the small and sparse clusters are not removed. These clusters over-constrain graph cuts, while not significantly contributing to the classification of obstacles. Therefore, I remove them using a smoothing morphological binary operation. The u -disparity is then mapped back to the original image to mark the obstacle patches. Finally, in an effort to regularize the segmentation, any pixel near an obstacle, with similar disparity, is also classified as an obstacle (see Fig. 4.2 bottom). Although Oniga *et al.* shows the advantage of this algorithm over those in the literature, its application to the dataset I am using provided sparse obstacle detection in most images, similar to Fig. 4.2 bottom. An effective obstacle segmentation is essential for the ensuing FSGC, thus I present another obstacle segmentation algorithm that takes advantage of my preliminary ground segmentation PNB-RGBD presented section 4.1.3.

My algorithm aims to find, in each image column, the pixels that have a high probability of corresponding to obstacles; the coordinates of these pixels are later used as seeds for the growing of obstacles in the ensuing interactive graph cuts segmentation discussed below. These candidate seeds are identified as the pixels whose u -disparity values are within a threshold γ of the maximum u -disparity in each column; the threshold γ is determined in a systematic manner as explained in the experimental section below.

The assumption made in here is that, in each column, an obstacle starts where free space ends; furthermore, all obstacles are assumed to be relatively vertical in orientation, with negligible relative depth, the implication of which is that the u -disparity values for the same obstacle are approximately the same. Therefore, all pixels classified as obstacles and located before the limiting region of free space are removed. Note that, at this point, free space was delineated using Positive Naive Bayes with RGBD data, the details of which are explained below.

Fig.4.3 illustrates the results of my obstacle detection algorithm: in the top image, ground pixels (pixels before the end of the ground segmentation) are colored in blue, and other areas—obstacles or background—are colored in red; the image in the middle shows the u -disparity of all non-ground regions. Finally, in the image at the bottom, pixels that are classified as obstacles with high certainty, are colored in red.

It should be noted that closer obstacles have a higher pixel representation per unit length, thus in general they have higher u -disparity values; thus, most obstacles detected by my algorithm are the closest to ground, and provide a better constraint in the ensuing IGC.

For convenience, I will note FSGC employing Oniga *et al.*'s obstacle detection algorithm as FSGC-O, while the one employing my obstacle detection algorithm

as FSGC-U.

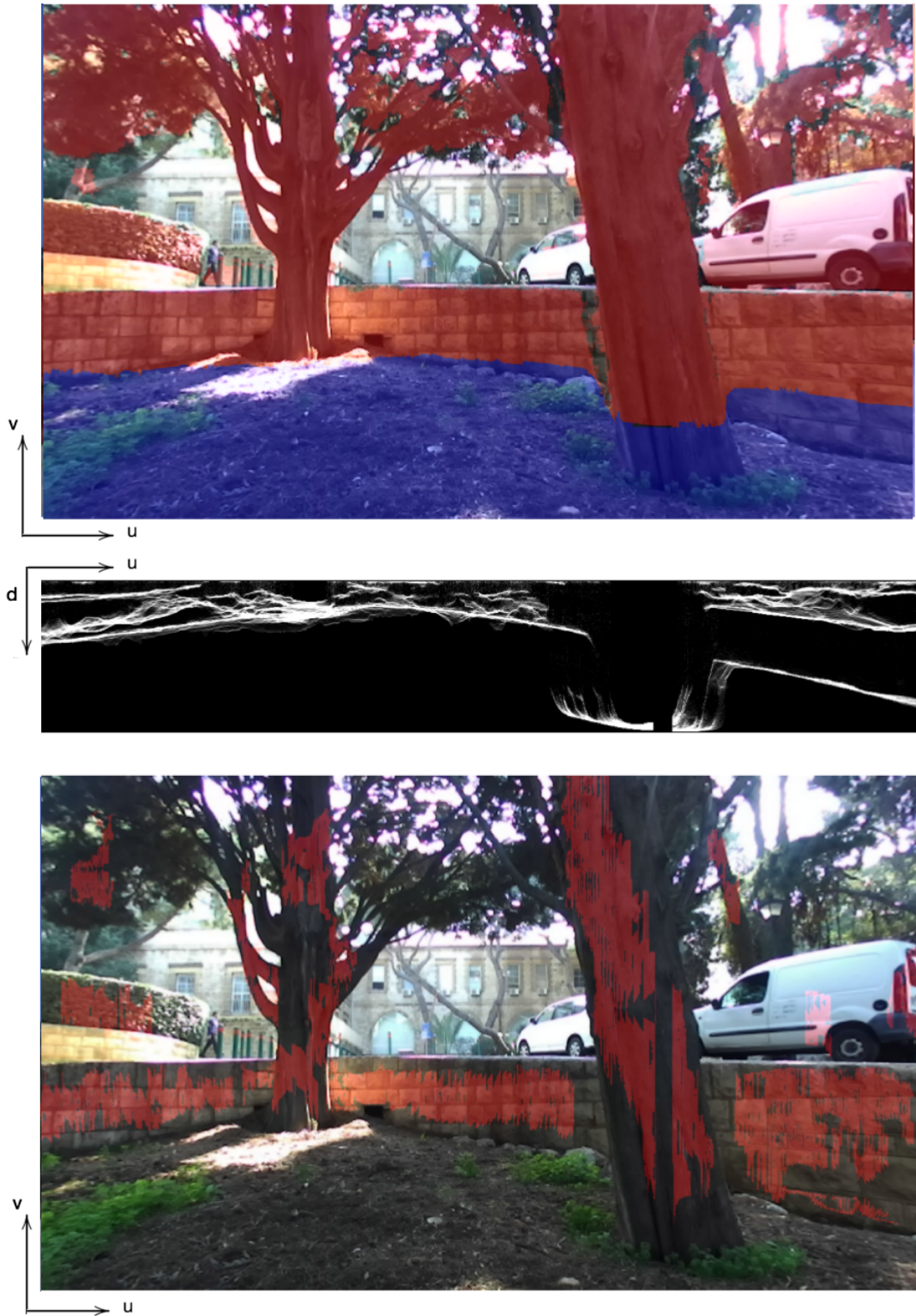


Figure 4.3: (top) original segmentation with ground pixels in blue, and the remaining pixels in red; (middle) u -disparity map corresponding to the regions in red from the top image; and (bottom) reliable training data for obstacles colored in red

4.1.3 Ground Classification and filtering

Once ground training data and obstacles are identified, the image is segmented using a sequence of two PNB classifiers. The first classifier, hereafter referred to as PNB-RGB, uses RGB data alone, and the second classifier, hereafter referred to as PNB-RGBD, uses RGB and geometry data as the input vector to the classifier. The geometry data include height, standard deviation of height in a 5-by-5 neighborhood, and height difference in a 3-by-3 neighborhood.

While PNB-RGB is applied to all pixels in an image, PNB-RGBD is only applied to pixels possessing reliable disparity values. PNB-RGB results in significant classification errors because of similarities in appearance that could exist between ground pixels and others; therefore, it warrants further processing as follows. First, edges are filtered from PNB-RGB using a Sobel edge filter, as some are erroneously classified as ground. Then PNB-RGBD is applied, and since it relies on depth in addition to appearance, its classification is more reliable. To prune pixels that were labeled as positive from the first PNB and negative from the second, I use a moving window, in which the number of positive labels according to PNB-RGBD are summed. If the sum is below a threshold, all the pixels inside that window, that were previously labeled as free space based on PNB-RGB, are now labeled as obstacles. The threshold is selected such that it takes into account patches of PNB-RGBD classifications, while ignoring floating pixels that have a relatively smaller area. This method removes positives from PNB-RGB that are spatially distant from positives in PNB-RGBD, without expensive computations. Finally, all pixels that maintain a free space label after the two PNB classifications are considered as *ground seeds* for the final IGC segmentation discussed below.

4.1.4 Interactive Graph Cuts

To segment an image using graph cuts, a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ for each pixel $p \in P$ is constructed, where P is the set of all pixels in the image. Then two terminal nodes (sink T and source S) are added to the graph, each representing a label. The terminal nodes are connected to each pixel with a set of edges noted as $\{p, T\}$ and $\{p, S\}$ respectively, and referred to as unary terms. The unary terms represent the resemblance between pixels with the same label.

In addition to the unary terms, each pixel is connected to its neighbors with another set of edges called the binary terms, which describe the differences between neighboring pixels. In interactive graph cuts, in addition to the unary and binary terms, there are seeds, which constitute non terminal vertices with pre-fixed labels, as shown in Fig. 4.4.

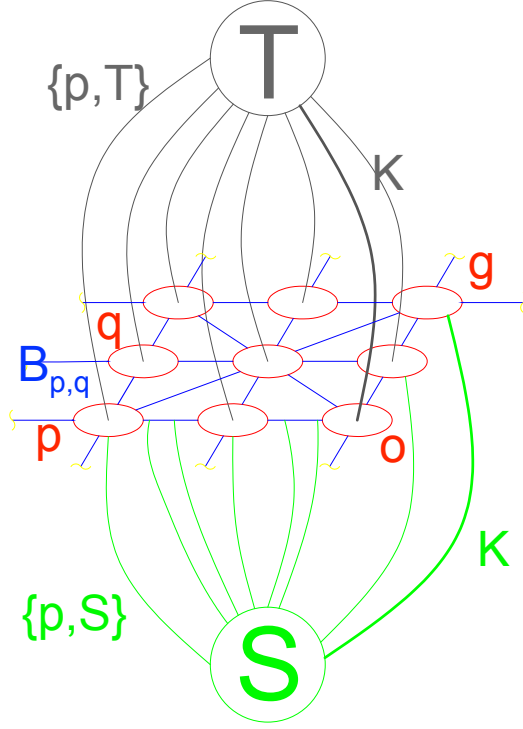


Figure 4.4: IGC for a 3x3 image. The node o represents an obstacle and g represents a ground seed. This figure is inspired by [4]

4.1.5 Unary Terms

Unary terms represent the similarity between vertices with the same label. As such, a unary term exists between each pixel, and each of the two terminal nodes, which represent free space and obstacle. In this work the unary term is calculated from the conditional probability of each label based on its RGB color, and, is calculated as the logarithm of the ratio between the two conditionals (log odds). To accommodate for pixels with near unity ratios, I add a correction factor κ to the unary term, chosen to advantage false negatives over false positives, otherwise classification for these pixels will be based only on binary terms. κ has been chosen experimentally as described later. The unary terms can be expressed as:

$$R(\text{free}) = -\log \frac{P(\text{free}|RGB)}{P(\text{obstacle}|RGB)}, \quad (4.5)$$

$$R(\text{obstacle}) = -\kappa - \log \frac{P(\text{obstacle}|RGB)}{P(\text{free}|RGB)}. \quad (4.6)$$

4.1.6 Binary Terms

Binary terms are used to coerce adjacent pixels, which have similar appearance, to be labeled equally. In general, for neighboring pixels $\{p, q\}$, the binary term is expressed as $B_{p,q} = \exp(-\Delta(p, q)\Sigma^{-1}\Delta(p, q))$, where $\Delta(p, q)$ is the difference in intensity and Σ^{-1} is the variance in intensity values.

4.1.7 Seeds

Seeds are pixels with pre-fixed labels; accordingly, the edges connecting a seed with the two terminals nodes are set to K and zero values for the similar and different terminals, respectively, where $K = 1 + \max_{p \in \mathcal{P}} \sum_{q: \{p,q\} \in \mathcal{N}} B_{p,q}$. In the system at hand, pixels identified as ground are used as seeds for the free space label, and those identified as obstacles are used as seeds for the other label. Then IGC consists of building the graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, where $\mathcal{V} = \mathcal{P} \cup \{S, T\}$ and $\mathcal{E} = \mathcal{N} \cup \{\{p, S\}, \{p, T\}\}$, and

$$\{p, q\} = B_{p,q} \text{ for } \{p, q\} \in \mathcal{N} \quad (4.7)$$

$$\{p, S\} = \begin{cases} \lambda.R(\text{free}) & \text{for } p \in \mathcal{P}, p \notin \text{seeds} \\ K & \text{for } p \in \text{free seeds} \\ 0 & \text{for } p \in \text{obstacle seeds} \end{cases} \quad (4.8)$$

$$\{p, T\} = \begin{cases} \lambda.R(\text{obstacle}) & \text{for } p \in \mathcal{P}, p \notin \text{seeds} \\ 0 & \text{for } p \in \text{free seeds} \\ K & \text{for } p \in \text{obstacle seeds} \end{cases} \quad (4.9)$$

λ is a tradeoff coefficient between binary and unary terms. Once the graph is complete, I use the iterative solution method [4] to find the final labels referred to as FSGC.

4.2 Deep Learning Techniques

Deep learning techniques are thriving in many computer vision applications, because of their ability to learn high dimensional representations [23], while being applied in real time on *GPUs*. Brust *et al* [14] proposed a CNN algorithm, referred to as CN24 for learning pixel-wise free space segmentation, while adding pixel location as a spatial prior to the fully connected layers. The pixel-wise segmentation network uses each pixel in the training images as training data points; while this approach easily provides an abundance of training data, it has a double sided drawback: if all images have the same representation, the network overfits to those data points, while if all images are different, the network will have to compromise some representations for others. Because of this training drawbacks,

and the variety of ground representations, fine-tuning is unavoidable. Sanberg *et al.* [15] proposed to employ a self-supervised fine-tuning algorithm, where the network is fine-tuned for each image with its self-supervised weak classifier. The weak classifier they used is the Stixel World that requires a couple of seconds on a *CPU* as stated by the authors, thus it is difficult to employ in a real-time application. I suggest employing FSGC on a downscaled image, thus offering real time training. In addition, instead of fine-tuning on each image, I suggest in what follows two methods that determine the need to retrain. While the first method examines the change in ground representation, the other assesses the performance of the network against a weak classifier. In what follows, training is done with the FSGC-U classification, with a downscaled image, while classification for network assesment is done with the Ground Seeds, as was explained earlier.

4.2.1 Deep Clustering

In this section, I present a novel CNN-based image clustering algorithm, hereby referred to as Deep Clustering Network (DCN). DCN is trained offline to cluster an image dataset while learning their representation, and a CN24 is then trained to segment ground in each of the learned clusters. When employed online, the system extracts a weak classifier for the given image, and tests its resemblance to the learned clusters; if it belongs to one of them, it is segmented with the corresponding CN24, otherwise the image is added as a new cluster, for which a new CN24 is fine-tuned. The system detailed below is shown in Fig 4.5.

In this system, I employ FSGC-U for training, because it has the best performance within my self-supervised classifiers. As for online comparison, I use the ground seeds presented in section 4.1.3, as they have the closest performance to FSGC with a fraction of its runtime.

In the conducted experiments, the first 10 images of each sequence are used as training data. DCN will recognize an image not belonging to a cluster if it is classified differently than its preceding images.

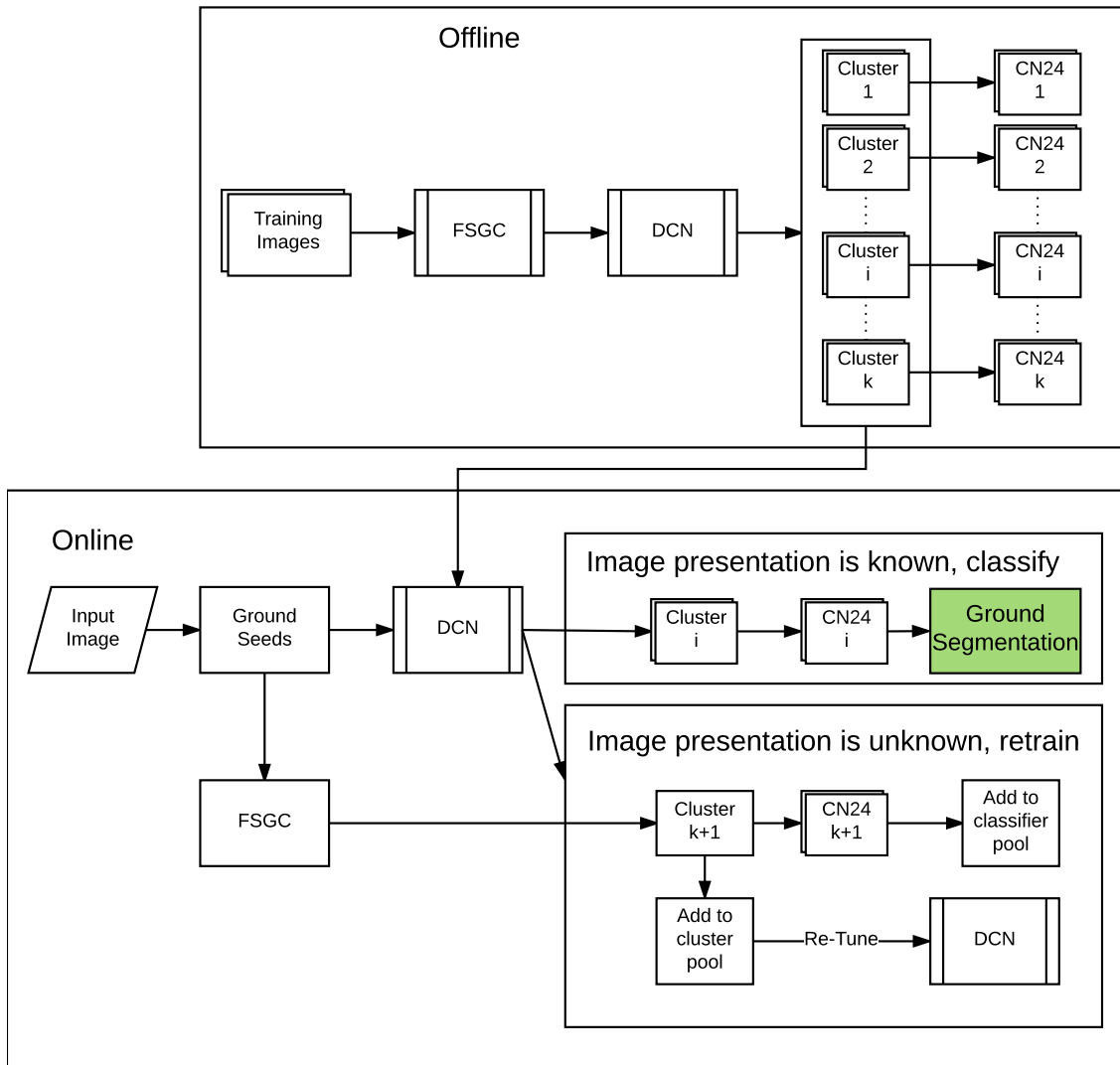


Figure 4.5: Graph showing my system employing DCN; offline DCN is used to cluster training data, while online it is used to choose the appropriate ground segmentation network.

Deep Clustering Network

A clustering network is an algorithm that can group data points in a dataset into a preset number of clusters, or a calculated one. The data points in each cluster should have features similar to those in the same cluster, and different from those in other clusters. Zagoruykou and Komodakis showed in [25] that a CNN can be designed to compare image patches. As such, my algorithm aims to match ground containing patches from the training images; as shown in Fig. 4.6, the algorithm divides each image into rectangles, and classifies only those containing ground pixels. As detailed in ALG1, DCN starts with a randomly initialized CNN, with two output classes. Until the network reaches convergence, DCN classifies the training rectangles such as all rectangles in an image have the most probable label of the image; then it creates a new class, with the images farthest from the mean probability as its training data.

The CNN architecture is a simple CNN, based on LeNet [21]. The network has been chosen to have enough Convolutional Kernels and Maxpooling layers to produce a row vector before the fully connected layers; as such, the larger the input image, the deeper the network should be. The depth of the image was chosen to produce the correct number of clusters based on a ground truth. While decreasing the depth will not allow the network to correctly differentiate the images in the dataset, the increase of depth will increase the number of clusters beyond what is required, as the network will learn higher dimensional features. As such, the chosen architecture is as follows:

Table 4.1: DCNs CNN architecture

Layer Type	Layer Size
Input Layer	10x10x3
Convolutional Layer	5x5x10
Convolutional Layer	5x5x30
Maxpooling	2x2
Fully Connected Layer	2
Softmax Layer	

In ALG1, $Prob_i$ is calculated similar to ALG2.

Algorithm 1 Deep Clustering Network

```
number_clusters = 2
Network{weights} = random()
Data = Relabeling{Network, Image, ground}
while average(Probi=1:end) < stoppingCriteria do
  Network{weights} = train{Network, Data}
  Data = Relabeling{Network, Image, ground}
  if no label changes then
    %change network last layer
    number_clusters ++
    avg = average(Probi=1:end)
    std = standardDeviation(Probi=1:end)
    label(Probi=1:end < avg - std) = last cluster
  end if
  remove empty clusters, reconfigure Network if needed
end while
```

Algorithm 2 Training Data = Relabel{Network, Images, ground}

```
TrainingData =  $\emptyset$ 
for image i  $\in$  images do
  Divide image into sub rectangles
  for rectangle j  $\in$  image do
    if sum(ground(rectanglej)) > threshold then
      labeli{j} = Network(rectanglej)
      Pi{j} = Probability(labeli{j})
    end if
    for class n  $\in$  Network classes do
      Probi{n} = sum(Pi(labeli == n))
    end for
  end for
  labeli{1 : end} = argmaxn Probi
  add {rectangle{1 : end}, labeli{1 : end}} to TrainingData
end for
return TrainingData
```

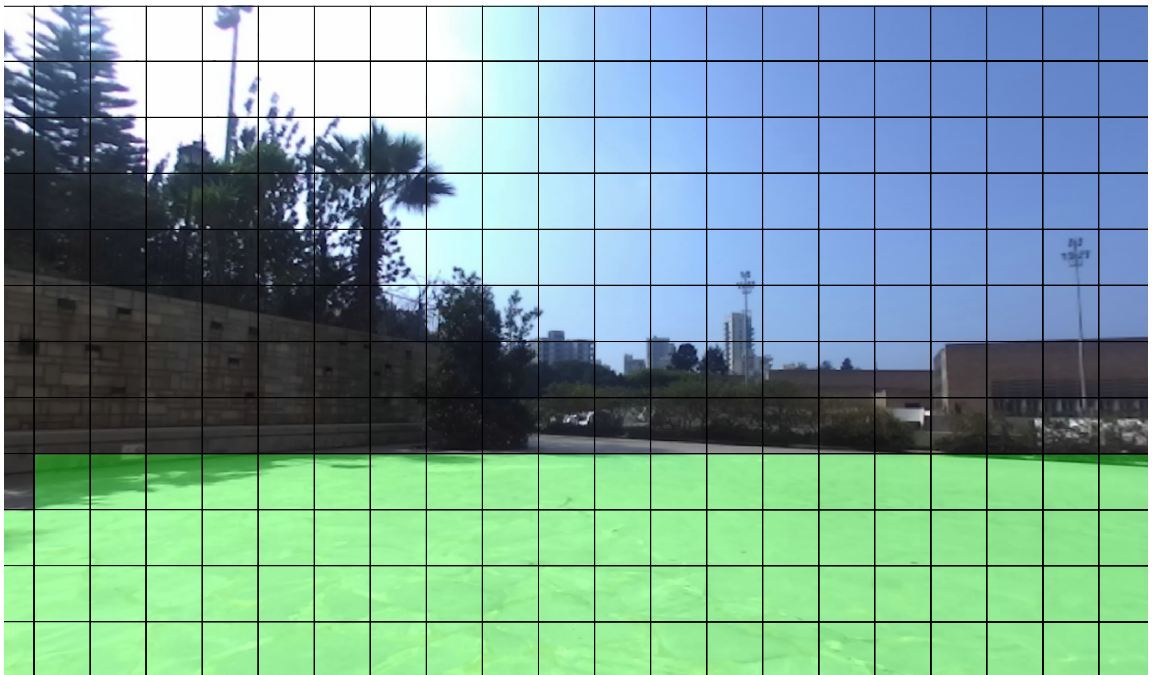


Figure 4.6: showing an example image, divided into rectangles, with the ones containing ground highlighted in green. My DCN will consider each of the highlighted rectangles as an input image.

4.2.2 Adaptive Learning

Theoretically, the ground segmentation network should be fine-tuned only when ground representation change is detected. However, this assumes that the CN24 is able to accurately segment ground similar to its training, and at the same time, the algorithm detecting representation change can learn the same features as CN24. Although this works in many situations, a better approach is to fine-tune the network when its performance drops. It is in this spirit that this system is designed; the system trains a CN24 using FSGC on the first couple of images it captures when launched, then for each new image the network's classification is assessed with a weak classifier's output. If the performance ($Precision \times Recall$) of the network decreases below a threshold, the network is fine-tuned as shown in Fig. 4.7. For this system, I also use my ground seeds, presented in section 4.1.4, as weak classifier.

This approach allows the system to run in real-time for all images, without any stop for fine-tuning. When fine-tuning is needed, FSGC is used for a couple of images, then its output on those images is used as training data. Once the system has enough training data, it fine-tunes the system and segments the next image.

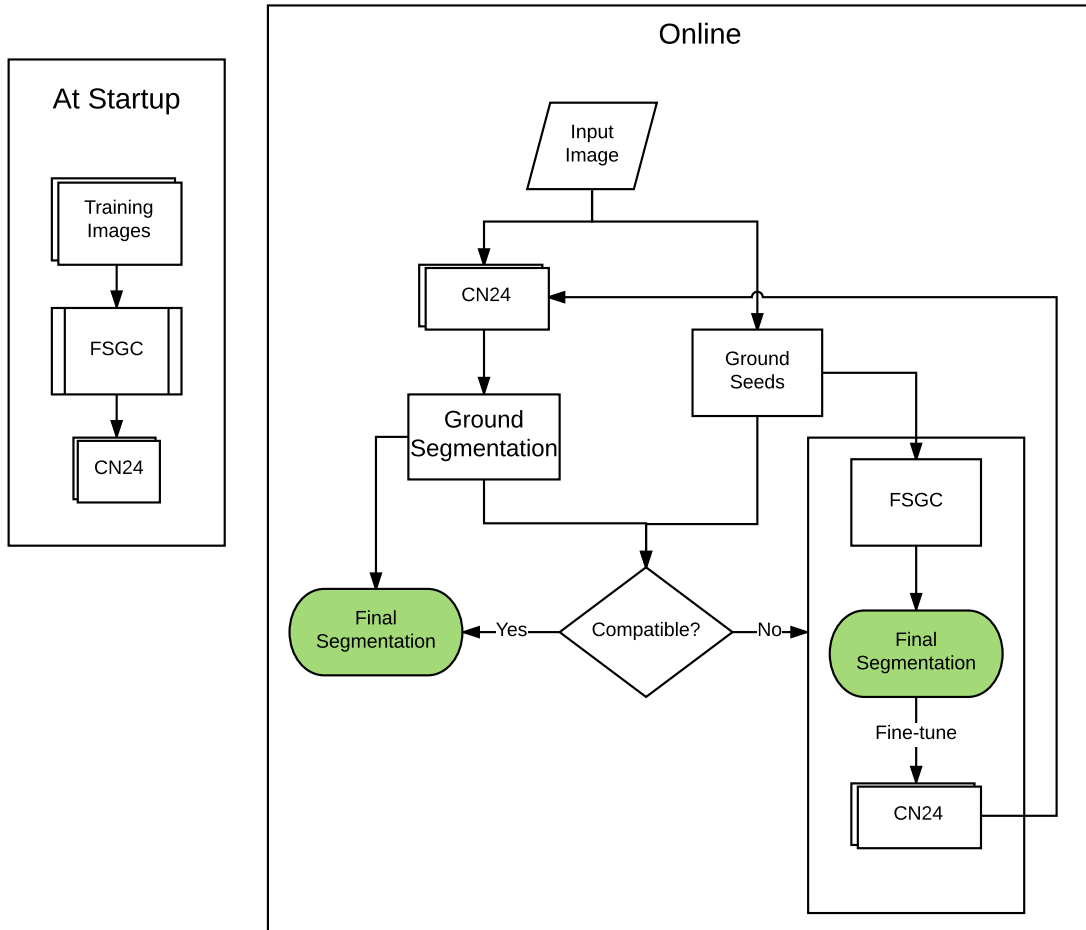


Figure 4.7: showing my adaptive learning segmentation system, where at startup the network is trained from a weak classifier, then online it is fine tuned when the weak classifier and the segmentation network are not compatible.

Chapter 5

Experiments and Results

Experiments were conducted to assess the efficiency of my proposed methods. This chapter first describes the experimental settings and the datasets that were used to assess the proposed systems. Then, results of the experiments are presented, along with a comparison with other free space detection algorithms, with emphasis on unsupervised or self-supervised systems.

5.1 Dataset

The AUB dataset [26] consists of 900 stereo images captured with a ZED stereo camera [27]; it features images of different ruggedness and flatness, ranging from easy (*i.e.*, very flat), to moderate, to difficult images (*i.e.*, off-road scenes). Each entry in the dataset consists of the left RGB image of a stereo pair, the disparity image, the corresponding XYZ point cloud, ground truth on all pixels from the left image, and ground truth on the pixels with reliable disparity. More specifically, the three datasets consist of the following:

5.1.1 Easy Dataset

The easy dataset consists of a sequence of 289 images, containing man-made ground patches, and thus labeled as easy to segment ground. This sequence was captured with the camera attached to an Unmanned Ground Vehicle (UGV).

5.1.2 Moderate Dataset

The moderate dataset consists of a sequence of 248 images, containing a combination of man made, and off road ground patches, and thus labeled as medium to segment ground. This sequence was captured with the camera attached to a UGV.

5.1.3 Difficult Dataset

The difficult dataset consists of two sequences, one of 176 and the other of 195 images, all of off-road ground patches, and thus labeled as hard to segment ground. These sequences were captured with a hand-held camera.

5.2 Benchmarking

I compare my results with two self-supervised free space detection algorithms from the literature, and the deep learning algorithm CN24 introduced earlier, trained on an offline dataset. The two self-supervised methods include:

1. ***v*-SVC** [10]: this method models the *v*-disparity data probabilistically to extract ground training data, and then using a feature vector of color and geometry, and trains a support vector classifier to identify ground pixels with reliable disparity values.
2. **Plane Fitting**: this method was used in Kim *et al.* [11] and Vernaza *et al.* [12], where planes were fit into the generated point cloud to identify ground patches. For maximum robustness towards outliers, M-estimator SAmple Consensus (MSAC) is used for plane fitting. The expected normal vector of the ground plane is required as an input.

5.3 DCN

To assess the Deep Clustering Technique, I tested its performance when provided with 10 images from each dataset, which it is required to distinguish as corresponding to different clusters.

The network is initialized with random weights, then it is provided with two distinct images, with different labels to correct its weight, before being provided with a larger set of images. This step ensures the network converges in the correct direction. The set of images are organized with the first 10 images from the Easy Dataset, the following are each from the two Difficult Datasets, and the last 10 are from the Moderate Dataset.

The first iteration of the clustering shown Figure 5.1 shows that the network so far cannot distinguish any of the clusters correctly; however, it clearly shows that while the images 11-30 have high probability to correspond to the same class, the moderate dataset images, and the falsely classified images in the easy dataset have very low probabilities. In later iterations, the network was not able to distinguish all of the easy dataset's images as corresponding to a single cluster until it added a new cluster in iteration 4 shown in Figure 5.2, which contained all of the moderate dataset's images. In addition, it is noted in Figure 5.2, that the

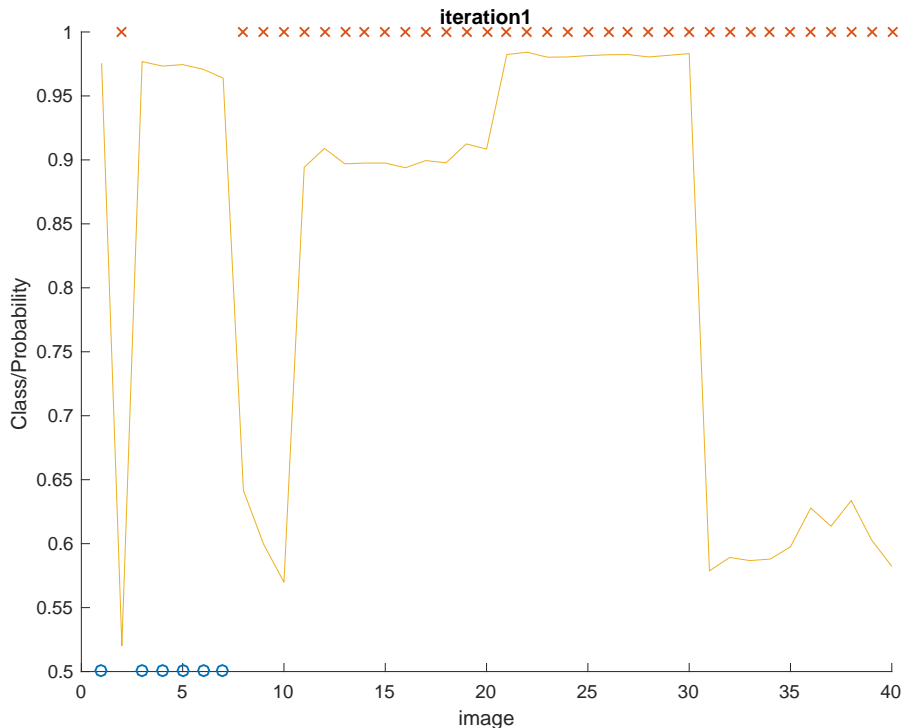


Figure 5.1: showing first iteration of the Deep Clustering Technique. Each marker shows corresponds to a class, and the plot shows average probability of the corresponding image to belong to the current cluster.

probabilities of one of the two hard datasets decreased substantially from iteration1; as the iterations progress, the network learns new ground representations, and thus becomes more aware of the possible differences between images of one class.

This awareness in the clustering algorithm allows it to finally classify images from each dataset in a corresponding cluster at iteration 7, as shown in Figure 5.3. Even though DCN does not converge on iteration 7; however, the later iterations do not change the number of clusters. When the algorithm tries to change the number of clusters, all the images shifted to the new cluster correspond to all images from one of the existing, thus shifting one of the clusters into a new one, without changing the configuration. This helps the network relearn the representation of the shifted cluster from scratch, thus grasping a better understanding of its representation.

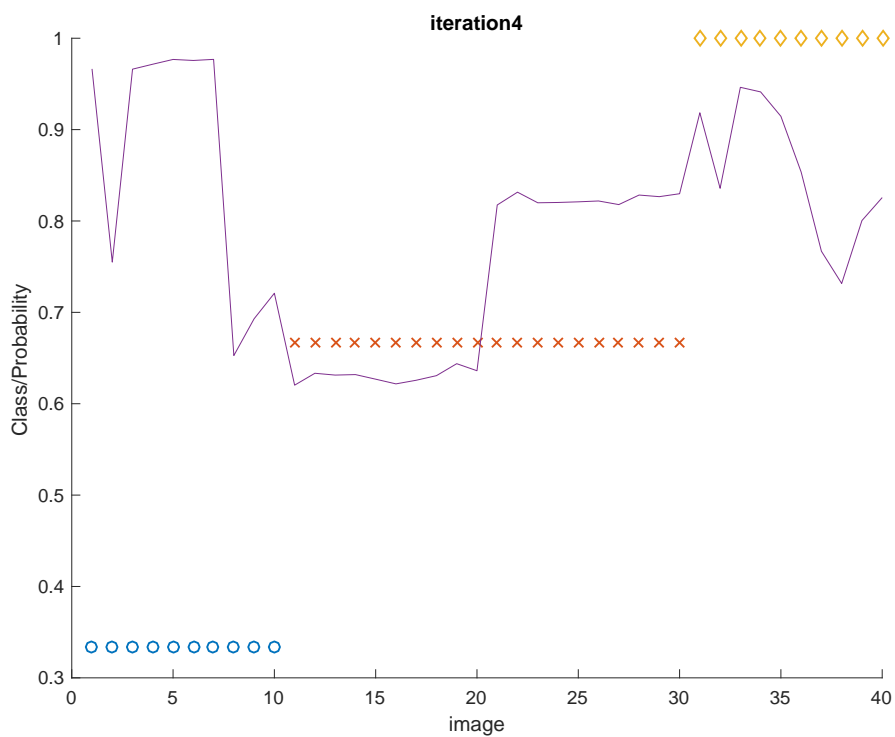


Figure 5.2: showing fourth iteration of the Deep Clustering Technique. Each marker shows corresponds to a class, and the plot shows average probability of the corresponding image to belong to the current cluster.

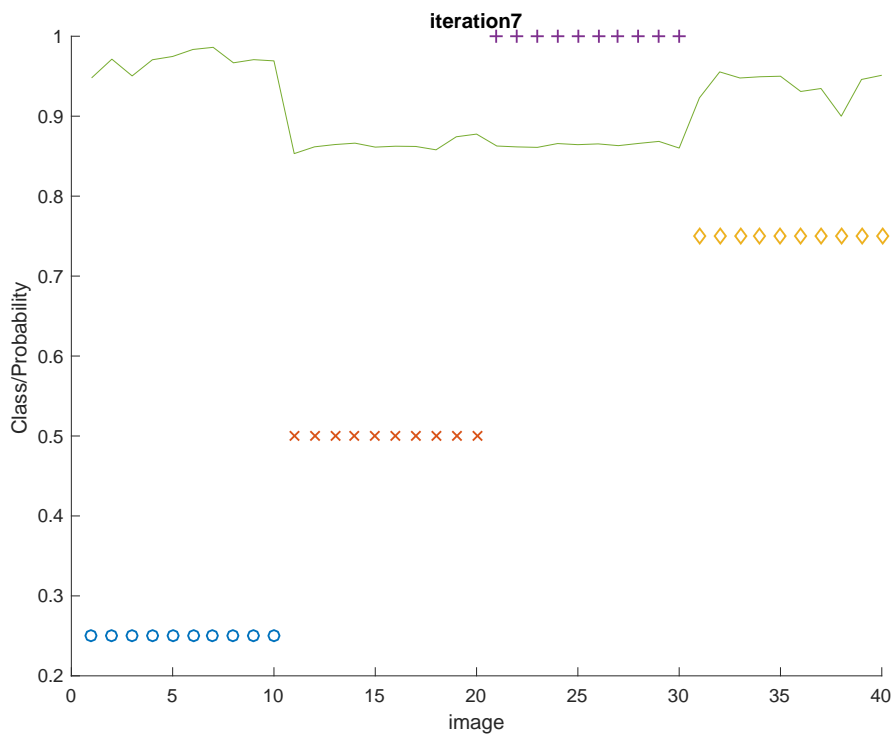


Figure 5.3: showing seventh iteration of the Deep Clustering Technique. Each marker shows corresponds to a class, and the plot shows average probability of the corresponding image to belong to the current cluster.

5.4 Analysis

In this paragraph we will first provide a comparison between FSGC-U and FSGC-O at different scales, as well as the methods used to choose FSGCs free parameters γ and κ . After that, I will compare FSGC with the above mentioned algorithms from the literature, along with the two deep learning algorithms I proposed in this work.

All of the evaluations are pixel-based. Recall, precision, and specificity are used as metrics for comparison. While recall reflects the fraction of ground pixels detected by the algorithm, precision reflects the fraction of correctly classified ground pixels. Finally, specificity provides a measure for the fraction of obstacle pixels detected by the algorithm.

5.4.1 FSGC obstacle constraint and runtime

The improved obstacle delineation entails a better understanding of the boundary between ground and obstacles in the IGC, allowing FSGC-U to outperform FSGC-O in terms of recall as shown in Table 5.1, and precision for most images as shown in Fig.5.4. It can be noted that FSGC-O outperforms FSGC-U in the easy dataset; the easy dataset presents uniform ground patches that can be classified easily with minimal obstacle restriction. While the ground seeds are the same in both algorithms, a more comprehensive obstacle delineation in FSGC-U over-constrains the IGC, resulting in a weaker performance. This problem is not faced in the harder datasets, as the ground and obstacles are closely intertwined, and ground patches can have multiple representations; thus the additional constraint imposed in FSGC-U allows it to learn a better ground segmentation. Table 5.1 shows that FSGC-U outperforms all other scales and versions of FSGC in terms of recall on most datasets; however, FSGC employed on the full scaled image requires 6.3 seconds per image, thus cannot be used for a real time system. Image rescaling has been used in ν -SVC, where the algorithm does not converge without extensive image rescaling, thus we proposed to downscale the image to reduce the runtime to two frames per second on the expense of a decrease in performance. In what follows, FSGC will refer to FSGC-U with an image rescaling of 0.3.

Table 5.1: showing a comparison between FSGC with my obstacle detection algorithm and the one proposed in [13], and comparison between both algorithms at different scales.

Easy Dataset				
Algorithm	FSGC-U		FSGC-O	
Scale	Full Scale	Down Scaled	Full Scale	Down Scaled
Recall	0.9487	0.9628	0.9573	0.9670
Precision	0.9698	0.9846	0.9614	0.9790
Specificity	0.9769	0.9832	0.9803	0.9851
Difficult Dataset: Cyprus Triangle				
Algorithm	FSGC-U		FSGC-O	
Scale	Full Scale	Down Scaled	Full Scale	Down Scaled
Recall	0.9843	0.9081	0.9226	0.9134
Precision	0.8063	0.8182	0.8449	0.8167
Specificity	0.9935	0.9633	0.9685	0.9651
Difficult Dataset: Jafet				
Algorithm	FSGC-U		FSGC-O	
Scale	Full Scale	Down Scaled	Full Scale	Down Scaled
Recall	0.9156	0.8617	0.8710	0.8675
Precision	0.9152	0.9461	0.9630	0.9452
Specificity	0.9572	0.9324	0.9364	0.9349
Moderate Dataset				
Algorithm	FSGC-U		FSGC-O	
Scale	Full Scale	Down Scaled	Full Scale	Down Scaled
Recall	0.9423	0.8801	0.8891	0.8878
Precision	0.9030	0.8689	0.8607	0.8668
Specificity	0.9745	0.9488	0.9521	0.9520

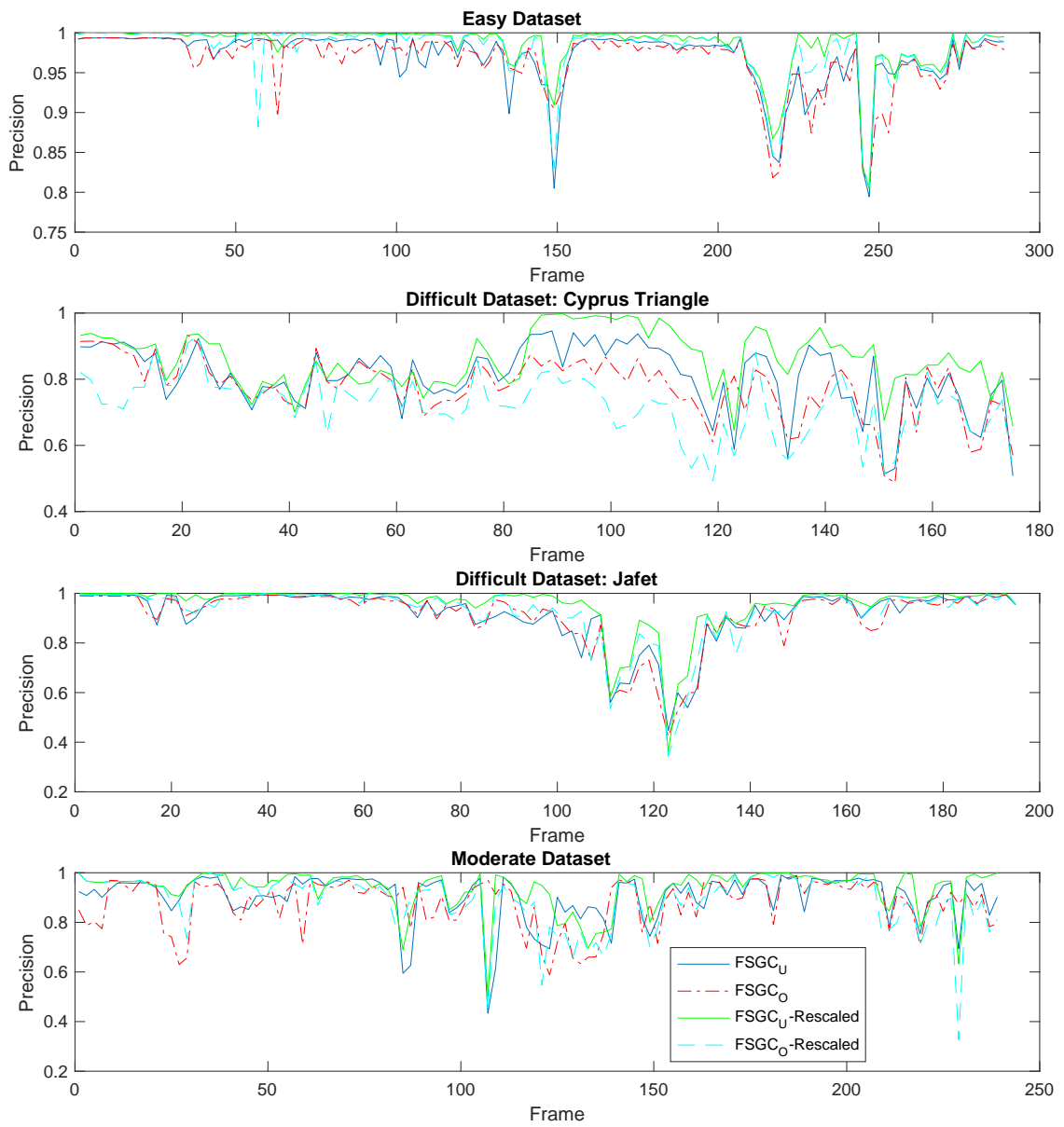


Figure 5.4: showing per image precision comparison between presented FSGC algorithm at different scaling factors

5.4.2 Free Parameters

The presented system has two free parameters: obstacle detection threshold γ in the u -disparity obstacle detection algorithm, and the correction factor κ in the unary term. While one could choose logical values for both parameters, we rely on Receiver Operating Characteristic (ROC) curve analysis to optimize my choice.

Obstacle detection threshold γ

The obstacle detection threshold γ determines the confidence interval about the largest value in each column of the u -disparity. While a near unity threshold under-segments the obstacles, and classifies only the pixels with the highest values in the u -disparity as obstacles, a near zero threshold is prone to adding noise and small objects. We rely on an ROC to select γ : we calculate the average recall and precision over all images, while varying the threshold γ between zero and unity, and keeping κ constant at unity. The ROC shown in Fig.5.5 (left), shows average precision versus recall for the different values of γ . A perfect detector should have a unity recall and precision; while that is difficult to achieve in a real system, we choose the threshold γ that has the shortest distance to the perfect detector, shown in red in Fig.5.5 (left), and corresponding to γ equal to 0.5.

Correction factor κ

The correction factor κ accommodates for near unity ratios in the unary terms log odds. κ should be chosen as a small positive number to advantage false negatives over false positives without adding many false negatives. Another ROC curve in Fig.5.5 (right) shows average recall versus precision over all images, while varying κ from 0.1 to five, and keeping γ constant at the value of 0.5, found previously. The detector with the shortest distance to the perfect detector, shown in red in Fig.5.5 (right), corresponds to κ equal to unity.

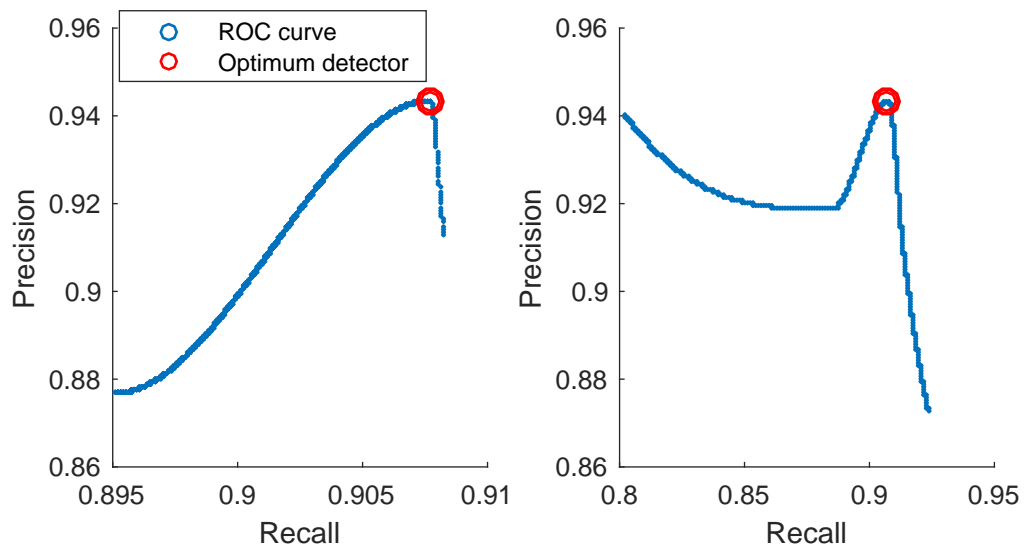


Figure 5.5: (left) ROC curve for varying γ from 0 to 1, (right) ROC curve for varying κ from 0.1 to 5

5.4.3 Runtime

The following discusses the runtime of the proposed systems as well as the ones from the literature. Some of the proposed systems have multiple runtimes as follows:

- **DCN** has two run-time: Online segmentation and Online Training. Online it requires to extract the ground seeds, choose the corresponding cluster through DCN, and apply the corresponding CN24 to find the final segmentation. In case it has to train, in addition to the extraction of the ground seeds and DCN clustering, it requires to fine-tune DCN and retrain a CN24.
- **Adaptive Learning** The Adaptive Learning technique also has two run-time: Online segmentation and Online Training. Online it requires to extract the ground seeds, and segment the ground with the CN24. In case of mismatch between the two segmentations, the algorithm employs FSGC for a couple of images, then fine-tunes the preceding CN24, with the worst case scenario being an FSGC and a CN24 segmentation on the mismatch image.

Table 5.2: evaluation of the runtime of my algorithm and the baseline algorithms.

system	FSGC	<i>v</i> -SVC	DCN Online	DCN Training
time(sec)	0.505	0.4321	0.13	55.13
system	Plane Fitting	CN24	Adaptive Online	Adaptive Training
time(sec)	0.4321	$2.5e^{-3}$	0.08	0.505

As shown in Table 5.2, the fastest algorithm is CN24, where an offline trained deep learning algorithm segments ground without any fine-tuning. It is a fast algorithm, however, it requires supervised training data, and does not adapt to unseen ground representations. FSGC, *v*-SVC, and plane fitting all achieve real time results, with near constant runtime throughout their operation; all these algorithms do not require any pretraining and exploit data directly from the currently available frame. DCN and the Adaptive Learning method exploit solutions in between, where they are pretrained on self-supervised data to be able to achieve a higher frame rate than the other self-supervised methods, while fine-tuning the network in a self-supervised manner when needed. The DCN network keeps in memory all pre-learned ground representations, thus rarely requires training, however, retraining requires extensive time. This is why the adaptive method provides the best compromise between runtime and data exploitation, as it exploits the self extracted weak classifier, while employing a self-supervised pre-trained network, achieving 12.5 frames per second online, and an acceptable training runtime.

5.4.4 Comparison

Table 5.3: evaluation of the classification of my systems and the baseline algorithms on all three datasets.

Easy Dataset						
Algorithm	FSGC	v-SVC	Plane Fitting	CN24	DCN	Adaptive Learning
Recall	0.9628	0.8017	0.8807	0.9981	0.8208	0.9015
Precision	0.9846	0.9744	0.9542	0.7411	0.9806	0.9718
Specificity	0.9769	0.9167	0.9478	0.9990	0.9259	0.9572
Difficult Dataset: Cyprus Triangle						
Algorithm	FSGC	v-SVC	Plane Fitting	CN24	DCN	Adaptive Learning
Recall	0.9081	0.8346	0.5231	0.9994	0.9130	0.9165
Precision	0.8182	0.8546	0.8811	0.6171	0.8871	0.8857
Specificity	0.9633	0.9391	0.8465	0.9997	0.9689	0.9690
Difficult Dataset: Jafet						
Algorithm	FSGC	v-SVC	Plane Fitting	CN24	DCN	Adaptive Learning
Recall	0.8617	0.7551	0.8150	0.9968	0.8485	0.8751
Precision	0.9461	0.9625	0.9680	0.7696	0.9631	0.9330
Specificity	0.9324	0.8918	0.9147	0.9981	0.9316	0.9413
Moderate Dataset						
Algorithm	FSGC	v-SVC	Plane Fitting	CN24	DCN	Adaptive Learning
Recall	0.8801	0.7859	0.8399	0.9990	0.9660	0.9187
Precision	0.8689	0.9434	0.9460	0.7228	0.9147	0.8939
Specificity	0.9488	0.9148	0.9328	0.9995	0.9843	0.9665

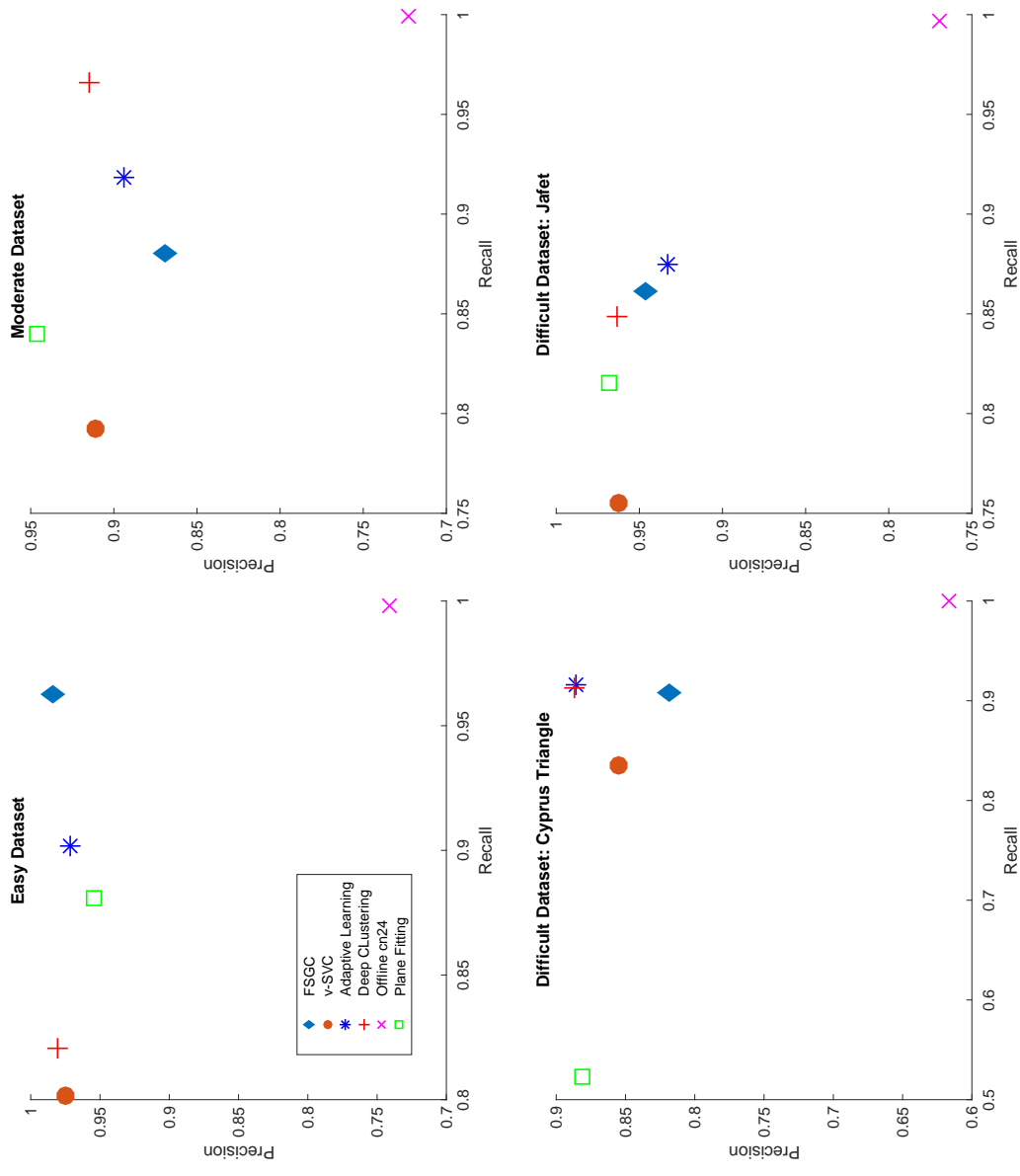


Figure 5.6: showing Precision vs Recall of all algorithms on all four datasets.

Table 5.3 shows average Recall, Precision, and Specificity of the proposed algorithms along with the most relevant from the literature, evaluated on all four datasets. The same data is shown in Figure 5.6 for an easier visual comparison, where Precision vs Recall graph for the above mentioned algorithms are shown for each of the four datasets. Figure 5.6 shows that my three proposed algorithms outperform the algorithms presented in the literature. The offline trained network shows the highest recall for all datasets, however, on the expense of a large decrease in precision. *v*-SVC shows the opposite performance, where it outperforms most other algorithms in terms of precision, however, on the expense of a lower recall. The Plane Fitting performance is comparable to my algorithms for most datasets, however, its recall drops below all others on the hard dataset, where ground cannot be segmented with a geometric surface, rather intensity information can be more informative.

Despite my algorithms outperforming all others, none of the three presented systems can outperform the other on all datasets. As seen in Figure 5.6, while in the easy dataset, FSGC is superior to all others, the other two have equal or superior performance on the different datasets.

Fig. 5.8 shows example segmentations from the different datasets using FSGC. Most of the results show accurate segmentation for ground and obstacles. Failure modes were also analyzed for FSGC, where it was observed to be sensitive to patterns of repeating texture such as foliage, where disparity values are not accurate; this can be observed in Fig. 5.8c, where ground and foliage from obstacles are almost connecting, and possessing similar color, rendering the image almost impossible for FSGC to correctly segment.

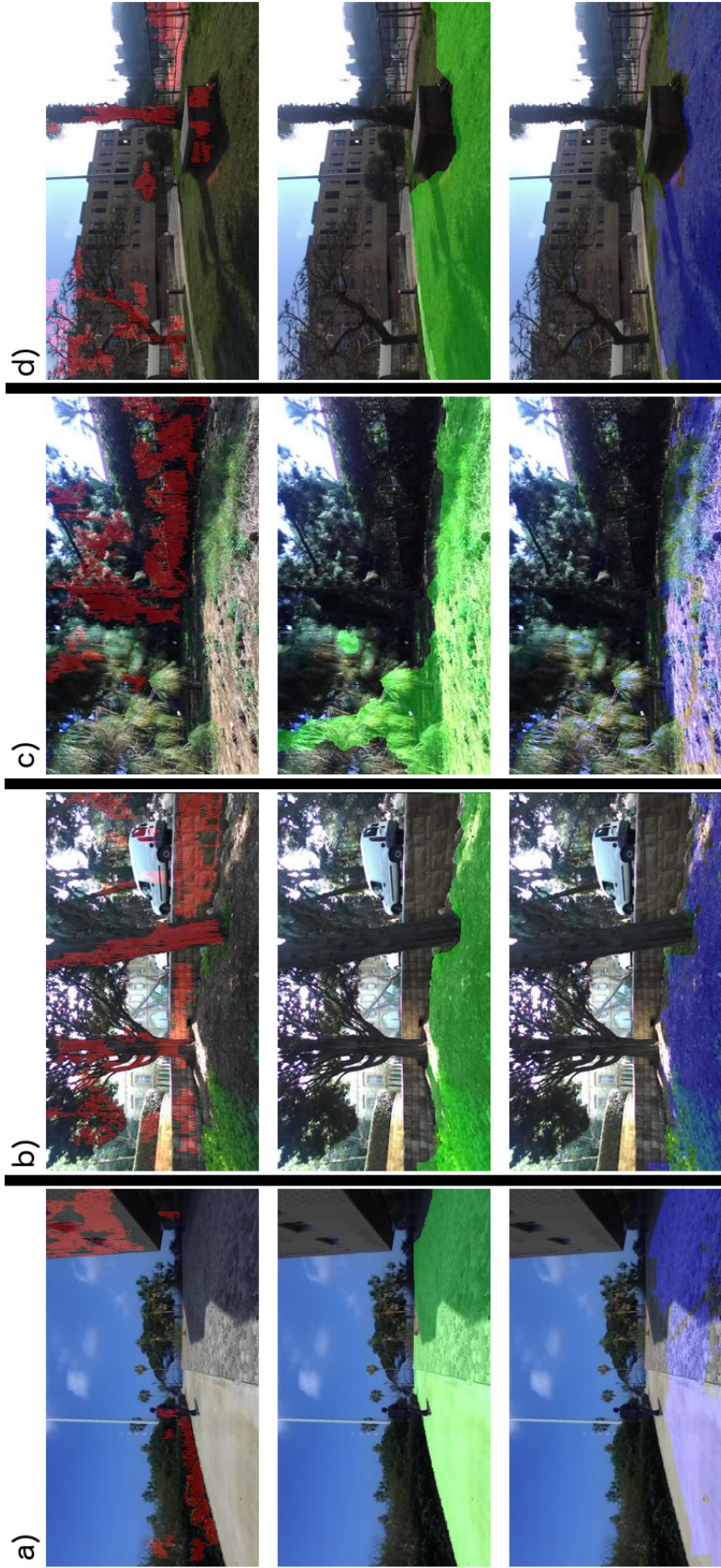


Figure 5.7: each column shows the results of the segmentation applied to an image taken from one of the three datasets. The top row shows, in red, the pixels labeled as obstacles; the bottom row shows the ground seeds overlaid in blue; and the middle row shows the final classification overlaid in green. Note how most segmentations are very good, clearly delineating the ground plane. The segmentation in column (c) shows a case where FSGC mistakenly segments branches of a tree as ground; this is due to the physical proximity, and similarity in color, between the branches and the ground plane.

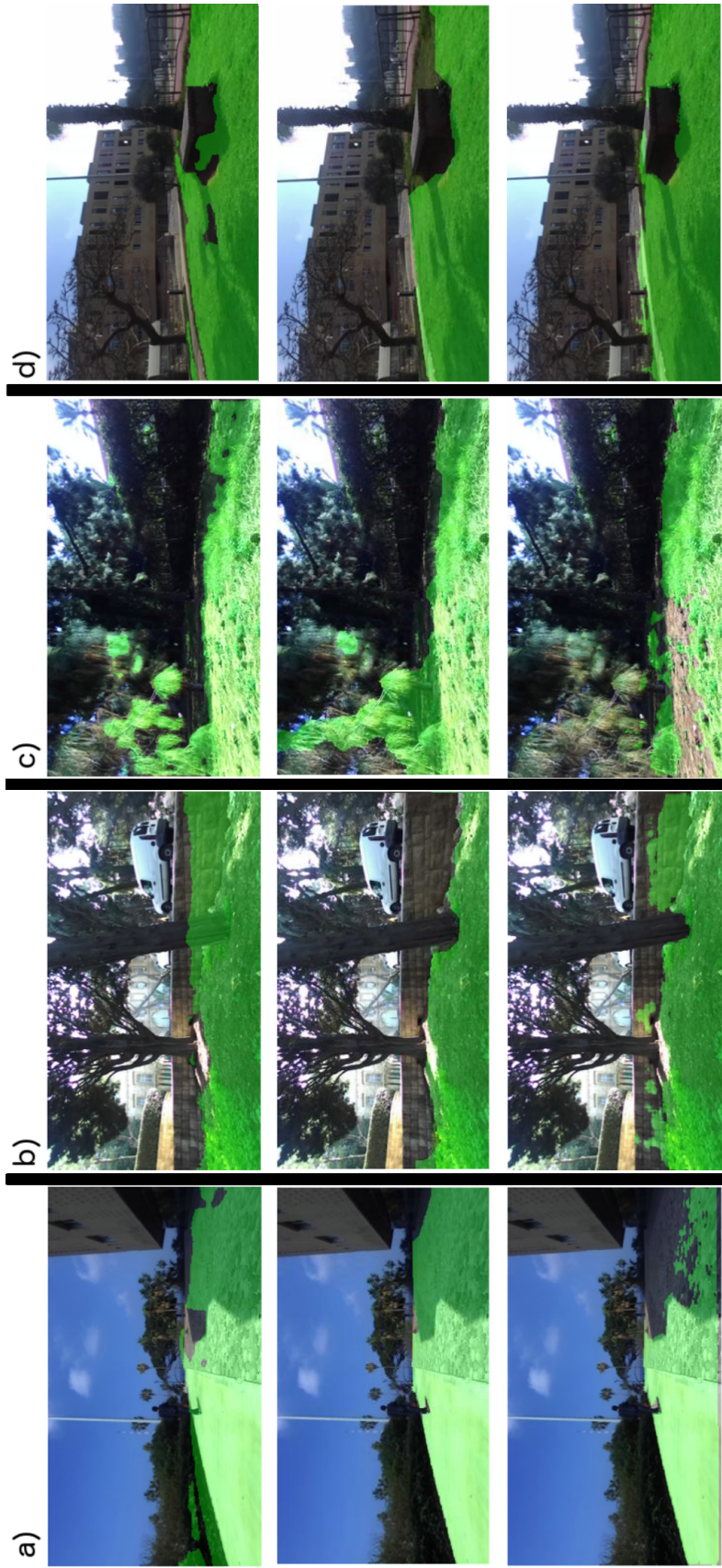


Figure 5.8: each column shows the results of the segmentation applied to an image taken from one of the three datasets. The top row shows segmentations from the Adaptive Learning algorithm, the middle row shows segmentation from FSGC, and the bottom row shows segmentation from the DCN algorithm.

Chapter 6

Conclusion and Future Work

The systems presented above are novel techniques for free space estimation. FSGC is completely self supervised, and performs well on structured and unstructured terrain; the Adaptive Learning and DCN segmentations provide a compromise between deep learning networks that have been trained in only unseen ground, and thus no guarantee that they will reflect the current scene, and between expensive network fine-tuned for each frame. The presented methods outperform all others from the literature in real time.

As a conclusion, the main contributions in this thesis are the following:

- A self supervised free space segmentation algorithm that employs a mixture of machine learning techniques to outperform all other self supervised algorithms in the literature.
- A Deep Clustering Technique that employs a CNN to cluster images based on their ground representation.
- A Deep Segmentation network fine-tuning on demand technique employing DCN.
- An adaptive learning technique that compares a weak classifier and a pre-trained one, to fine-tune the pretrained network when required.

Despite the contributions in this thesis, much work remains to be done; DCN and the Adaptive technique each assesses only one aspect of the segmentation, and I expect a combination of the two methods to outperform them both. In addition, implementing DCN inside the CN24 can allow the DCN to learn the representation that CN24 has been trained on, thus improving both algorithms. Another appealing solution is to add a temporal prior to the spatial prior of the CN24, thus constraining consecutive images in a sequence to have similar segmentations, thus avoiding large drops in recall or precision that can be seen in

Figure 5.4. A last necessary step is to implement FSGC on a *GPU* thus reducing further the runtime of all algorithms.

Bibliography

- [1] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440, 2015.
- [2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *arXiv preprint arXiv:1506.02640*, 2015.
- [3] R. Girshick, “Fast r-cnn,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1440–1448, 2015.
- [4] Y. Boykov and G. Funka-Lea, “Graph cuts and efficient nd image segmentation,” *International journal of computer vision*, vol. 70, no. 2, pp. 109–131, 2006.
- [5] B. Suger, B. Steder, and W. Burgard, “Traversability analysis for mobile robots in outdoor environments: A semi-supervised learning approach based on 3d-lidar data,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3941–3946, IEEE, 2015.
- [6] H. Dahlkamp, A. Kaehler, D. Stavens, S. Thrun, and G. R. Bradski, “Self-supervised monocular road detection in desert terrain,” in *Robotics: science and systems*, vol. 38, Philadelphia, 2006.
- [7] A. Milella, G. Reina, J. Underwood, and B. Douillard, “Visual ground segmentation by radar supervision,” *Robotics and Autonomous Systems*, vol. 62, no. 5, pp. 696–706, 2014.
- [8] R. Labayrade, D. Aubert, and J.-P. Tarel, “Real time obstacle detection in stereovision on non flat road geometry through” v-disparity” representation,” in *Intelligent Vehicle Symposium, 2002. IEEE*, vol. 2, pp. 646–651, IEEE, 2002.
- [9] A. Harakeh, D. Asmar, and E. Shamma, “Ground segmentation and occupancy grid generation using probability fields,” in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 695–702, IEEE, 2015.

- [10] A. Harakeh, D. Asmar, and E. Shamma, “Identifying good training data for self-supervised free space estimation,” *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [11] D. Kim, S. M. Oh, and J. M. Rehg, “Traversability classification for ugv navigation: A comparison of patch and superpixel representations,” in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3166–3173, IEEE, 2007.
- [12] P. Vernaza, B. Taskar, and D. D. Lee, “Online, self-supervised terrain classification via discriminatively trained submodular markov random fields,” in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 2750–2757, IEEE, 2008.
- [13] F. Oniga, E. Sarkozi, and S. Nedevschi, “Fast obstacle detection using u-disparity maps with stereo vision,” in *Intelligent Computer Communication and Processing (ICCP), 2015 IEEE International Conference on*, pp. 203–207, IEEE, 2015.
- [14] C.-A. Brust, S. Sickert, M. Simon, E. Rodner, and J. Denzler, “Convolutional patch networks with spatial prior for road detection and urban scene understanding,” *arXiv preprint arXiv:1502.06344*, 2015.
- [15] W. P. Sanberg, G. Dubbelman, and P. H. de With, “Free-space detection with self-supervised and online trained fully convolutional networks,” *arXiv preprint arXiv:1604.02316*, 2016.
- [16] H. Badino, U. Franke, and D. Pfeiffer, “The stixel world—a compact medium level representation of the 3d-world,” in *Joint Pattern Recognition Symposium*, pp. 51–60, Springer, 2009.
- [17] F. Denis, A. Laurent, R. Gilleron, and M. Tommasi, “Text classification and co-training from positive and unlabeled examples,” in *Proceedings of the ICML 2003 workshop: the continuum from labeled to unlabeled data*, pp. 80–87, 2003.
- [18] J. He, Y. Zhang, X. Li, and Y. Wang, “Naive bayes classifier for positive unlabeled learning with uncertainty,” in *SDM*, pp. 361–372, SIAM, 2010.
- [19] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.
- [20] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, “Handwritten digit recognition with a back-propagation network, 1989,” in *Neural Information Processing Systems (NIPS)*.

- [21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [22] D. A. Forsyth and J. Ponce, “A modern approach,” *Computer vision: a modern approach*, pp. 88–101, 2003.
- [23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [24] D. C. Asmar, J. S. Zelek, and S. M. Abdallah, “Tree trunks as landmarks for outdoor vision slam,” in *2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW’06)*, pp. 196–196, IEEE, 2006.
- [25] H. Badino, U. Franke, and D. Pfeiffer, “Learning to compare image patches via convolutional neural networks,” in *Joint Pattern Recognition Symposium*, pp. 51–60, Springer, 2009.
- [26] “Aub ground segmentation stereo dataset.”
- [27] “S.labs.”