Babeș Bolyai University

Cluj-Napoca

Faculty of Mathematics and Computer Science

Pole detection using machine learning

LICENSE THESIS

Supervisor

Asist. Dr. Mircea Ioan-Gabriel

Student

Nașca Sergiu Alin

Cluj-Napoca, Romania

Year III

Universitatea Babeș Bolyai

Cluj-Napoca

Facultatea de Matematică și Informatică

Detecție de stâlpi folosind machine learning

LUCRARE DE LICENȚĂ

Coordonator științific

Asist. Dr. Mircea Ioan-Gabriel

Student

Nașca Sergiu Alin

Cluj-Napoca, România

Anul III

Table of Contents

Abstract

The thesis license purpose is to extend a machine learning toolbox, the initial toolbox was design for pedestrian detection, the final state and purpose of the toolbox is to detect poles using the grayscale image and disparity(depth) image, another purpose of this toolbox is to make prediction in real time, using this prediction to measure the distance between the car and the poles and increase the accuracy of the GPS car by triangulating GPS positions.

The first chapter contains a short history about AI in automotive driving. Also in this chapter is presented the structure of this license thesis and the problem statement and motivation.

Chapter two introduce some basic concept about machine learning, the architecture design, different types of algorithms or different techniques used to extract features.

Chapter three contains the extension phase of the existing toolbox, I have fully explained how I generated data and how are they pre-processed. This chapter also contains an explanation of boosting algorithm(AdaBoost) and a comparison between two models.

Chapter four contains details of the results obtained and their accuracy.

Chapter five shows the tools needed to run the toolbox and the system minimal requirements. In this chapter are presented portions of code used to for training and for predictions.

Chapter six has an own conclusion about my work and an auto evaluation of my work and also the new things that I added to the domain of research. This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

Chapter 1

Introduction

Today's world is meeting with an increased number of vehicles on the road. As the population grows the technology advances very fast. Cars become more complex, they have more safety features to protect passengers against injures in a crash accident, and they also have a performance system to avoid accidents.

1.1 Short history about automotive driving

In 2004 was held the first competition of Darpa (Defense Advanced Projects Agency) called also "DARPA Grand Challenge" during which many of the teams had to realise a soft. That soft had to simulate human driving, to analyse the situation and to make the best decision, and also to act according to those; The distance that had to be driven by a self-driven car was 240 km through Mojave Desert, from United States of America. The teams were based on laser, GPS, lidar and proximity sensor and other systems. The best of the teams has managed to travel 11.78 km remaining stuck after making a wrong turn[12].

In 2005 Darpa organised one more time the challenge, this time the record of 11.78 km was overtaken by almost all teams, 5 of them managed to complete those 240 km without any accident. Winning time was set at 6 hours and 54 minutes[12]. The winning car was not based only on laser systems, GPS and lidar systems but also on artificial intelligence with the images being made by a color camera[10].

This contest attract the attention of many companies,Google was among them. They join in the competition to develop self-driving algorithms.

A huge problem was that the laser systems and lane systems were way too voluminous in order to be mounted on street cars. We can say that in accordance with

the development of self-driving algorithms, the car manufacturers try to minimised the computer board size needed to perform this type of algorithms.

Over the years, several algorithms have been developed based on artificial intelligence, one of them being pedestrian detection, maintaining bookmarks based on traffic, detecting cars around the car, etc.

Nowadays the autonomous driving section is in full development, there are already some models of cars that offer this benefit. We can say that in accordance with the development of self-driving algorithms, the competition is minimised for the necessary systems to create a self-driving machine. Some of these may be replaced by self-driving cars ex: Tesla Model S, Mercedes-Benz S Class and many others, but this benefit is limited because it is currently not 100% safe, having enough accidents, produced by the automatic pilot.

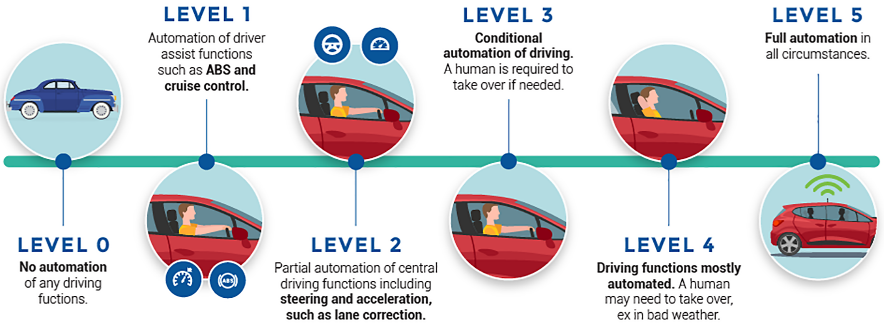


Figure 1.1: Level of Autonomous Driving Technology [1]

In Figure 1.1, the levels of automation are highlighted, the current level being 3.

Level 0: The driver performs all the operating tasks like braking, accelerating, slowing down or steering.

Level 1: At this level, the vehicle can assist with some functions(cruise control, auto brake system), but this functions are limited and the driver still need to handles the acceleration or braking, also must pay attention at the surrounding environment and monitoring it.

Level 2: Level 2 have more improvements, the vehicle can assist with steering and acceleration functions and allow the driver to disengage from some of their tasks. The driver must always be ready to take control of the vehicle, it is responsible for most safety-critical functions.

Level 3: The biggest improvement at this level is that the vehicle itself controls

[1] Figure from `https://www.caa.ca/wp-content/uploads/2017/08/17-CAA_Av-Infographic_ENG.png`

all monitoring environment(using sensors like LiDAR, or proximity sensors). At this level the driver attention is more passive, but it must be focused in case of emergency.

Level 4: At level 4 and 5, the vehicle is capable of steering, braking, accelerating, and monitoring the surrounding environment. The vehicle will respond to events, and it will make it's own decision.

Level 5: Last level of autonomous driving requires absolutely no human attention. There is no need for pedals, or steering wheel, as the autonomous vehicle system controls all critical tasks.

1.2 Thesis structure

The following provides a short description on each subsequent chapter in this license thesis:

- **Chapter 2** provides a description of related state of the art implementations and makes a comparison between machine learning and deep learning.
- **Chapter 3** provides an in-depth review over the architecture of the implementation system, and also is highlighting the incremental growth through which it was obtained.
- **Chapter 4** shows a quantitative analysis on the performance of the designed system.
- **Chapter 5** describes what technologies have been used to design the system and how to use them.
- **Chapter 6** provides an overview of the entire project, how the goals were achieved and also future improvements possibilities.
- The **Bibliography** section at the end of the document which highlights references spread throughout the document.

1.3 Problem statement and motivation

This thesis license is aimed at designing, implementing, testing and evaluating an original solution for pole detection using machine learning. Since technology has

evolved greatly in recent years, it is now possible for production cars to have a performing computer board that can support an Artificial Intelligent algorithm for object detection, and also combine data from sensors like radars or LiDAR and perform complex computation. The detection of the poles is not used only for object detection and object avoidance, it is used to localize the surrounding objects(static objects) and create a high definition map; Why we need high definition map? Because we want that the next algorithm based on artificial intelligence and depth map to 'know' exactly where the line is ending, to change the line also when the computer is facing an open road to 'know' it can speed up, or to overtake a car. Figure 1.2.

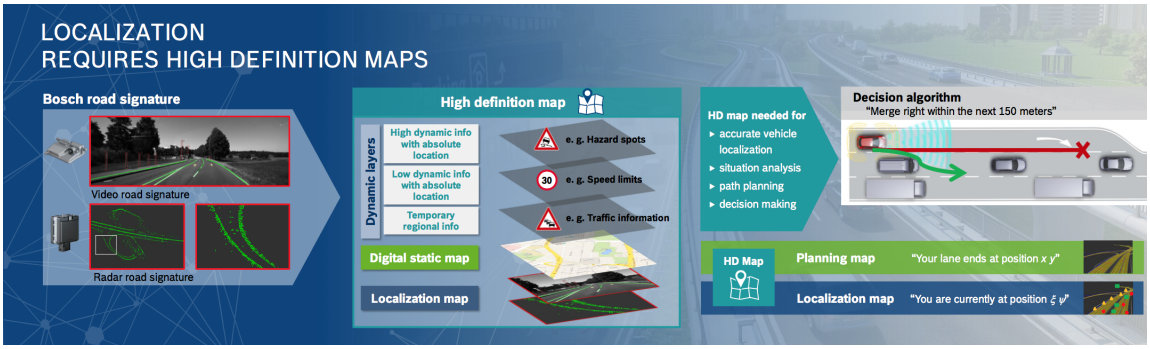


Figure 1.2: Importance of the HD map, and localization

How does the position localization works? Figure 1.3 illustrates this; Suppose that we have detected two poles, this poles are marked on map and they have a GPS position, the car also has a GPS position but this position is not very accurate due to the car's velocity and the refreshing rate of the GPS. Using the depth map, which is very precise in depth measurements, we can calculate the distance between the car and the poles, and by triangulation of this three GPS coordinates, we can measure the offset of the car position. Using the accurate position of the car we can mark down road signs, street lights or any static object.

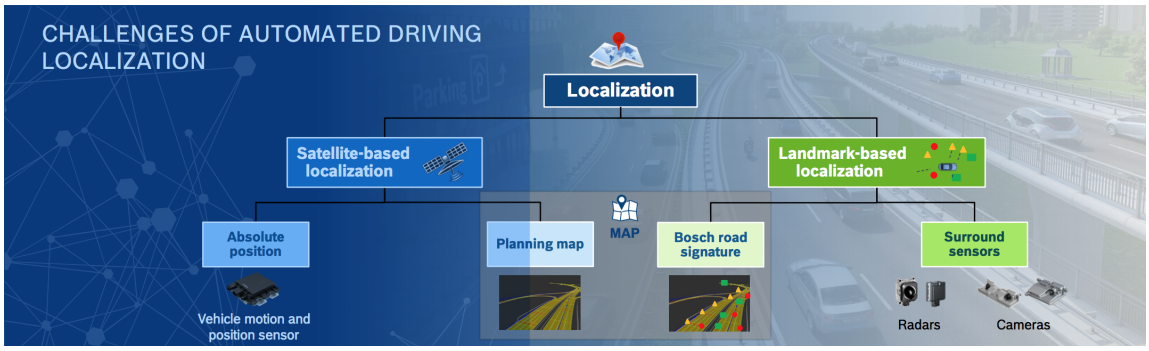


Figure 1.3: Localization tree, using GPS, motion sensor, radars, cameras and HD map

Chapter 2

Bibliographic Research and Theoretical Foundations

Any new scientific research project, or in this case, a licenses thesis, should begin first by analysing the work of others in the same domain. These similar works provide an understanding of the current state of the art techniques and results already existent. Only after these have been revised, can new innovations or improvements be done accordingly.

2.1 Difference between Machine Learning and Deep Learning

Machine learning is a part of artificial intelligence, in fact we can say that machine learning is a way to implement it. Like Tom Mitchell said "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."[9]. This could confuse us but to understand it better I have found two examples given by Faizan Shaikh and those are: predicting weights based on heights and a storm prediction system. The first example wants to predict/define the expected weight based on the height of a person and in order to do that we must collect data. In the Figure 2.1, we can see a graph with many points and each of that point represents one data point.

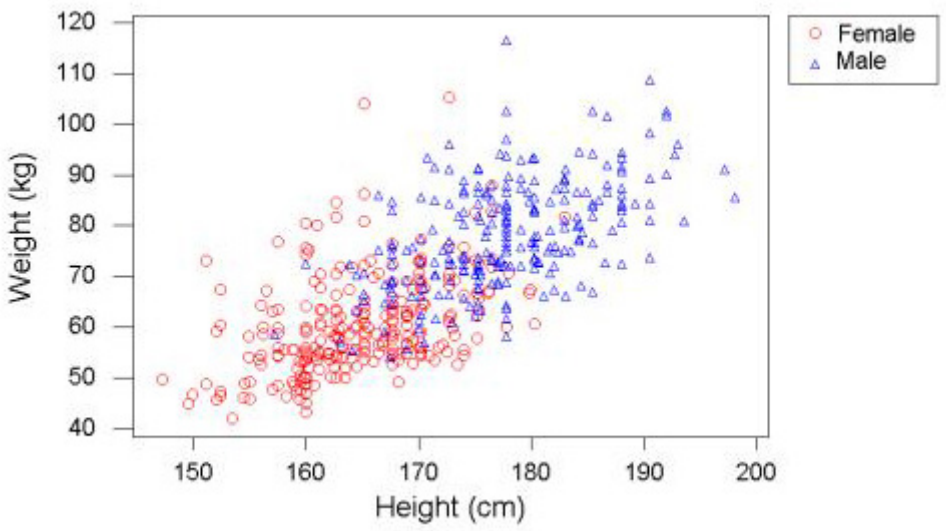


Figure 2.1: Distribution of person weight and height based on gender [1]

"To start with we can draw a simple line to predict weight based on height. For example a simple line: $Weight(inkg) = Height(incm)\check{}100$, can help us make predictions. While the line does a decent job, we need to understand its performance. In this case, we can say that we want to reduce the difference between the predictions and actuals. That is our way to measure performance. Further, more data points we collect (Experience), the better will our model become. We can also improve our model by adding more variables (e.g. Gender) and creating different prediction lines for them."[11]

The other example, which refers to storm prediction system can be defined like this "Our task 'T' here is to find what are the atmospheric conditions that would set off a storm. Performance 'P' would be, of all the conditions provided to the system, how many times will it correctly predict a storm. And experience 'E' would be the reiterations of our system."

Letting aside a bit machine learning and talking about deep learning we can see a difference even if 'deep learning' is a term of machine learning. Through deep learning we can implement machine learning by using artificial neural networks algorithm.

First of all we can say that "Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones."[11]

Also for this term, we can exemplify through an example, shape detection. In order

[1] Figure from `https://www.analyticsvidhya.com/blog/2017/04/comparison-between-deep-learning-machine-learning/`

to differentiate a square by another shape the main thing that we check is whether there are four lines or not, if yes, we further check for connections through them, if they are closed, equal or perpendicular.

Theoretically, we took a complex task and broke it in many others simple tasks and this is the way deep learning works in each case.

The differences between these two types of learning are quite numerous. We can compare the two by data dependencies, feature engineering, problem solving approach, execution time and interpretability.

- We can see the difference given by its performance of data volume; Deep learning algorithms are quite slow when we have small amounts of data, which we cannot say about machine learning that needs a few data in order to understand;
- Deep learning algorithms needs high-end machines (due to the "large amount of matrix multiplication operations"[11]) meanwhile machine learning algorithms can work perfectly on low-end machines;
- If we talk about feature engineering, "In Machine learning, most of the applied features need to be identified by an expert and then hand-coded as per the domain and data type." and "Deep learning algorithms try to learn high-level features from data."[11]. The feature engineering part is a major step ahead form machine learning because deep learning wants to reduce the tasks of developing new feature extractor for every problem.
- Considering the problem solving approach deep learning has a technique to solve the problem end to end meanwhile, machine learning use to break the problem in two parts and recombine them at the end. To have a better understanding follow the next image:

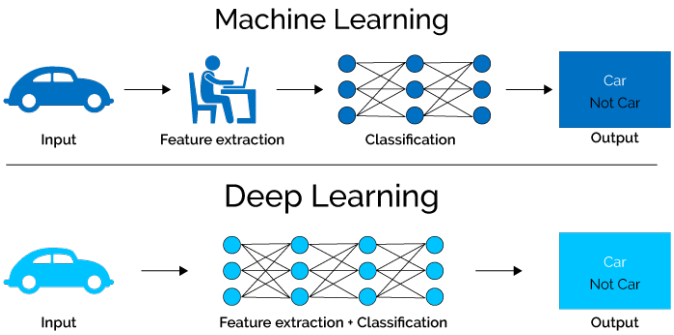


Figure 2.2: Difference between Machine Learning and Deep Learning [2]

[2]Figure from `https://www.slideshare.net/pranavathiyani/machine-learning-in-biology`

- For deep learning we have a long time to train, there are cases that takes two weeks to train completely a learning algorithm but it's not the case of machine learning where we have from a few seconds to a few hours, time to train.

- It's quite difficult to interpret the results regarding deep learning, and this because we don't know why the score of performance reach a certain level. On the other hand there are machines learning with an easy way to interpret the reason behind its algorithms. To sum up everything, machine learning uses algorithms to parse data, learn from that data and make informed decisions based on what it has learned. Deep learning structures algorithms in layers to create an "artificial neural network" that can learn and make intelligent decisions on its own. Therefore, deep learning is a subcategory of machine learning while both of them are subcategories of artificial intelligence.

2.2 Machine Learning Algorithms

There is a common principle describing, in a mathematical way, machine learning algorithms.

"Machine learning algorithms are described as learning a target function f that maps input variables X to an output variable $Y : Y = f(X)$".[8]

2.2.1 Architecture

The full process of machine learning, its architecture, is built by different infrastructure functions as follows: data acquisition (this is where data is collected, prepared and forwarded for processing), data processing (where datasets samples are selected and trained in order to execute the machine learning routines), data modeling or model engineering (data models design and machine algorithms used in data processing), execution (consists in experiments, testing and tuning) and deployment (where the usable results of the machine learning are deployed to enterprise systems).

2.2.2 Boosting

Boosting is a machine learning algorithm (meta-algorithm) that perform supervised learning, increments improvement of learned function and forces the weak learner to generate new hypotheses that make less mistakes on harder parts. Kearns and Valiant asked once if "a set of weak learners can create a single strong learner"[1]. The most

part of boosting algorithms consist of iteratively learning which means that boosting is not constrained by the algorithm. Practically, the boosting algorithm consists in adding weak classifiers to a stronger classifier meanwhile they are weighted in a certain way, in order to re-weight them after adding it to the stronger classifier. In the end, those who are misclassified gain weight and the others one are classified correctly. The main difference between boosting algorithms, is given by their method "of weighting training data points and hypotheses". There are a few examples of boosting algorithms and some of them are the following: AdaBoost, LPBoost, TotalBoost, BrownBoost, etc.

2.2.3 AdaBoost

AdaBoost comes from Adaptive Boosting, and is known as being the first practical boosting algorithm. To explain better this algorithm, I prefer to insert a diagram, in which we assume that has 10 input observations that we will classify as "+" (stands for all the inputs and predict in blue region) or "-" (stands for inputs in the reddish region).

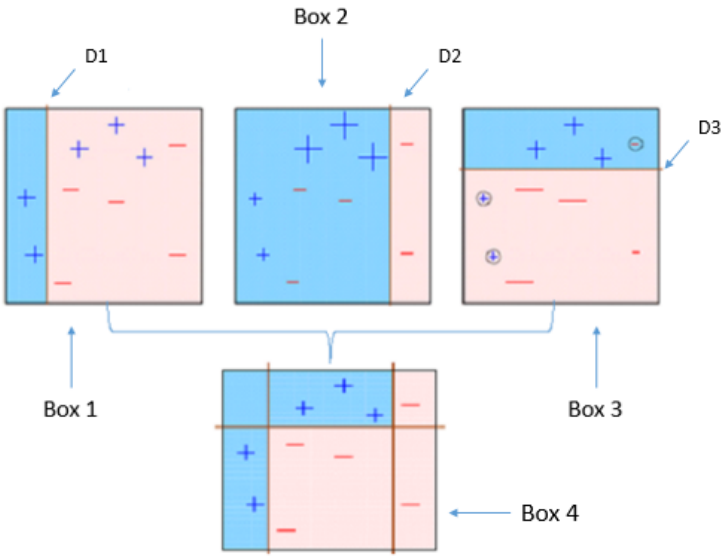


Figure 2.3: Explanation of AdaBoost algorithm [3]

In the first box, the boosting algorithm assigns equal weights to all inputs from the blue and reddish region, using decision stump D1. The next iteration, box 2, let us see weights of wrongly classified plus signs that are greater than other inputs

[3]Figure from `https://www.analyticsvidhya.com/wp-content/uploads/2015/11/boosting10.png`

and therefore we will have a second decision (D2) in order to classify correctly these observations. The final box, box 3, has three misclassified negatives from the previous run and to correct that it will take a third decision (D3). Finally, we can see that the output strong learner (box 3), has a strong rule made by the combination of the weak decisions stumps.

2.2.4 Random Forest

Random Forest is another machine learning algorithm that, almost in all cases, produces a great result without hyper-parameter tuning. Because of its simplicity is the most used algorithm. It is called like this because it's actually like a real forest, this algorithm creates a random forest using decision trees merged all together in order to have an accurate and stable prediction. This algorithm is used also for classification that for regression problems. Through the qualities of this algorithm, we find also the easiness to measure the relative importance of each prediction's feature. Below we can see an image with a "forest" compound from two trees:

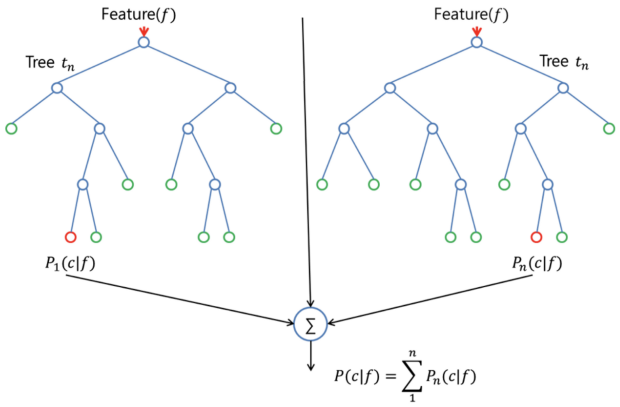


Figure 2.4: Random forest example [4]

[4]Figure from `https://cdn-images-1.medium.com/max/1600/0*tG-IWcxL1jg7RkT0.png`

2.3 Integral Channel Features

Given an input image I, a corresponding channel is a registered map of the original image, where the output pixels are computed from corresponding patches of input pixels (thus preserving overall image layout). A trivial channel is simply $C = I$ for a gray-scale image, likewise for a color image each color channel can serve as a channel. Other channels can be computed using linear or non-linear transformation of I, various choices are discussed below. We use Ω to denote a channel generation function and write $C = (I)$. To allow fast detection using sliding window, detector channels must be translationally invariant, that is given I and I' related by a translation, $C = \Omega(I)$ and $C' = \Omega(I')$ must be related by the same translation. This allows Ω to be evaluated once on the entire image rather than separately for each overlapping detection window. [5]

- **Gray**: The simplest channel is simply a gray-scale version of the image.[5]

- **Nonlinear Transformations**: There are countless translationally invariant non-linear image transformations, here we present a few illustrative cases. Gradient magnitude captures unoriented edge strength while Canny edges, more explicitly, compute edge information. Gradients for both can be computed at different scales by smoothing the input image; moreover, for color images a common trick is to compute the gradient on the 3 color channels separately and use the maximum response.[5, 3]

- **Gradient Histogram**: The essential thought behind the histogram of oriented gradients descriptor is that local objects appearance and shape, within an image, can be described by the distribution of intensity gradients or edge directions. The image is divided into small connected regions called cells, and for the pixels within each cell, a histogram of gradient directions is compiled. The descriptor is the concatenation of these histograms. For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a block, and then using this value to normalize all cells within the block. This normalization results in better invariance to changes in illumination and shadowing [7]. The first step of calculation in many feature detectors, in image pre-processing, is to ensure normalized color and gamma values. As Navneet Dalal and Bill Triggs[3] point out, however, this step can be omitted in HOG descriptor computation, as the ensuing descriptor

normalization essentially achieves the same result. Image pre-processing thus, provides little impact on performance instead, the first step of calculation is the computation of the gradient values. The most common method is to apply the 1-D centered point discrete derivative mask in one or both of the horizontal and vertical directions. Specifically, this method requires filtering the color or intensity data of the image with the following filter kernels:

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix} and \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

Dalal and Triggs author of the article "Histograms of Oriented Gradients for Human Detection" tested other complex masks, such as the 3x3 Sobel[6] mask or diagonal masks, but these masks generally performed more poorly in detecting humans in images. They also experimented with Gaussian smoothing before applying the derivative mask, but similarly found that omission of any smoothing performed better in practice.

- **HOG (Histogram of oriented Gradients)**: The second step of calculation is creating the cell histograms. Each pixel within the cell casts a weighted vote for an orientation-based histogram channel based on the values found in the gradient computation. The cells themselves can either be rectangular or radial in shape, and the histogram channels are evenly spread over 0 to 180 degrees or 0 to 360 degrees, depending on whether the gradient is “unsigned” or “signed”. Navneet Dalal and Bill Triggs[3] found that unsigned gradients used in conjunction with 9 histogram channels performed best in their human detection experiments. As for the vote weight, pixel contribution can either be the gradient magnitude itself, or some function of the magnitude. In tests, the gradient magnitude itself generally produces the best results. Other options for the vote weight could include the square root or square of the gradient magnitude, or some clipped version of the magnitude.[3]

Chapter 3

Detailed Design and Implementation

This chapter will describe the implementation and the architecture of the pole detection algorithm. It will focus not only on showing the complete, final solution, but also the intermediate and incremental steps that leads to it.

3.1 Dataset organisation and manipulation

3.1.1 Scale and structure

The data for training and testing the solution is collected by the author of the thesis. The dataset consists of grayscale images and disparity images that were taken using stereo camera mounted on a car(for grayscale images are used the left images), and includes samples used for training the model and evaluating its performance. More details about these disparity images and how they are obtained are presented in Section 3.1.2. The size of the dataset is roughly 250 "images", where each image is composed of a grayscale image and its image of disparity. Each "image" try to capture different situation of poles. Obviously poles obstructed by other objects or vegetation are hard to detect. Another example hard to detect are poles which doesn't have disparity(the grayscale image does not provide enough detail for matching the pixels from left image with pixels from right image).

The reference output of the network also called ground truth has to be present in the dataset, due to a fact that, the implementation presented in this license thesis has supervised learning. The ground truth of the input images consist of bounding boxes which has next properties $xMin, yMin, width\ and\ height$, the $xMin\ and\ yMin$ represents the top left corner of the bounding box, and *widht and height* represents the size of the bounding box.

In 250 images were labeled 933 poles, resulting 933 positive samples. The negative samples are extracted from the same images but excluding positive samples, so from 250 images resulted in 6048 negative samples.

The performance of the detector is checked by computing the difference between the actual output of the detector, and the ground truth(what the detector says about a bounding box, versus what it's in this bound box).

3.1.2 Disparity image in detail

A disparity image is computed in the following way. Two identical cameras (called a stereo camera pair) are placed at the same height inside the car, with a small horizontal shift between them. They capture grayscale images at the same time, and each image (left and respectively right) capture the same view of the world, but slightly shifted. Each object that is close to the two cameras have a high horizontal shift between the two images, that is, each pixel that compose the object is ideally located at the same height y in the image, but at the different horizontal coordinate x. Similarly, objects that are far away will have only a small horizontal shift. This difference between the horizontal displacement of each pixel in the left and right images $\Delta x(\Delta x = x_2 - x_1$ where x_2 is the position of a pixel in the left image, and x_1 is the position of the corresponding pixel in the right image. The corresponding image refers to the one that represents the same real world point) is called **disparity**. Thus, an algorithm for computing the disparity will be reduced to taking the two images(left and right), finding the corresponding pixels, and deducing the disparity for almost all pixels from the image using the horizontal offset of it.(See Figure 3.1)

The red point represents a real world point (top of the mountain) that will be captured in different horizontal offsets(x_1 *and* x_2) by the two cameras. The search for the disparity involves finding the corresponding matching for each point in the left image, into the right image. The difference offset for this example can be seen by the dotted green and blue lines that represents the point's location in the left and right images respectively.

Thus, a disparity image will contain, at each pixel, instead of the grayscale value of the pixel, the value of the disparity of the pixel. It will, in essence, represent the inverse depth of each point of the image, because disparity is directly correlated with depth(points closer to white will have a high disparity, i.e. are closer to the car, while points closer to black will have a low disparity, i.e. are far away). This is very useful because it will give a three dimensional view of the entire image(the x and y

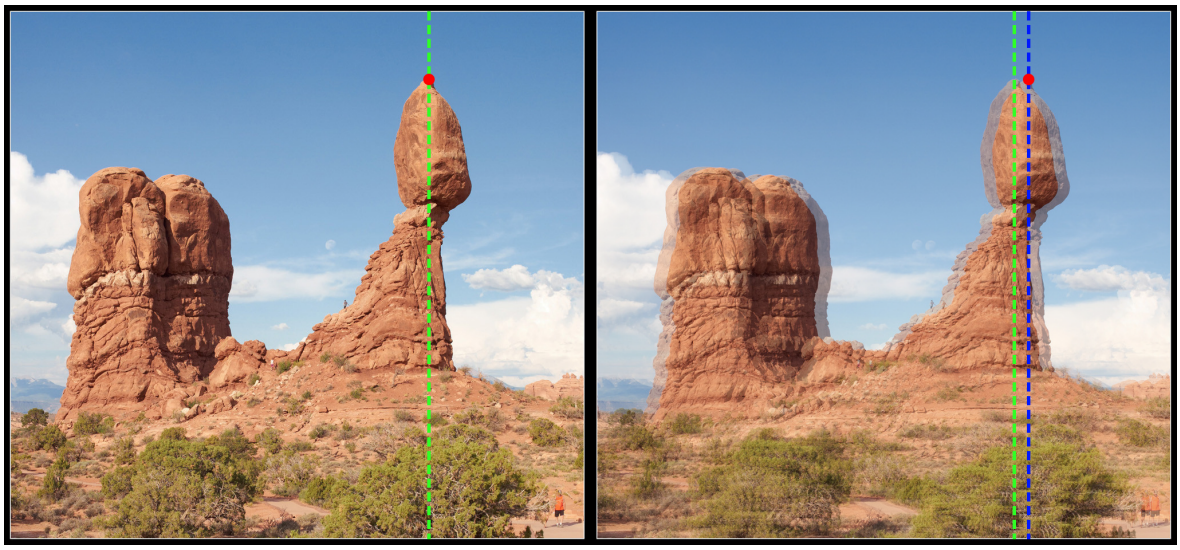


Figure 3.1: Stereo image, in the left side there is the left image, and in the right side there is the right image overlaid by left image [1]

coordinates of the pixel, plus the value of the pixel which corresponds to z, will give a good estimation of the real-world position of the point that the pixel represents). From the disparity image is easy to find where the poles are, but there is also a disadvantage, the disparity map is not perfect and it is very noisy.

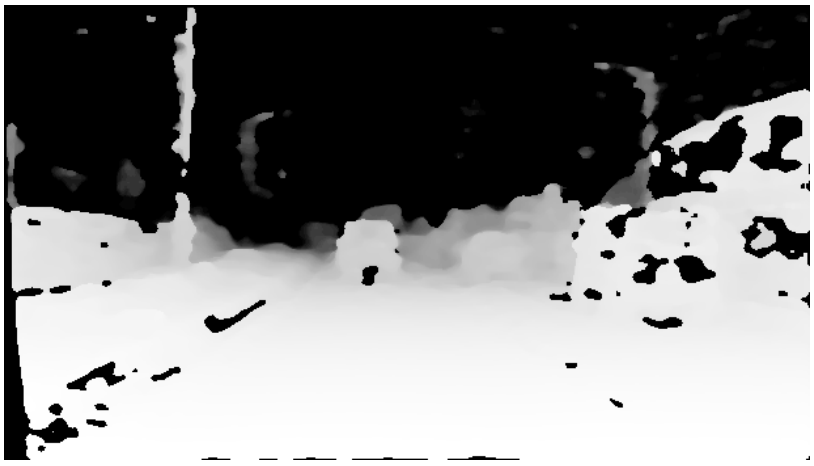


Figure 3.2: Disparity image smoothed with a median filter

[1] Figure from `http://i.imgur.com/Cc2SvXL.jpg`

Figure 3.3: Gray scale image accordingly to the disparity image from top.

3.2 Implementation steps and tests

3.2.1 Initial toolbox model and architecture

The initial model of the toolbox was designed and tested for pedestrian detection. The input channel could be an image, usually, for pedestrian detection is used an RGB image or a grayscale image. Before the extraction of features, was applied a smoothing filter(triangular filter) for removing the noise. Several filters like HOG, gradient magnitude or LUV were used to extract the features, then they were aggregated and vectorized and applied boosted tree. After the detections are computed they are sent to post-processing part, where are filtered based on their score.

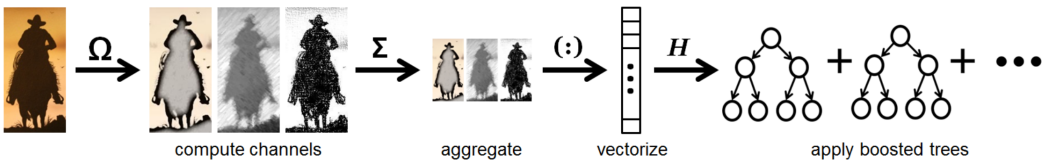


Figure 3.4: Toolbox initial architecture[4].[2]

The final architecture and implementation is detailed in Section 3.3.

3.2.2 AdaBoost(Adaptive Boosting)

AdaBoost can be used to boost the performance of any machine learning algorithm. It is best used with weak learners. These are models that achieve accuracy just above

[2]Figure from `https://pdollar.github.io/files/papers/DollarPAMI14pyramids.pdf`

random chance on a classification problem. The most suited and therefore most common algorithm used with AdaBoost are decision trees with one level. Because these trees are so short and only contain one decision for classification, they are often called decision stumps. Each instance in the training dataset is weighted. The initial weight is set to:

$$weight(x_i) = \frac{1}{N}$$

where x_i is the i'th training instance and n is the number of training instances.[2]

The decision tree consists of few strong-learners which are based on weak-learners. In our application I define 4 strong-learners which consists on the following weak-learners [64 256 1024 4096]. I test different values for decision tree but seems that one is the best, and I also test different values for the depth of the trees and the best one is 5 or 4.

A weak classifier (decision stump) is prepared on the training data using the weighted samples. Only binary (two-class) classification problems are supported, so each decision stump makes one decision on one input variable and outputs a + 1.0 or -1.0 value for the first or second class value. The misclassification rate is calculated for the trained model. Traditionally, this is calculated as:

$$error = \frac{correct - N}{N}$$

Where error is the mis-classification rate, correct are the number of training instance predicted correctly by the model and N is the total number of training instances. For example, if the model predicted 78 of 100 training instances correctly, the error or mis-classification rate would be (78-100)/100 or 0.22. This is modified to use the weighting of the training instances:

$$error = \frac{sum(w(i) * terror(i))}{sum(w)}$$

Which is the weighted sum of the mis-classification rate, where w is the weight for training instance i and $terror$ is the prediction error for training instance i which is 1 if mis-classified and 0 if correctly classified. For example, if we had 3 training instances with the weights 0.01, 0.5 and 0.2. The predicted values were -1, -1 and -1, and the actual output variables in the instances were -1, 1 and -1, then the terrors would be 0, 1, and 0. The mis-classification rate would be calculated as: $error = (0.01 * 0 + 0.5 * 1 + 0.2 * 0)/(0.01 + 0.5 + 0.2)$ or $error = 0.704$ A stage value is calculated for the trained model which provides a weighting for any predictions that

the model makes. The stage value for a trained model is calculated as follows:

$$stage = \ln\left(\frac{1 - error}{error}\right)$$

Where stage is the stage value used to weight predictions from the model, $ln()$ is the natural logarithm and error is the mis-classification error for the model. The effect of the stage weight is that more accurate models have more weight or contribution to the final prediction. The training weights are updated giving more weight to incorrectly predicted instances, and less weight to correctly predicted instances. For example, the weight of one training instance w is updated using:

$$w = w * exp(stage * terror)$$

Where w is the weight for a specific training instance, exp() is the numerical constant e or Euler's number raised to a power, stage is the mis-classification rate for the weak classifier and terror is the error the weak classifier made predicting the output variable for the training instance, evaluated as:

$$terror = 0 \quad if(y == p) \quad otherwise1$$

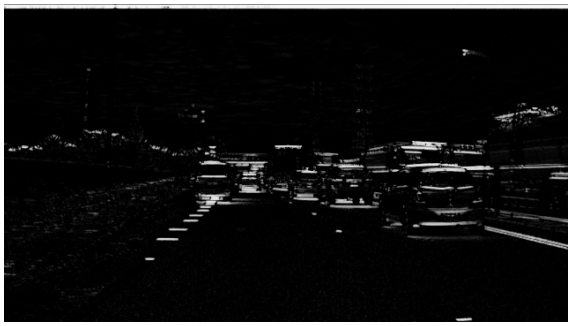
Where y is the output variable for the training instance and p is the prediction from the weak learner. This has the effect of not changing the weight if the training instance was classified correctly and making the weight slightly larger if the weak learner mis-classified the instance.[2] A problem that poles does have is that they are not a complex shape where you have a lot of information, corners, round shape or other types, they are like straight lines, and they are very common to other types of object, like corner of the building, or like the bars on the trucks.(See Figure 4.5.) Why I choose to use AdaBoost Algorithm? Because I want to run the model very fast, approximately in real time, and another reason is because car manufactures don't want to put expensive hardware in cars. This algorithm does not require so much resources and I think this is the best and easiest algorithm to use.

3.2.3 Grayscale image input channel

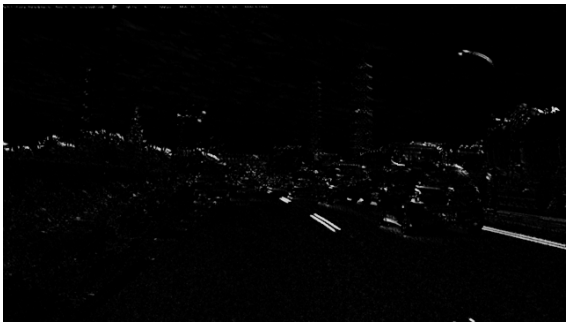
One input channel is the grayscale image, after this image is smoothed, there are applied a few filters to extract features. There are applied 7 filters, the gradient magnitude and histogram of oriented gradients. There are 6 orientations(0°, 30°, 60°, 90°, 120° and 150°). These are 7 filters and there is also the grayscale filter resulting 8 features channels.

(a) Grayscale image

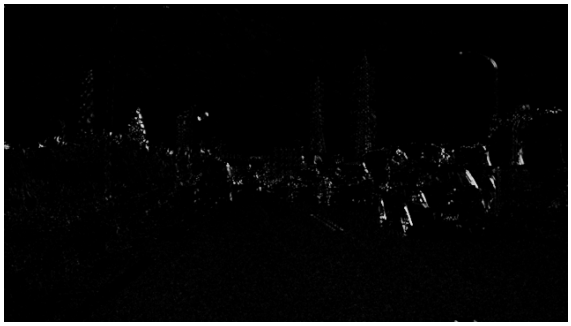
(b) Gradient Magnitude



(c) Hog at angle 0°



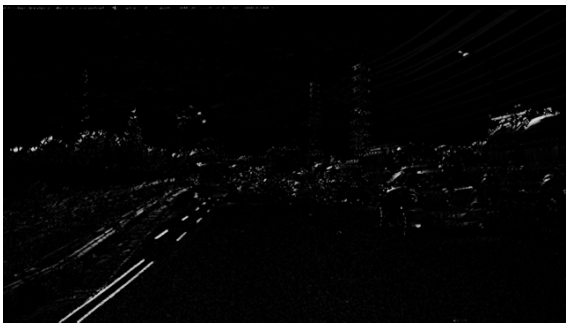
(d) Hog at angle 30°



(e) Hog at angle 60°

(f) Hog at angle 90°

(g) Hog at angle 120°

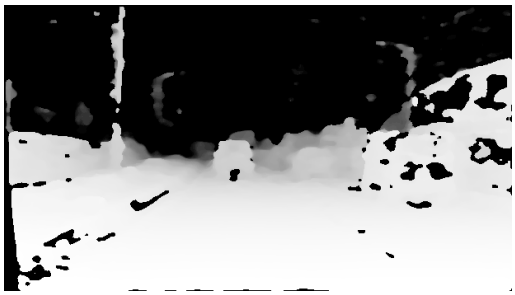


(h) Hog at angle 150°

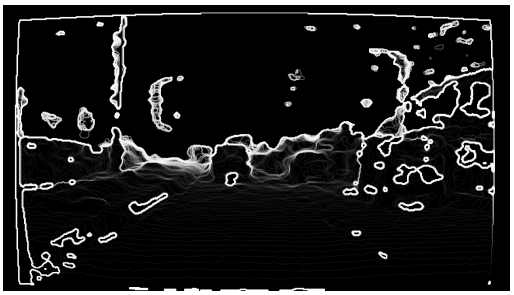
Figure 3.5: Filters applied to grayscale image to extract features. Images are processed for better viewing(increased the highlights zones).

3.2.4 Disparity image input channel

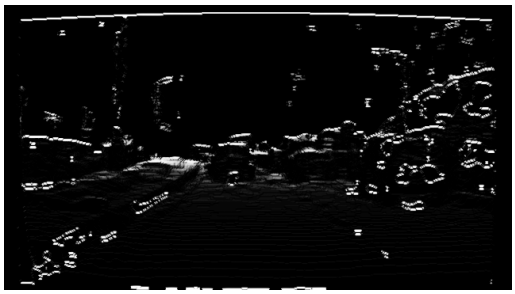
Like grayscale image, there are applied same filters on disparity image, the gradient magnitude, and the HOG(histogram of oriented gradients).



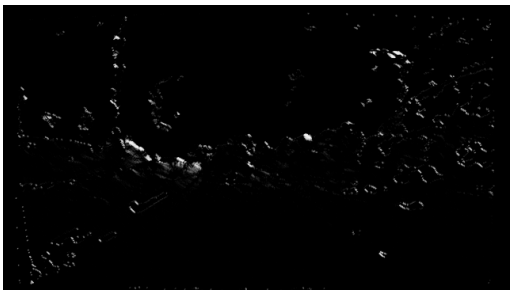
(a) Grayscale image



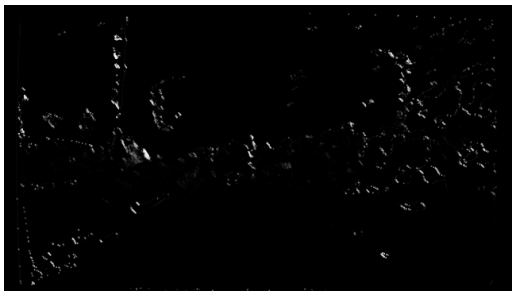
(b) Gradient Magnitude



(c) Hog at angle 0°



(d) Hog at angle 30°



(e) Hog at angle 60°

(f) Hog at angle 90°

(g) Hog at angle 120°

(h) Hog at angle 150°

Figure 3.6: Filters applied to disparity image to extract features. Images are processed for better viewing(increased the highlights zones).

3.2.5 Pre-processing and post-processing

This section will detail methods used to pre-process the input data and post-processing methods applied to detections. First of all for the grayscale images I apply a smoothing filter(triangle filter). Disparity images are computed using the rectified grayscale images. The disparity images values are stored on 2 bytes, the first 5 bits(the most significant) stores different bits like, if this pixel is unique or is repeated, the last 5 bits(the most insignificant) store details about the float part of disparity number, and the last 7 bits stores details about the integer part of the integer part. Because the grayscale images are distorted I are forced to apply a distortion to disparity images. To make a correlation between the grayscale images and disparity images. After I applied the distortion, I tested and came to the conclusion that applying a median filter to the disparity images removes the noise from it. The valid depth are between 0m and 113m(inclusive).

After I trained the model with this values, I set a threshold to the maximum distance and set this value to 50m(inclusive), and I made a comparison between this 2 models results and found out that the model with 50m has the best results.

For post-processing, on the detection I have set a minimum score 35 needed for a detection to be valid. For non-maximum suppression the threshold is set to 0.6 which means if 2 detections overlap more than 60% than the detection with lower score is removed. Another post-processing feature is that if there are 2 detections and this 2 overlap more than 20% than, this detections becomes one. Why I use this combination? Because there are o lot of different types of poles, there are small poles, there are tall poles(for illuminating the street), and the tall poles should be detected 2 times, the top part and the bottom part and if this 2 detections overlap more than 20% it means that there are chances to be only one pole.

3.3 Final architecture and algorithm

The next figure show the final architecture of the model for training. The pre-processing and post-processing parts are detailed in Subsection 3.2.5

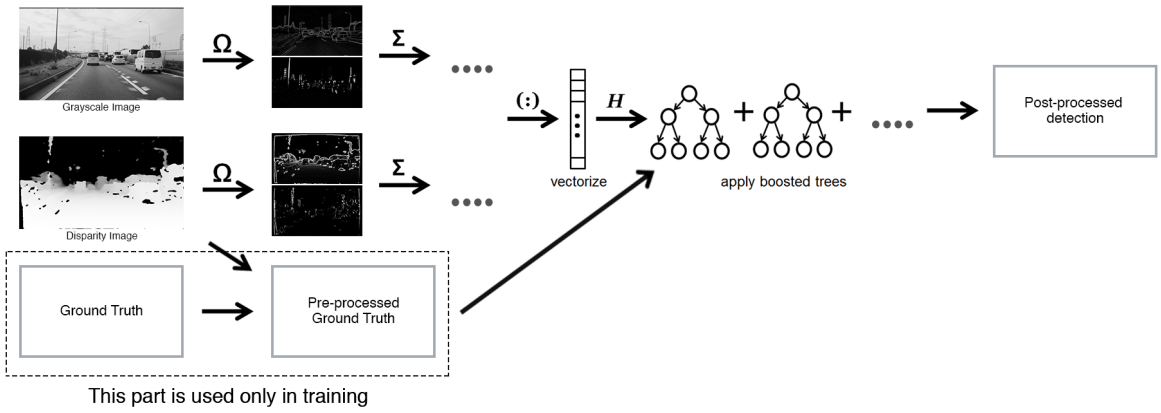


Figure 3.7: Toolbox final architecture after extension of existent toolbox.[4].[3]

3.4 Comparison of results

In this section I will make a comparison between the 2 models, the first one is the model that was trained using the grayscale images and disparity images, and the second model which was trained using only grayscale images.

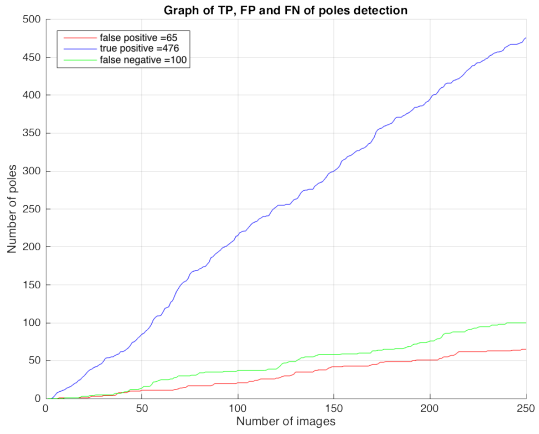
First of all the first model use a pre-processing technique(described at Section 3.2.5) which eliminates poles that doesn't have enough information, here I talk about insufficient disparity pixels. If a bounding box has valid disparity pixels less than 50% of its size then the sample is not used as positive sample. The second model doesn't have this facility, because this doesn't have information about the z axis.

But both of them have a facility of pre-processing, a bounding box with a smaller surface area than 0.2% of image size will be disregarded as being positive sample. Let's take an example, grayscale images are 1280x720 resulting a surface of 921600 pixels

$$\frac{0.2}{100} * 921600 = 1843,2$$

we assume that a pole has an aspect ratio of 1:12, a bounding box based on this aspect ratio should be $\approx$ 13 pixels width and $\approx$ 156 pixels height, which I think it doesn't have enough information neither in grayscale image, neither in disparity image.

The next 2 Figures presents the results of the models on testing set.

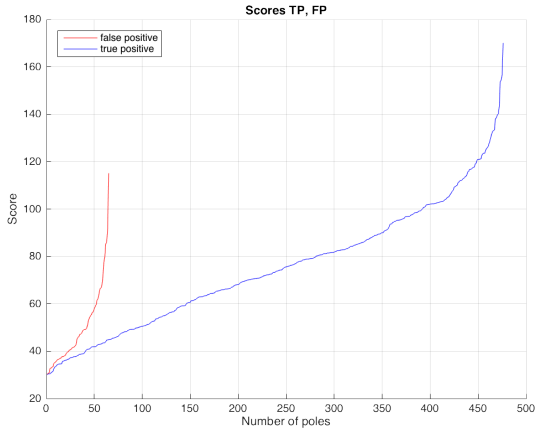


(a) Grayscale + Disparity

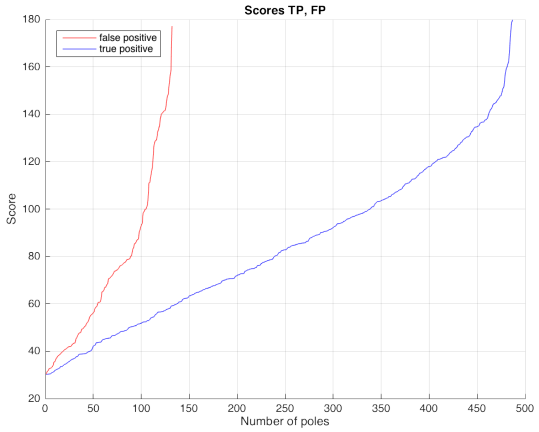
(b) Grayscale

Figure 3.8: Results on testing set

In panel a) is the model trained with grayscale images and disparity images, having a pre-processing feature based on disparity image, in panel b) is the model trained only with grayscale image, we can see that in panel b) there are twice false positive detections comparing with panel a), and there are a lot more false negative. In my opinion the disparity channel is an improvement.



(a) Grayscale + Disparity



(b) Grayscale

Figure 3.9: Detection scores

Chapter 4

Testing and Validation

In order to review the performance of the model, its error on testing and training sets are reviewed. Usually the testing error is the lowest, due to the fact that it contains the data on which the model was actually trained.

4.1 Accuracy on training and testing sets

This thesis license purpose is to prove that the disparity channel input is an improvement added to the classifier, this is why I don't have many data for training and for testing. There is only one data set used for training and for testing having various scenarios. The data set contains 250 images with approximately 933 poles, resulting 933 positive samples before **pre-processing**. The toolbox extract the negative samples by using the positive images and the negative samples are extracted from positive images subtracting the positive samples. From 250 images resulted 6048 negative samples.

The left side of Figure 4.1 shows the evolution of detections, I can see that at the end of all 250 images the true positive value is 431, the false positive value is 70 and false negative value is 117, which means that the detector accuracy is approximately 97%.

The right side of Figure 4.1 shows detections scores sorted ascending. One example of the interpretation of this figure could be: "There are 50 detections that have a score between 30 and 60".

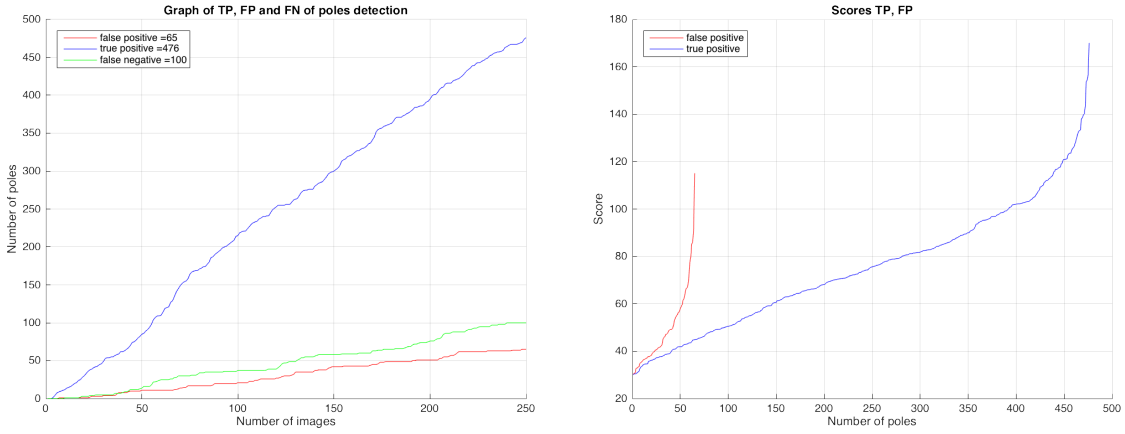


Figure 4.1: Detection evolution using model with grayscale image and disparity image and score evolution an false positive and true positive detections

The Figure 4.2 shows much more complex results of this model, model trained with grayscale channel and disparity channel as input as a table form.

		Bounding box(labeled or generated)		
		Condition Positive	Condition Negative	
Pole detection test outcome	Test outcome positive	**True Positive** (TP) = 476	**False Positive** (FP) = 65	Positive predictive value = TP / (TP + FP) = 476 / (476 + 65) ≈ 88%
	Test outcome negative	**False Negative** (FN) = 100	**True Negative** (TN) = 6340	Negative predictive value = TN / (FN + TN) = 6340 / (100 + 6340) ≈ 98%
		Sensitivity = TP / (TP + FN) = 476 / (476 + 100) ≈ 82%	**Specificity** = TN / (FP + TN) = 6340 / (65 + 6340) ≈ 99%	

Figure 4.2: Detector Results

4.2 Overview and visualization of results

The results obtained were very good. The model identified poles at various depth, with different sizes and different types. But because of noisy disparity map and the invalid pixels depth, in some cases the model introduced false positive detection.

- Poles at different depth and different size.

Figure 4.3: Poles Detection (Different Depth)

- Pole obscured by vegetation.

Figure 4.4: Pole obscured by vegetation

- False pole detection on truck.

Figure 4.5: False Detection

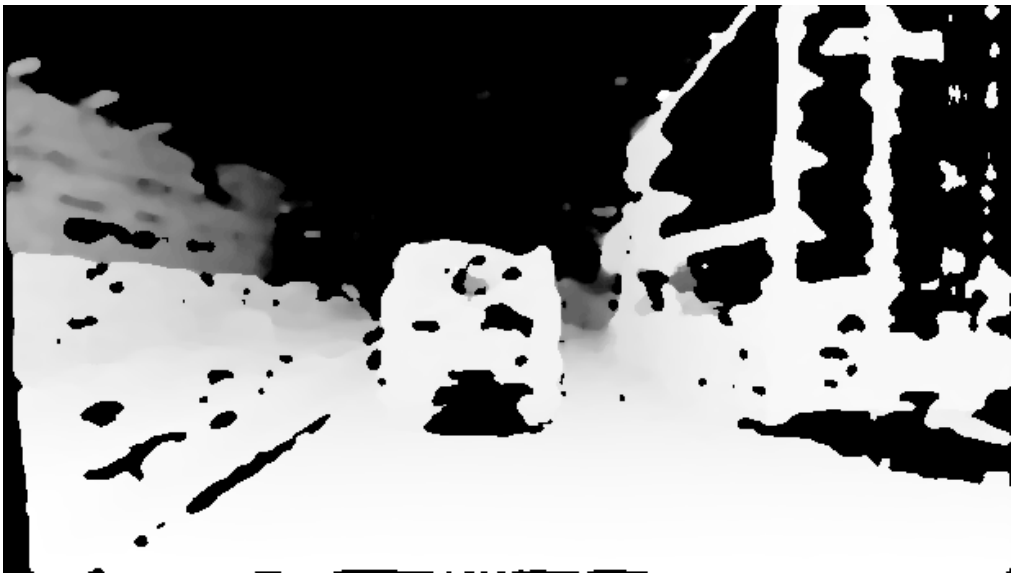


Figure 4.6: Disparity image according to Figure 4.5

Chapter 5

User's manual

This chapter provides a description on how to install the required software for both using model, prediction and training model. Also describes how to use the minimalist application to visualize the prediction.

5.1 Minimum and recommended requirements

The minimum requirement for using the matlab machine learning model is having a PC running a Microsoft Windows operating system or an Mac OS X. All the software libraries, drivers and toolboxes that are needed are detailed in next section. The model was tested for Microsoft Windows and macOS Sierra(10.12.6), it is not guaranteed that they will work correctly for other operating system. Additionally, if it is desired to furtherly train, or re-train the model, it is recommended that a powerful GPU is used in combination with "Parallel Computing Toolbox" for matlab.

5.2 Software and toolbox installation

The installation of the required software for training and predicting using the model is described in several steps:

1. Download and install Matlab using the following link `https://www.mathworks.com/products/matlab.html`
2. Optional: Download and install Parallel Computing Toolbox using the following link `https://www.mathworks.com/products/parallel-computing.html`

3. After installing Matlab, set path of toolbox sources by adding the root folder of toolbox to the 'Set Path' sub-menu from 'Home' menu.

4. For client side we need to install Java JRE 8 to run the application. Install by using the URL:`http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html`

5.3 Using the toolbox and the application

Detectors are stored in files that have the ".mat" extension. They store the decision tree and all associated information related to a model.

5.3.1 Training

A detector can be trained by specifying the input grayscale image folder, disparity image folder and annotation folder, after initializing with default values.

```
opts = acfTrain();

opts.posGtDir = groundTruthFolderPath;
opts.posImgDir = grayImageFolderPath;
opts.dispPgmDir = disparityImageFolderPath;
opts.name = detectorPathFolder;

pLoad={'lbls', {'pole'}, 'ilbls', {''} };
opts.pLoad = [pLoad 'arRng', [-inf inf]];

% train detector (see acfTrain)
detector = acfTrain( opts );
```

The first line initialize the detector's options with default values. Next 4 lines set the folder path for grayscale images, disparity images, annotation files and detector path. Next 2 lines are specifying what annotations consider valid or not. The last line train the model based on the specified parameters.

5.3.2 Prediction

New predictions can be made, by giving the trained model a gray image and a disparity image, and using its output(bounding boxes), by executing a Matlab script with the following lines:

```
name = detectorFolderPath;
% load the detector
detector = load([name 'Detector.mat']);
detector = detector.detector;

% load gray and disparity images
opts = detector.opts;
imgGray = imread(grayImagePath);
imgDisp = imread(disparityImagePath);
img = ImgAndDisp2Img(Img, ImgDisp, opts.pPyramid.pChns.pDisparity);
dt = acfDetect(img, detector);
```

The first line specified where the model is stored for mac respectively, for windows. The next 2 lines load the model into variable detector. Next line load the parameters of the detector into variable opts, next 2 lines read the grayscale image and the disparity image. Next line is one of the most important step, it's combining the grayscale image and the disparity image into one single image(input image for model), the disparity is the second channel. The last line make predictions based on the input image and returns an array of detections.

5.3.3 Communication between server and client

The application consists of two "app", one is the client part which is written in Java and the second part is the server which is written in Matlab. The communication between server and client is made by using sockets. The server opens a connection at default IP Address "0.0.0.0" using port "30000" and it's waiting for the client response. The client connects to the server address and they begin communicating. The Figure 5.1 show has the communication is realized.

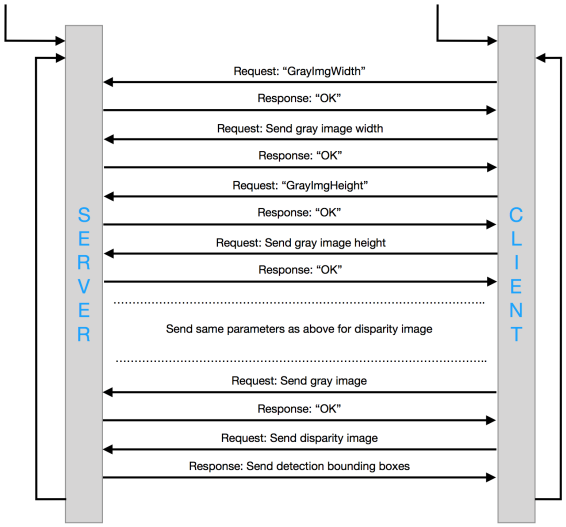


Figure 5.1: Communication between Server and Client

5.3.4 Visualization

Visualization can be done by using the client app. After connecting to the server, only 2 things are needed, the grayscale images folder path and disparity images folder path, after setting this 2 values press the **Send** button and wait for the server to process the data. The Figure 5.2 shows the client interface. 1 - server address(text field), 2 - server port(text field), 3 - Connect/Disconnect(button), 4 - grayscale images folder path(text field), 5 - disparity images folder path(text field), 6 - Send/Stop(button)

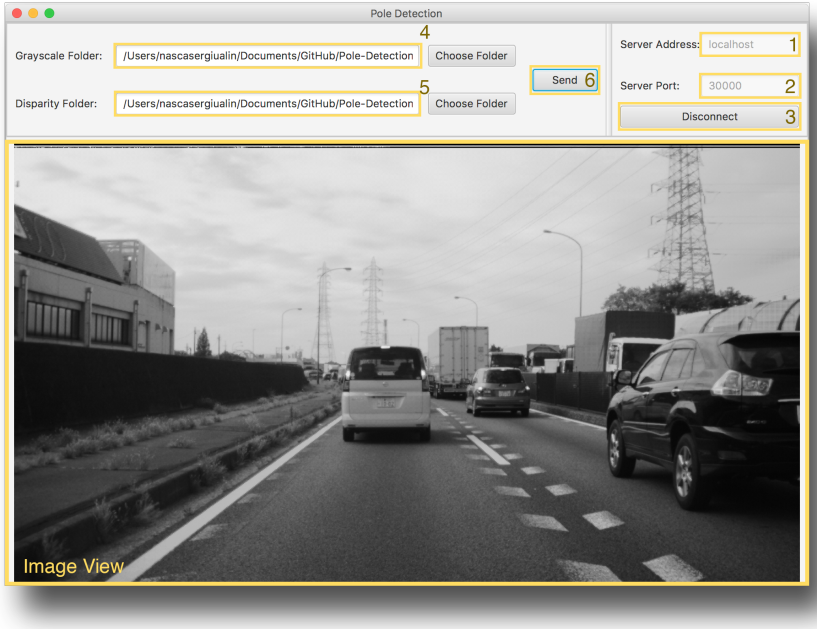


Figure 5.2: Client interface(Java)

Chapter 6

Conclusions

6.1 Summary and contribution to field

6.1.1 Results

The work presented in this license thesis was aimed at creating and extending a system that can predict pole, street poles, based on grayscale images and disparity images. The system was implemented using machine learning algorithm and decision trees; The base of the system was implemented in a matlab toolbox from Piotr Dollar and extended by me. The thesis shows that the poles are one of the many object that a machine learning algorithm can learn. The thesis proves that the disparity image is a great input channel, because it has information about z axis(depth axis) and this is a great information for the system and for the classifier.

6.1.2 Importance

The results obtained by the presented implementation prove the fact that autonomous driving is within reach. Although current systems do not have the desired reliability to be implemented on production cars; Research of new methods and implementations, such as the one presented in this license thesis, move a step closer towards the final goal: autonomous driving that has a better performance than human driving. However, current systems could be used as driver assistance rather than full driver replacement, which can increase the safety of modern commutes. The pole detection can be used to localize the car at high precision and can localize the surrounding object like, street signs or road or any other static object that can help the car to make a vision about the current situation and help it to make the best decision. Moreover,

the results obtained prove that machine learning or any artificial intelligence algorithm shows a great potential in solving human tasks. However along the road advancements can only be made through research.

6.2 Further development

6.2.1 Larger Dataset

The performance of the implemented machine learning system can be improved in several ways. Firstly, training it on a larger dataset would increase its performance and the detection rate, it will 'learn' different types of poles and different orientation of it. The model implemented in this thesis was trained on a very small dataset, because it was very hard to find relevant sequence of images with different types of poles and different position of it, but I think the algorithm has potential, yet for an improvement a much larger dataset is needed.

6.2.2 Better disparity image

Figure 6.1: Noisy disparity image

The disparity image is very noisy(it has a lot of pixels that have invalid disparity value) Figure 6.1, and this noise appears because a point(region of pixels) from the left image is searched in the right image in a bounding box and there are multiple match between them. I applied some filter for smoothing the image, first of all, I applied a median filter[13] which filters almost all the noise from disparity image, see Figure 6.2. At this moment this are the filters applied to improve disparity map. There are more and better techniques to do it better, but it's not the purpose of this license thesis. One of the better technique is to modify disparity based on gray scale image. A better disparity image will increase the performance of the algorithm.

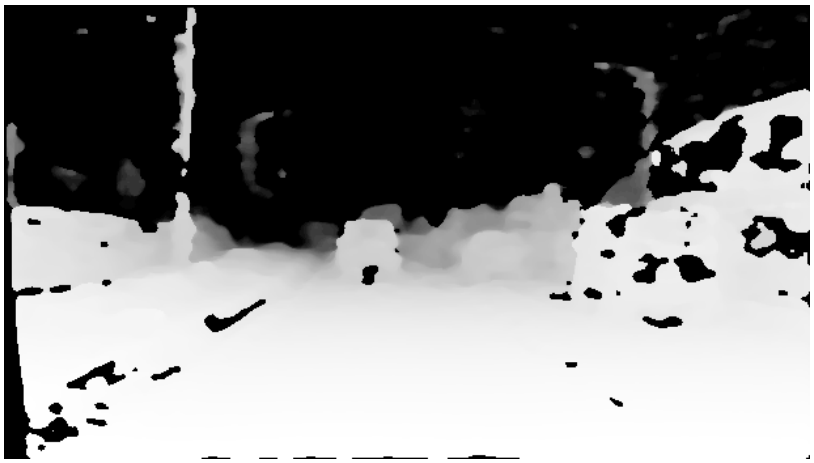


Figure 6.2: Disparity images smoothed with median filter

6.2.3 Object Tracking

The third way to improve the detection rate and the performance of the algorithm is to track the detection in future images. We know that the car is a dynamic object, using some features from GPS like position, steering degree, and current speed. We can calculate the trajectory and the velocity of the car, and we also know that a pole is a static object; so we can calculate where should be the pole in the next frame. If camera shots at 8 frames per second it means that one frame is taken at 125 milliseconds(it's a very short time from one frame to the next one). But this feature can be used only in an image sequence.

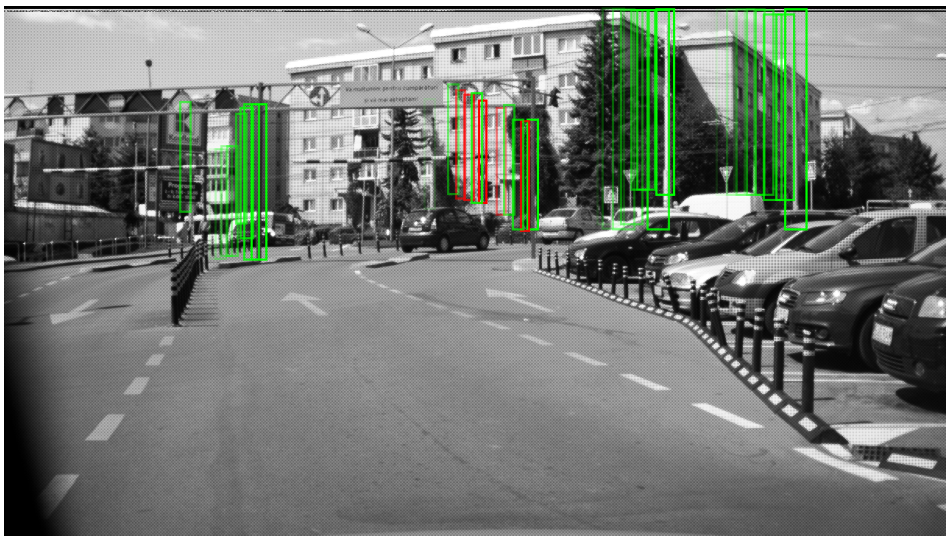


Figure 6.3: This figure shows detection of poles in an image sequence; with green are represented detections and with red the missing detections. The missing detections could be identified by an algorithm for object tracking.

Bibliography

[1] Leo Breiman. Bias, variance, and arcing classifiers. Technical report, 1996.

[2] Jason Brownlee. Boosting and adaboost for machine learning.

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, June 2005.

[4] Piotr Dollár, Ron Appel, Serge Belongie, and Pietro Perona. Fast feature pyramids for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(8):1532–1545, January 2014.

[5] Piotr Dollár, Zhuowen Tu, Pietro Perona, and Serge Belongie. Integral channel features. *Proceedings of the British Machine Vision Conference*, pages 91.1–91.11, September 2009.

[6] N. Kanopoulos, N. Vasanthavada, and R. L. Baker. Design of an image edge detection filter using the sobel operator. *IEEE Journal of Solid-State Circuits*, 23(2):358–367, Apr 1988.

[7] P. H. King. Digital image processing and analysis: Human and computer applications with cviptools, 2nd edition (umbaugh, s.; 2011) [book reviews]. *IEEE Pulse*, 3(4):84–85, July 2012.

[8] James Le. Machine learning.

[9] Tom Mitchell. Machine learning. 1997.

[10] Michael Montemerlo, Sebastian Thrun, Hendrik Dahlkamp, and David Stavens. Winning the darpa grand challenge with an ai robot.

[11] FAIZAN SHAIKH. Deep learning vs. machine learning. April 2017.

[12] Wikipedia. Darpa grand challenge.

[13] Wikipedia. Median filter.