AUTONOMOUS TRAIL FOLLOWING

MASOUD HOVEIDAR SEFID

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

GRADUATE PROGRAM IN COMPUTER ENGINEERING
YORK UNIVERSITY
TORONTO, ONTARIO

NOVEMBER 2017

© Masoud Hoveidar Sefid, 2017

Abstract

Trails typically lack standard markers that characterize roadways. Nevertheless, trails are useful for off-road navigation. Here, trail following problem is approached by identifying the deviation of the robot from the heading direction of the trail by fine-tuning a pre-trained Inception-V3 [1] network. Key questions considered in this work include the required number, nature and geometry of the cameras and how trail types – encoded in pre-existing maps – can be exploited in addressing this task. Through evaluation of representative image datasets and on-robot testing we found: (i) that although a single camera cannot estimate angular deviation from the heading direction, but it can reliably detect that the robot is, or is not, following the trail; (ii) that two cameras pointing towards the left and the right can be used to estimate heading reliably within a differential framework; (iii) that trail nature is a useful tool for training networks for different trail types.

Acknowledgements

I would first like to thank my thesis supervisor, Professor Michael Jenkin of the Department of Electrical Engineering and Computer Science at York University. He believed in me from the first day to the last. I would also like to gratefully acknowledge the help and support of NSERC Canadian Field Robotics Network (NCFRN) during my research on this thesis. I would also like to thank Professor Marcus Brubaker of the Department of Electrical Engineering and Computer Science at York University as my committee member, and I am gratefully indebted to him for his very valuable comments on this thesis.

Finally, I must express my very profound gratitude to my precious family, specially my father, who taught me to be a stronger man and my mother who gave me her unconditional love and made me a better person. I also like to thank my sweet sister, my charming brother and all the other family members for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Driving on modern western roads seems a relatively simple task. Indeed humans manage to do it every day. Machines have also proven to be quite expert 'road' drivers and decades of research in this area have resulted in effective systems that 'solve' the problem and are expected to lead to widely available commercial systems in the next decade. That being said, off-road navigation still remains a challenge. As roads are replaced with trails and tracks, the problem of driving along them becomes more difficult [3], [4]. Although a number of approaches exist in the literature (see [5] and [6]), no globally effective solution exists for autonomous off-road driving. The problem is that unaided navigation is very difficult. When road markings are removed, or as the normal structure of roadways becomes less well defined, there is less and less 'common' and 'standard' structure for an algorithm to latch on to in order to follow the roadway. Nevertheless, such tracks and paths are important for a range of interesting applications, from large-scale infrastructure maintenance and security, to recreational activities away from large urban centres.

Recent results in autonomous driving have demonstrated that deep neural networks can be an extremely effective approach to addressing the driving problem on-road (see [7], [8], [9] and [10]). Can a similar approach

(a) *(b)*

Figure 1.1: *(a) shows the Pioneer 3-AT robotic platform. (b) shows a dirt trail covered with leaves on York University campus captured with a wide field camera.*

work off-road? And if so, what is the best way to train and operate a neural network to follow trails and paths? Fundamentally, the premise of this thesis is that using front facing cameras onboard the robot, and exploiting a Deep Neural Network (DNN) for image classification on trails, a robot can compute a motion command that will enable the robot to follow off-road trails. In this work, the underlying goal is not to follow highly structured roadways, rather the goal is to use an adaptive algorithm to enable the following of paths, walkways and other pedestrian-related walkways. Addressing this problem requires the solution to two underlying sub-problems. The first deals with a general image classification problem – determining if the robot pointed along the trail or is it rotated to the left or to the right – and the second involves using the output of the latter in order to calculate a motion command for the robot to follow the trail. It is this first problem that is considered in this work.

1.1 The approach

The goal of this thesis is to develop algorithms and the corresponding software tools to enable an autonomous vehicle to perform point to point locomotion autonomously given the assumption that the robot is placed on a trail. Using front facing cameras, the robot maintains an ongoing representation of its orientation on the trail while planning and executing motion commands in the real world. Figure 1.1b shows a dirt trail covered with leaves and Figure 1.1a shows the robot used in this work. At each moment in time the robot captures forward-facing images of the environment and uses this to compute a local motion (a twist vector) that drives the robot along the road as highlighted in Figure 1.2. As is evident in Figure 1.1b paths are very different from the roads found in western countries. They lack straight line boundaries, traditional road markings, etc. So how can we enable to a robot to follow a trail? One property that might be exploited is that although trails come in different types - dirt, pavement, etc.– the information about the nature of a particular trail is often well known, and such information is typically stored in a map which makes the information available for both human and (potentially) machines to exploit. Existing digital maps contain information about road maps, natural and man-made landmarks, trail routes, rivers and etc. Is it possible or necessary to integrate 'trail type' information into the roadway following algorithm? A second question involves determining what is the best way to characterize the pathway in front of the robot. Is one camera sufficient or should multiple cameras be used? In either case, what is the best way to learn the characterization of the pathway so that the robot can be driven along the path?

This thesis considers the problem of trail following using one or more cameras for an autonomous outdoor vehicle. Furthermore, it postulates this problem in terms of the development of a convolutional neural net (CNN) that is trained using transfer learning, that is that is trained by adapting a pre-trained network from another task. Key questions that this thesis considers include:

1. Can fine tuning a pre-trained DNN model be used for trail following classification task? Pre-trained

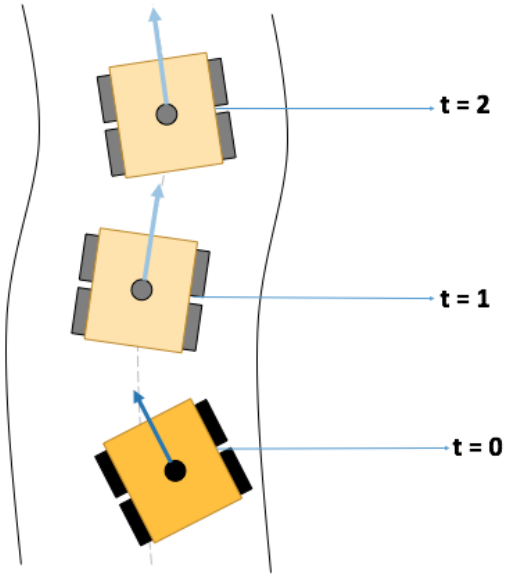


Figure 1.2: *Visualization of the trail following task. The robot at time $t = 0$ is placed on trail and has a certain deviation compared to the heading direction of the trail. Robot publishes appropriate motion command in order to follow the trail direction and compensate its deviation with a proper angular adjustment command. At times $t = 1$ and $t = 2$ it is maintaining its direction along the trail.*

neural networks have been successfully adapted to other tasks, can a similar approach be used for trail following? If so, this will enable the CNN to be trained in a much more efficient manner and to exploit advances in pre-trained networks as they are developed.

2. What is the effect camera's field of view on the trail following task? Cameras can be equipped with lenses that have different focal lengths. Long focal lengths 'focus' light from a narrow region of the scene, while wide focal length cameras 'see' more of the scene but at a lower resolution.

3. Is one camera sufficient to drive the robot along the path or should multiple cameras be used to control the vehicle? In terms of the framework considered here, this involves answering the question "is a single camera sufficient to characterize the orientation of the pathway in front of the vehicle"? Or should multiple cameras be used to address this question?

4. Is it desirable to exploit the expected nature of the trail when training a neural network to follow the trail itself? Trails come in a range of different flavors, from structures that 'look' like roads to tracks to pedestrian walkways. Is it possible to train a single DNN for all possible trail types or is better performance found when different networks are trained for different trails? If it is necessary to train different networks for different trails, is it possible to exploit standard existing maps of such trails in order to choose which network to use?

In any autonomous vehicle research effort the actual task of deploying robots and sensors in the field consumes considerable resources and time. This work uses the Pioneer 3-AT robot [11] shown in Figure 1.1a as an experimental platform. This robot is a skid steer vehicle that has an onboard processor to control the basic motion of the vehicle. Running the Robot Operating System (ROS) [12], the computational capabilities of the vehicle is augmented using an onboard laptop. In terms of sensing, the vehicle is augmented with two wide field of view cameras for trail navigation. In particular the Pioneer 3-AT robot has been augmented with a mount for wide field cameras. These cameras are used to capture datasets to train the CNN, and to drive the vehicle.

1.2 Structure of this thesis

This thesis is organized into five chapters. This chapter provided an introduction about the autonomous driving task on trails and states its challenges, described the basic questions to be asked in this work: can transfer learning on a CNN be used to enable an autonomous robot to perform trail following? Furthermore, it describes a number of questions that this thesis will attempt to answer about the nature of the training process and the need or desirability to use different networks for different trail types. Chapter 2 provides a literature review about neural networks and DNN and specifically Convolutional Neural Networks (CNN), their use in the autonomous driving task, and other classical approaches for autonomous driving. Chapter 2 also provides an example of a classical image processing-based approach for trail following. Beyond providing

details of the problem of trail or road following, this review further highlights the differences between classic trail following algorithms and CNN/DNN-based approaches. Chapter 3 presents a transfer learning approach to the use of CNN for trail following. The algorithm developed in Chapter 3 uses a classification-based approach to identify the likely misalignment of the vehicle with respect to the roadway and then uses an offset camera difference process to integrate information from three or more classifiers to identify the appropriate steering angle. Chapter 4 provides experimental results and the evaluation of different sensors and tuning approaches to the problem of CNN-based trail following. In particular, it considers the impact of different training categories, different road types, camera number and different camera field of view/focal lengths on trail following performance. This evaluation is provided both in terms of trail orientation classification as well as in terms of the actual ability of the robot to follow the trail. Finally, Chapter 5 provides a summary of this research along with a description of possible future work in this area.

Chapter 2

Background

This thesis deals with the task of developing an algorithm that enables a robot to follow off-road trails using vision. In developing a trail-following algorithm one basic decision that must be made is whether to base the approach on traditional image- and sensor data-processing algorithms or to choose to use a deep neural network-based approach such as convolutional neural networks (CNN). In this work a CNN-based approach is followed. CNN have proven to be an effective way of solving complex and ill-posed problems, and in particular they have proven to be an effective approach for road-following. It is thus expected that a CNN-based approach should work well for trail following as well. Although this work concentrates on a CNN-based approach, for completeness here we review both traditional and neural network-based approaches to the problem of trail following. We start with a review of deep neural networks and then move on to the problem of trail following using both traditional and neural network-based approaches.

Neural network (NN) approaches have been used for pattern recognition and object detection purposes for many years and early NN approaches to road following can be traced back at least to the 1980's and the

ALVINN project [13]. This work was followed by many other systems. For example, Jeong et al., exploited an artificial neural network to detect road boundaries using linear dynamic equations and Kalman filters[14]. while in [15], Moghadam et al. present a self-supervised learning algorithm for terrain classification that exploits near-field stereo vision in front of the robot and which combines this information with terrain features extracted from monocular vision. Although these and other NN-based approaches of this time showed some successes, in general the NN approaches of this period could be described as limited, as they needed to be tuned carefully in order to build a feature extractor to process natural data in different domains. Furthermore, the lack of computational power at the time limited the depth of such networks and such approaches fell out of favor as more traditional image-processing based approaches proved more successful. With the advent of more powerful and readily available parallel computing hardware, and the development of more sophisticated NN architectures in the 2010's, deep neural networks (neural networks with multiple hidden layers) have now found wide application across a range of computational problems and have performance characteristics that are highly competitive and in some cases superior to more traditional approaches. Convolutional Neural Networks (CNN) [2], deep neural networks with complex internal architectures rather than the fully connected architecture associated with neural networks generally, are now found in domains from large-scale automatic image colorization [16] to object recognition and classification [17] and in particular have found good use in the control of autonomous vehicles (see [18], [19] and [7]). CNN and deep learning approaches have demonstrated good performance in a range of different domains of Artificial Intelligence (AI) and there is hope that DNN-based approaches will be able to solve an even wider range of problems in the field in the future [20]. Two main reason for the success of today's deep learning relative to the neural networks of the 1980's are 1) the availability massive available labeled datasets to train the network and 2) the tremendous capacities of graphics computing units (GPU) to train the network [20].

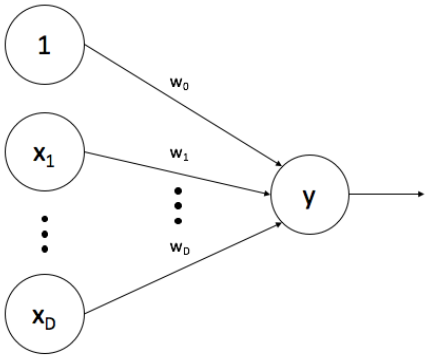


Figure 2.1: *A block diagram of processing block y in a neural network which accepts D inputs. In order to simplify the mathematical model of the computation at the node y, it is typical to also add a constant input* 1 *to provide a bias term to the computation.*

2.1 Deep Neural Networks

Neural networks are inspired by the way human brain learns and represents information through interconnected neurons. This section provides a brief introduction on the basics of neural networks and follows the mathematical notation described in [21] (for a more detailed introduction to the field see [22] and [23]). Convolutional neural networks first were introduced by LeCun et al. in [2]. Convolutional neural networks accept data, such as images as input and they learn a set of features and use them for the task for which they have been trained. These features are extracted and learned in a hierarchical manner and this helps a CNN to learn complex features and also to be reasonably invariant to translation and distortions in the image using these deep and complex features in higher layers of the hierarchy.

2.1.1 From neural networks to deep learning

Neural networks consist of a collection of processing blocks (also called nodes or neurons) which are interconnected in a feed forward way. Each node accepts a set of inputs as $1, x_1, x_2, ..., x_D$ and computes an output

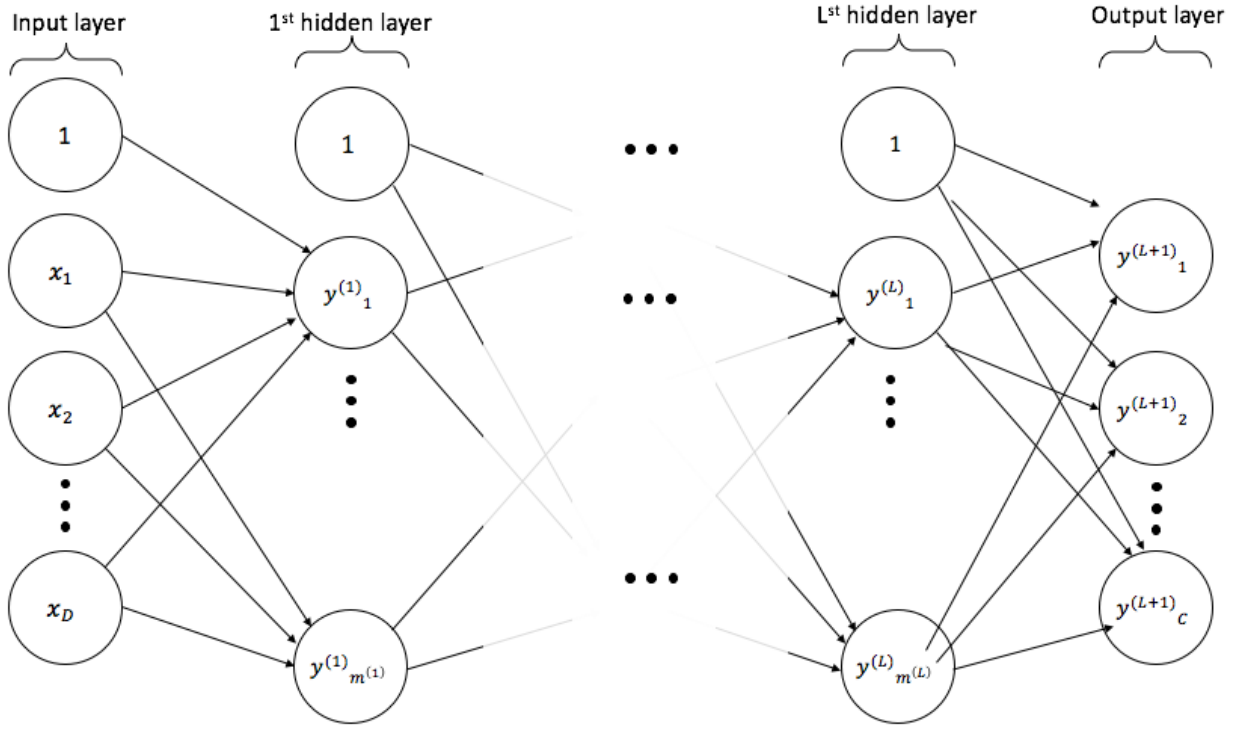


Figure 2.2: *Multilayer perceptron Neural Network graph. It is a* $(L+1)$*-layer neural network with* D *input nodes and* C *output nodes. Redrawn from [21].*

y (see Figure 2.1). A processing block computes $y = f(z)$ where f is the activation function. Typically f is described as a dot product between the input vector $[1, x_1, x_2, ..., x_D]$ and a weight vector $[w_0, w_1, ...w_D]$ followed by an activation function f. As shown in Figure 2.1 a single processing block can be drawn as a directed graph. A processing block accepts an arbitrary but fixed number of inputs and publishes an output that can be fed to the next layer as an input as shown in Figure 2.2. Given a fixed structure for the NN, the process of training the NN involves determining the set of weights that define the computation.

Multilayer perceptron. Figure 2.2 depicts a simple $(L+1)$-layer network which has D inputs and returns C outputs and between which are L layers. These inner layers are known as hidden layers. Each layer consists

of an arbitrary number of processing blocks. The i^{th} node in layer l calculates it's output as

$$y_i^{(l)} = f_i(z_i^{(l)}) \textbf{ with } z_i^{(l)} = \sum_{k=0}^{m^{(l-1)}} w_{i,k}^{(l)} y_k^{(l-1)} \tag{2.1}$$

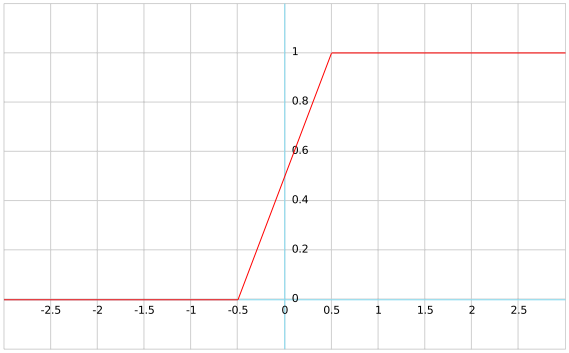
where $w_{i,k}^{(l)}$ is the weighted interconnection between i^{th} node in layer l and k^{th} node in the previous layer $(l-1)$. $w_{i,0}^{(l)}$ is the bias term and $m^{(l)}$ is the number of nodes in layer l. $D = m^{(0)}$ and $C = m^{(L+1)}$. We define a constant $y_0^{(l)} \triangleq 1$ in each layer in order to incorporate a bias term in that layer. Using the vector representations of w, y and z, then Equation 2.1 can be written as

$$z^{(l)} = w^{(l)}.y^{(l-1)} + y_0^{(l)} \tag{2.2}$$

where $z^{(l)}$, $w^{(l)}$ and $y^{(l-1)}$ correspond to the matrices and vectors of all the inputs $z_i^{(l)}$, weights $w_{i,k}^{(l)}$ and outputs $y_k^{(l-1)}$, respectively. The term 'deep neural networks' is used to describe networks that have more than three hidden layers and the process of training them is called deep learning in the literature [24]. Generally, in neural network computations it is desirable to scale all the inputs and outputs in the range of $[-1, 1]$, as sometimes inputs of a network have different scales and higher numbers of one input can totally diminish small values of another input.

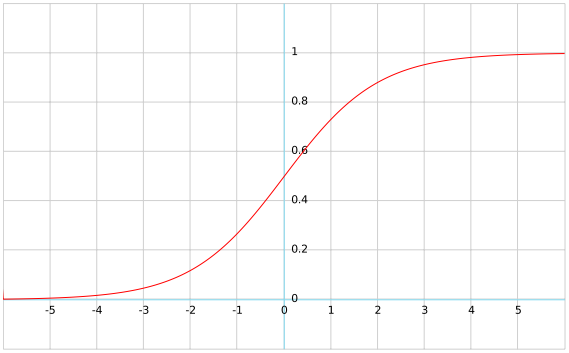
Activation functions. The activation function f maps the dot product from the output at the previous level. There are a large number of activation functions in the literature (see [25] and [21]), but in general activation functions can be divided to different types: piecewise-linear functions such as threshold functions and non-saturating activation functions (e.g., Rectified Linear Units [17] or ReLUs for short), and also sigmoid functions. These four types of activation function are illustrated in Figure 2.3. Although DNN and CNN can be found using all of these activation functions, the use of ReLUs enhances the performance of supervised learning of DNN as their derivative is 1, which provides considerable computational advantages over other activation functions such as those shown in Figure 2.3.

Supervised learning. The task of computing the network weights in such a way that the network can approximate or learn a target mapping $g : x \rightarrow t$, is called supervised learning [21]. In order to find the



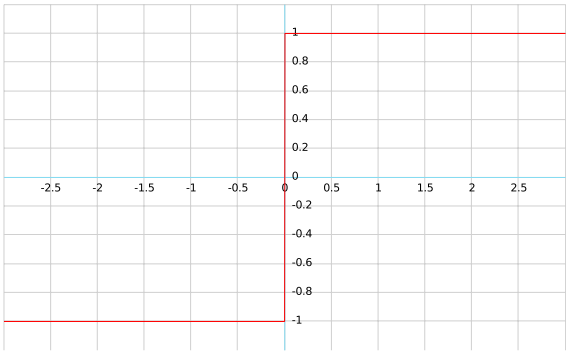
$$f(z) = \begin{cases} 1, & z \geq 0.5 \\ z + 0.5, & -0.5 < z < 0.5 \\ 0, & z \leq -0.5 \end{cases}$$

(a) *Piecewise linear activation function.*



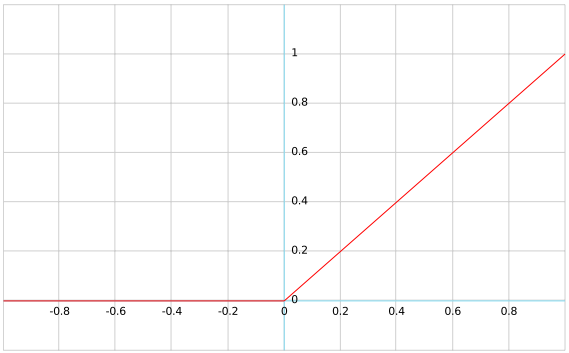
$$f(z) = \frac{1}{1+e^{-z}}$$

(b) *Logistic sigmoid activation function.*



$$f(z) = \begin{cases} 1, & z > 0 \\ 0, & z = 0 \\ -1, & z < 0 \end{cases}$$

(c) *Threshold activation function.*



$$f(z) = max(0, z)$$

(d) *ReLU activation function.*

Figure 2.3: *Activation functions.*

weights that define g, a training set is used. The training set T_s is a collection of pairs that define the function that the network should learn and is defined as

$$T_s := \{(x_n, t_n) : 1 \leq n \leq N\} \tag{2.3}$$

where g maps inputs of x_n to desired target values t_n (i.e., $t_n \approx g(x_n)$).

Error measurement. Training a neural network is typically presented as a task that aims to minimize the objective function based on the error between target values t_n and the output of the network $y(x_n)$ through a process of refining the weights associated with the neurons (nodes) of the network. Two well known error measures are the sum of squared error:

$$E(w) = \sum_{n=1}^{N} \sum_{k=1}^{C} ||y_k(x_n, w) - t_{n,k}||^2 \tag{2.4}$$

and the cross-entropy measure:

$$E(w) = \sum_{n=1}^{N} E_n(w) = \sum_{n=1}^{N} \sum_{k=1}^{C} t_{n,k} \log(y_k(x_n, w)) \tag{2.5}$$

where $t_{n,k}$ denotes the k^{th} element of the target value t_n. Although the sum of squared error is perhaps the simplest error function, the cross-entropy error function is often preferred to the sum of squares error method for classification task.

Training approaches. Training a neural network involves updating the weights of the network to minimize the error function. Training approaches can be categorized into four main groups [21]:

- Stochastic training: In this method, one of the input values of training set is chosen randomly for training.
- Batch training: In this approach, all of the input values are used in order to update the weights with minimizing the error over all inputs as $E(w) = \sum_{n=1}^{N} E_n(w)$.

- Mini-batch training: This approach uses a combination of stochastic training and batch training in which a subset M (mini-batch) of the training set is randomly chosen, where $M \subset \{1, 2, ..., N\}$ and the weight updating process is computed based on the cumulative error of $E_M(w) := \sum_{n \in M} E_n(w)$.

- Online training: Every single input value will only be used once in the training process and updating the weights based on the error of $E_n(w)$.

Stochastic approaches have shown promising results in recent architectures and are perhaps the only practical way to deal with large amount of training data [21].

Parameter optimization process. In order to train the network, the objective function E_n is minimized with respect to the weight values w. The basic approach is to optimize the weights until no further change can be found in the error function with changes in the weights, or more specifically to optimize the weights until:

$$\left| \frac{\partial E_n}{\partial w} \right| \approx 0. \tag{2.6}$$

In order to find a solution to this equation, due to the complexity of error function E_n, iterative numerical methods are used. Weight values are updated in each iteration using a linear weighting updating process

$$w[t+1] = w[t] + \Delta w[t] \tag{2.7}$$

where $w[t]$ is the weight vector in the i^{th} iteration of the computation and $\Delta w[t]$ is the weight update in the step. In order to find $\Delta w[t]$ in each step, a Gradient descent approach is typically used [21]. The basic idea of the gradient descent approach is to take steps towards the steepest gradient of the error E_n in the direction of the negative derivatives, to reach to the minima of the error function. Hence, $\Delta w[t]$ is calculated as

$$\Delta w[t] = -\gamma \frac{\partial E_n}{\partial w[t]} = -\gamma \nabla E_n(w[t]) \tag{2.8}$$

where ∇E_n is the gradient of error E_n with respect to w and γ is the learning rate [21].

Weight Initialization. Weight initialization is crucial step in the optimization process. Some method is required to initialize the weights of the network prior to learning. A number of approaches are described in the literature (see [25] for example). Stochastic approaches are the most appropriate. One method to initialize weights stochastically is presented in [26]. It defines the range for weights to be initialized in as

$$-\frac{1}{\sqrt{m^{(l-1)}}} < w_{i,j}^{(l)} < \frac{1}{\sqrt{m^{(l-1)}}} \tag{2.9}$$

where m is the number of nodes in a layer. This range has been chosen under the assumption that all the input values are in range of $[-1, 1]$ and belong to a Gaussian distribution. Another approach for weight initialization is called "normalized initialization" and defines the range as:

$$-\frac{\sqrt{6}}{\sqrt{m^{(l-1)} + m^{(l)}}} < w_{i,j}^{(l)} < \frac{\sqrt{6}}{\sqrt{m^{(l-1)} + m^{(l)}}}. \tag{2.10}$$

This method leads to improved learning based on the experimental results presented in [27] and because of this is used in more recent network architectures.

Learning via backpropagation. There are a number of algorithms that can be used to train a supervised NN (e.g., [28] and [25]) but perhaps the most well known and often used is the backpropogation algorithm [21].

Within the structure of a neural network, gradient descent can be realized through the propagation of errors backward through the network [21]. In fact backpropogation is an iterative process of propagating output errors $E_n(w)$ through the levels of the network to refine the weights w to minimize $E_n(w)$. The basic algorithm is as follows:

1. Feed all the inputs values x_n to the network and calculate the input and output of each node throughout the entire network. This is known as the feedforward phase of the algorithm.

2. Compute the error values $E_n(w)$ of output layer $\delta_i^{(L+1)}$ and compute local error for each element of the output layer

$$\delta_i^{(L+1)} := \frac{\partial E_n}{\partial y_i^{(L+1)}} f'(z_i^{(L+1)}). \tag{2.11}$$

3. Calculate the error $\delta_i^{(l)}$ of each hidden layer l by propagating the errors backward in the network

$$\delta_i^{(l)} := f'(z_i^{(l)}) \sum_{k=1}^{m^{(l+1)}} w_{i,k}^{(l+1)} \delta_k^{(l+1)}. \tag{2.12}$$

4. Determine the derivatives of the error function throughout the network

$$\frac{\partial E_n}{\partial w_{j,i}^{(l)}} = \delta_j^{(l)} y_i^{(l-1)}. \tag{2.13}$$

5. Update individual network weights using:

$$\Delta w[t] = -\gamma \frac{\partial E_n}{\partial w[t]} \textbf{ and } w[t+1] = w[t] + \Delta w[t] \tag{2.14}$$

Using this algorithm iteratively refines the weights w so as to reduce the $E_n(w)$ function based on the training regime followed. Figure 2.4 shows the feedforward and backpropagation of errors in the weight updating process on a sample neural network.

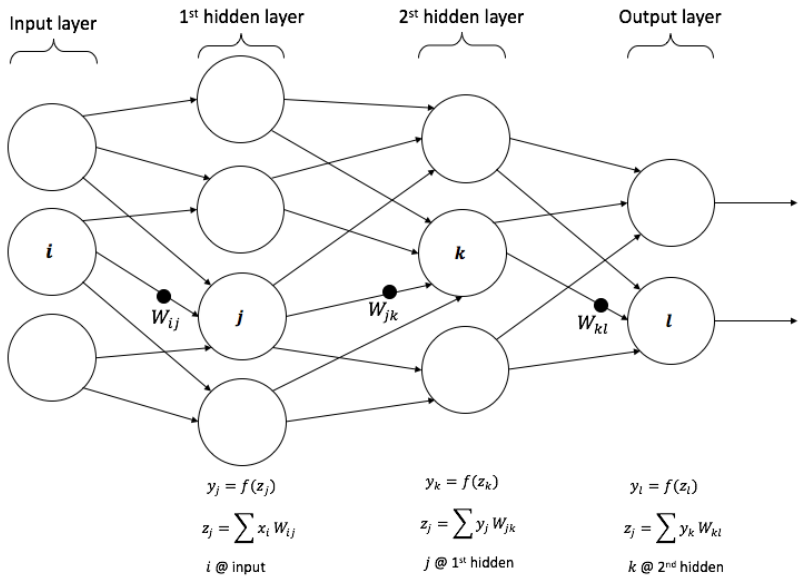
2.1.2 Regularization

One issue that arises in training a neural network is the problem of overfitting [25]. Overfitting occurs when a network has a substantially lower error value for the training set in comparison with the error of the test set (i.e., any unseen data for the network which has similar criteria as the training set). Regularization methods help to generalize the network model for both seen and unseen data simultaneously [21]. One regularization approach is through the addition of a regularizing term to the error function E_n in order to control the form and complexity of the solution [25]:

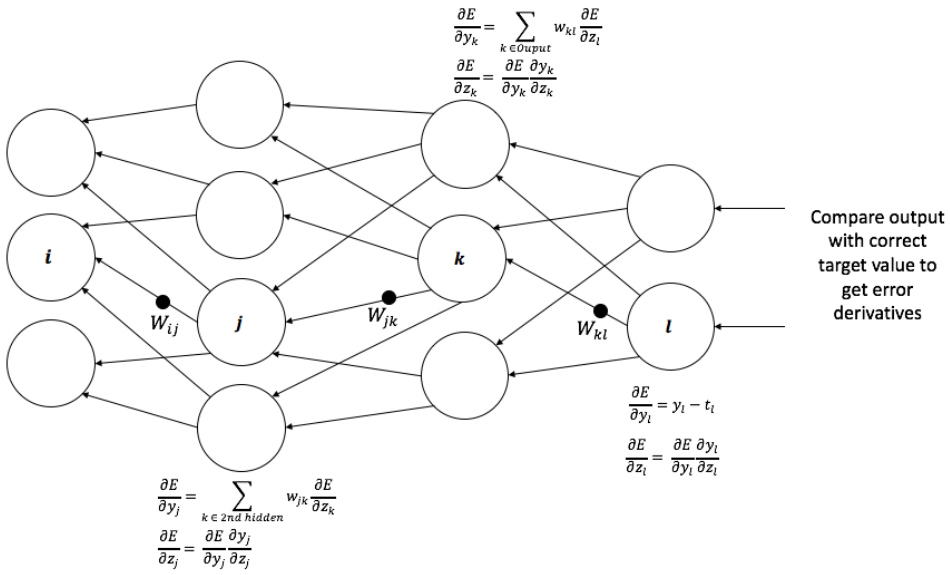
$$\hat{E}_n = E_n(w) + \eta P(w). \tag{2.15}$$

Here η denotes a balancing parameter and $P(w)$ is a function of weights in order to impact the form of the solution. In the L_2-regularization [21] approach the function P(w) is

$$P(w) = w^T w. \tag{2.16}$$



(a) *Feed forward phase*



(b) *Backpropagation stage*

Figure 2.4: *Feed forward pass and backpropagation of errors operations of the network. (a) shows the feed forward operation on a simple neural network with two hidden layers. (b) shows the backpropagation operation of errors and update process of the weights on the network. Redrawn from [20]*

This method is also called weight decay and the idea is to avoid larger weights from overfitting the network.

Another method for regularization is the "early stopping" method [29]. In this regularization method, as the overall error function of the network for the training set decreases over iterations, the validation error also decreases. The moment that the validation error starts to increase, overfitting is starting to occur and the basic concept of early stopping is to stop the network from training when the error of the validation set reaches a minima and starts to rise.

The "dropout" method attempts to generalize the operation of the network. Introduced first in [30], dropout skips the computation associated with every hidden node with a probability of 0.5 in any forward pass of the training process. This helps the network to be able to operate even in the absence of other processing blocks. This method is also called "model averaging" as it builds a model based on the average of multiple models. For the backpropagation step, the nodes which were ignored in the forward pass are also ignored in calculating the new weights in that iteration.

The "weight sharing" approach tries to harmonize nearby elements in the network. Introduced in [28] in the weight sharing approach different nodes in a layer use the same weights rather than computing their own. The reason that this method is counted under the class of regularization methods is because it reduces the complexity of the whole network. The difference between the weight sharing method and the traditional backpropogation algorithm is that the computation of backpropagation error used in traditional backpropagation in Equation 2.13 becomes

$$\frac{\partial E_n}{\partial w_{j,i}^{(l)}} = \sum_{k=1}^{m^{(l)}} \delta_k^{(l)} y_i^{(l-1)}. \tag{2.17}$$

with the assumption of all the nodes in layer l are using same weight values, i.e., $w_{j,i}^{(l)} = w_{k,i}^{(l)}$ for $k \leq m^{(l)}, 1 \leq j$.

2.1.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are DNN that are specifically tailored to operate on data in which adjacent nodes have related meaning, as is the case for image data. Essentially CNN restrict the connections from one layer to the next to be based on local operations rather than on operations that take place over the entire image. Such an approach has been found to enhance performance on image data [2]. The following provides a brief introduction to CNN. For a more complete description see [21] which introduced the notion of CNN and [31] which provides a recent survey of the field.

Convolution operation. Let us assume that the input to a given layer in a NN is a greyscale image scaled to the range $[-1, 1]$ defined as

$$I : \{1, 2, ..., n_1\} \times \{1, 2, ..., n_2\} \to W \subseteq \mathbb{R}, (i, j) \mapsto I_{i,j}. \tag{2.18}$$

Here image I is presented as a matrix with size of $n_1 \times n_2$. Defining $K \in \mathbb{R}^{2h_1+1 \times 2h_2+1}$ as a filter, then, the discrete convolution of image I and filter K is given by:

$$(I * K)_{r,s} := \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} K_{u,v} I_{r+u,s+v} \tag{2.19}$$

where the indices of filter K are defined as

$$K = \begin{pmatrix} K_{-h_1,-h_2} & \cdots & K_{-h_1,h_2} \\ \vdots & K_{0,0} & \vdots \\ K_{h_1,-h_2} & \cdots & K_{h_1,h_2} \end{pmatrix}. \tag{2.20}$$

Note that in order to do the convolution on the boarders of the image, some consideration is required. For example, if filter K is applied on the pixel of first row and first colomn of I, Equation 2.19 requires access to pixels beyond the edge of the image which do not exist. Various solutions to this issue have been proposed in the literature. One of the methods is to pad images (i.e., add a constant value padding around the image) to be able to do the convolution on boarder pixels) or to do the convolution only on the pixels which the

Equation 2.19 is valid. In the latter approach, the output image of the convolution will be slightly smaller than the original input image.

Layers. As with DNN, CNN consist of a number of different layers that feed into each other. Whereas DNN typically have only one kind of layer architecture – a fully connected layer – CNN are typically constructed of layers with different computational structures. A number of different types of layers which have been used in CNN architectures are briefly described below.

Convolutional layer. A convolutional layer l accepts $m_1^{(l-1)}$ feature maps with sizes of $m_2^{(l-1)} \times m_3^{(l-1)}$ from the previous layer. Therefore, the i^{th} feature map $Y_i^{(l)}$ in layer l is achieved as

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_l^{(l-1)}} K_{i,j}^{(l)} * Y_j^{(l-1)} \tag{2.21}$$

where $B_i^{(l)}$ is a bias matrix for layer l. $K_{i,j}^{(l)}$ is the convolutional filter which connects the j^{th} feature map in layer $(l-1)$ to the i^{th} feature map in layer l. These convolutional networks cover the input image or the feature map from the previous layer. They can sweep the matrix and move with one pixel per operation or skip pixels in the previous layer. The size of this skipping is called the "stride". The nature of the stride is an important factor which can decrease the size of the output of convolution operation with respect to the size of the input matrix. Figure 2.5 illustrates differences between a convolution operation on an input image with stride sizes of 1 and 2.

Rectification. As discussed previously section, Rectified Linear Units (ReLUs) have shown promising enhancement in the performance of CNN architectures [32]. ReLU layers do not change the size or depth of the feature maps and can be written as

$$Y_i^{(l)} = max(0, Y_i^{(l)}). \tag{2.22}$$

Rectification layers add a non-linearity to the network that can help the network to learn more complex models [33].

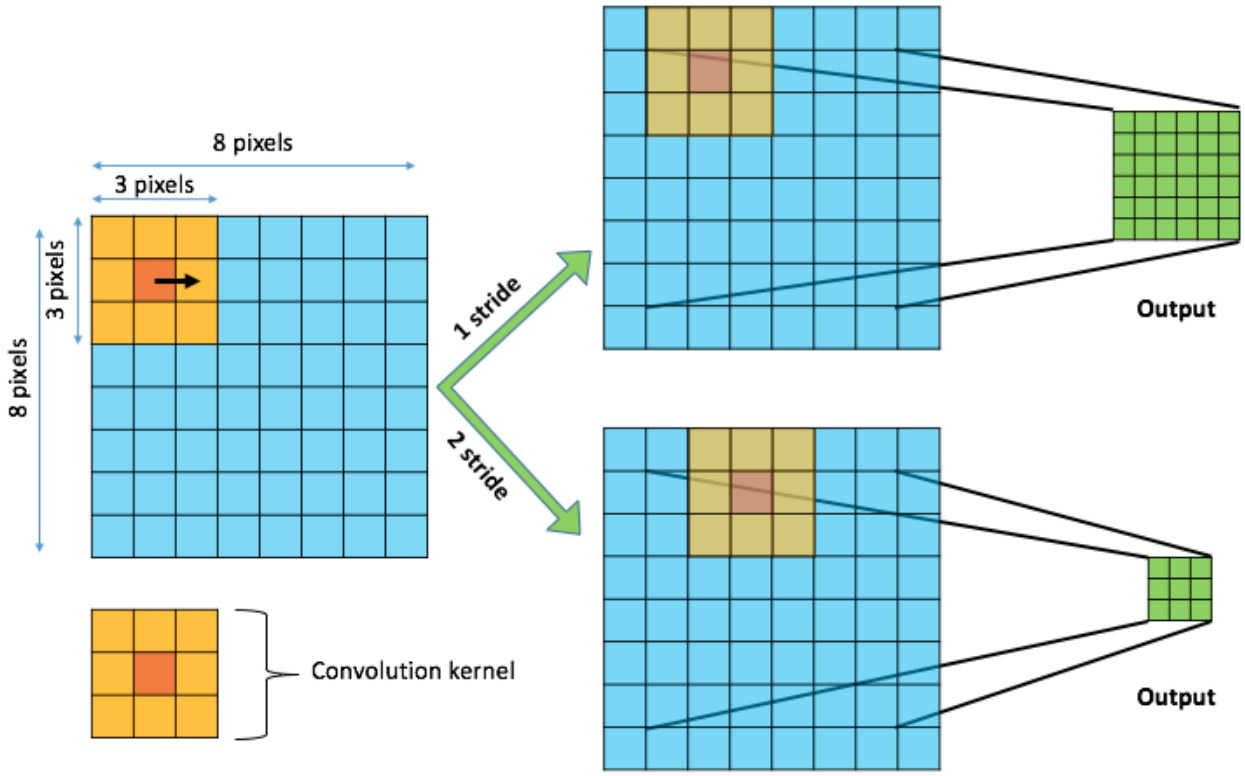


Figure 2.5: *Stride size effect on convolution operation on images. The blue matrix is an arbitrary input image with the size of* 8×8 *and a* 3×3 *convolution kernel is used to compute the next layer of the network. With stride size of 1, the convolution is computed on more number of pixels than bigger stride sizes. Therefore, the size of output of output in stride size 1 is bigger than the output of stride size of 2.*

Pooling. Pooling and subsampling layers are used in order to make the network robust to noise and invariant to the distortions present in the input data [32]. Pooling and subsampling layers reduce the resolution of their input and usually return an output with smaller dimension size than their inputs. These layers subsample the preceding feature map with non-overlapping kernels and return either the average of the values of the elements of feature map that the kernel operates on (Average Pooling) or return the maximum of all those values (Max Pooling). [34] reports that the max pooling approach helps to achieve faster convergence in the training process of the network. Figure 2.6 depicts both pooling methods.

Fully connected layers. These layers are typically found at the end of the chain of layers in a CNN.

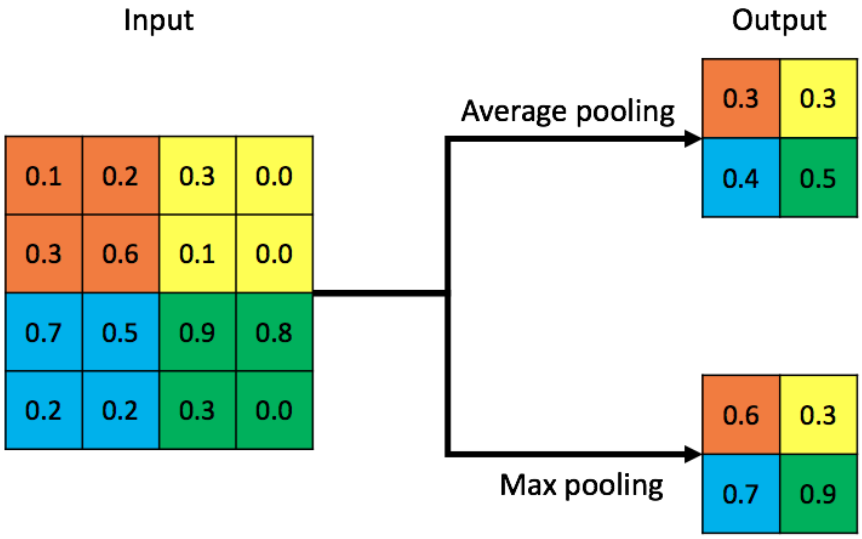


Figure 2.6: *Average and Max pooling. A* 2×2 *pooling kernel with stride size of* 2 *is applied on a* 4×4 *input matrix in both ways: 1) Average pooling and 2) Max pooling.*

Assuming layer l to be a fully connected layer and the layer $(l-1)$ is also a fully connected layer, then Equation 2.2 can be applied to them, but if not, the i^{th} node in layer computes

$$y_i^{(l)} = f(z_i^{(l)}) \textbf{ with } z_i^{(l)} = \sum_{j=1}^{m_1^{(l-1)}} \sum_{r=1}^{m_2^{(l-1)}} \sum_{s=1}^{m_3^{(l-1)}} w_{i,j,r,s}^{(l)} (Y_j^{(l-1)})_{r,s} \tag{2.23}$$

where $w_{i,j,r,s}^{(l)}$ is the weight that connects the node at position (r,s) of the j^{th} feature map in the layer $(l-1)$ to the i^{th} node in layer l. These fully connected layers are typically used for classification purposes in CNNs after a sequence of convolution, rectification and pooling layers. Fully connected layers publish the classification results of the network as a k-dimensional array output, where k is the number of classes and all the elements of the output array is in the range of $[0,1]$ and they add up to 1. This is done using the "softmax" function. The softmax layer takes as input a k-dimensional array with arbitrary values and converts this into a new k-dimensional array where each element is in the range of $[0,1]$ and the elements sum to 1. The softmax function is computed as

$$\textbf{softmax}(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}} \textbf{ for } i = 1, 2, ..., K \tag{2.24}$$

where z is the output of the fully connected layer before softmax function, e is the base of the natural

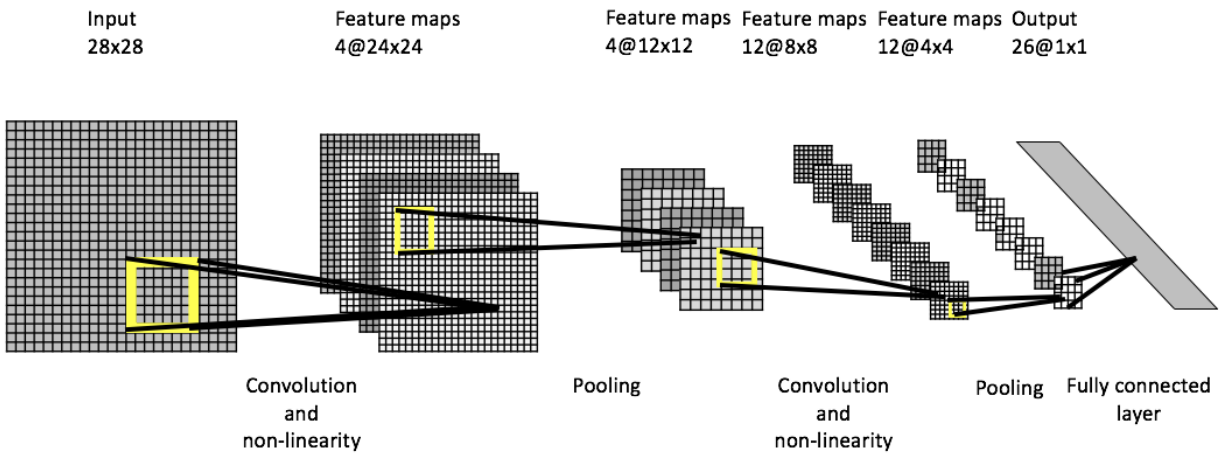


Figure 2.7: *The original convolutional neural network architecture introduced in [2]. Input image matrix, feature maps and the output of the network are shown. Each feature map is shown with the (number of maps)@ size* $\times$ *size. The output layer is a* 26 *element array which returns the probability of every English character.*

logarithm and i denotes the index of each element in z array.

2.1.4 Network architectures

A collection of layers can be put together in a number of different ways and the term 'architecture' is used to describe how the layers are connected. Figure 2.7 shows the architecture of the original CNN architecture introduced in [2] for a hand-writing recognition task. Figure 2.7 shows that the input of this network architecture is a grayscale image with a size of 28×28 pixels and how the convolutional, non-linear functions and pooling layers are applied to the input. At the end of the network a fully connected layer using a softmax function returns the probability of each class of data. There are many different network architectures in the literature. See [17], [35] and [36] for examples.

2.2 Transfer learning

Transferring knowledge from a trained network on a given task to another network operating on a target task is called transfer learning and a survey of different approaches in this domain is presented in [37]. The basic approach involves taking a pre-learned CNN and then surrounding this CNN with a small number of layers and then training just these layers on the specific problem. For example, a transfer learning approach is presented in [38] which uses a pre-trained model on the source task of ImageNet dataset classification challenge [39] for the Pascal VOC dataset classification challenge [40]. In this approach the last layer of the network (also known as classification layer) which is a fully connected layer, is replaced with two new fully connected layers and only the parameters of these two layers are trained on the Pascal VOC dataset. By retaining the weight and bias values of prior layers of the CNN pre-trained on ImageNet dataset, and retraining the new network with Pascal VOC dataset, a higher accuracy was achieved in comparison with the winner of Pascal VOC 2012 challenge [41]. Transfer learning transfers the mid-range features learned by the source network and exploits them in a new classification task. Obtaining a rich and vast annotated dataset on a new task is a very tedious and sometimes very expensive procedure. As a consequence, transfer learning can be very helpful when the target task does not have a huge dataset from which to train a CNN from scratch.

2.3 Road and trail following

Road following has been a central challenge for mobile robots since the beginning of the field. Using on-board sensors to find a particular pattern in the environment and then following that pattern by generating appropriate motion commands to the vehicle is an enabling task. For indoor robots this may involve following walls or lines on the floor. For outdoor robots a key application involves following road markings. Many different ways to achieve road have been developed in the literature. Visual sensors (cameras) and laser

range finders play a significant roll in this field. Although it may seem simple and straightforward to identify and follow a 'road' in a scene, this problem becomes difficult under adverse weather and lighting conditions. The 'simple' problem becomes even more difficult when 'roads' are replaced with 'pathways' or 'walkways' (aka off-roads and trails) as they are not as well structured as the streets and highways normally found in the urban environments. A range of different algorithms have been developed to enable robots to follow road-like structures. The majority of these approaches can be categorized into three main groups concerning the sensor that they use for the sake of path detection:

- Cameras (vision-based approaches),
- Laser (LIDAR-based approaches),
- Fusion of vision and LIDAR based approaches.

The following subsections briefly review each of these approaches.

2.3.1 Vision-based approaches

From almost the beginning of research into autonomous robots, cameras have been used to develop a model of the surrounding environment. Hence there is a huge literature concerning the use of vision to guide autonomous robots. Road following approaches can be traced back to the 1980's (e.g., [42],[43] and [44]), and research in this field continues today. It is not practical to review all approaches here (see [45], [46] and [47] for reviews), but rather a few examples are used to illustrate the vast number of different approaches. Vision-based approaches can be classified into two main groups: 1) classical image processing and computer vision approaches and 2) deep learning methods. A brief review of both approaches follows.

Classical image-processing based approaches. Within the set of vision-based approaches one can distinguish between 'classical image processing' approaches and approaches that utilize learning methods

to follow the roads. Approaches in the former class utilize classical image processing techniques (e.g., edge finding) to identify features in the image that correspond to the roadway, and then group these features into higher level primitives. Examples of algorithms that approach the problem in this way include [48] and [49].

Research continues in terms of classical image processing approaches to road following to the present day. Algorithms utilize edge following, color, stereo vision and other techniques to find the road in the scene. There is a large range of potential approaches. To highlight two examples, in [50] Moghadam et al. describes an approach for road direction estimation based on detecting the vanishing point of the road. In the first stage of this method, it localizes the vanishing point based on one frame of image and then in the second stage it uses a sequence of images to predict the direction of the given road and has used the dataset of the DARPA Grand Challenge for method evaluation. The algorithm detects the skyline and eliminates the unnecessary information from image and then uses gradient of the image to estimate the orientation of the upcoming road. A second real time approach using monocular vision is presented in [51]. This method segments camera image frames using a graph-based segmentation. Using a selected region of interest (ROI) in front of the ego vehicle (assuming robot is already on the road), a mixture model of the pixels in the ROI is developed and this is compared to the other segments. If a segment with known mean and covariance of pixels in HSV color space has a Mahalonobis distance to the mixture model of the ROI less than a specified threshold, then that segment is labeled as road. After connecting all the road segments, road boundaries for the navigation and calculation of the steering commands are extracted.

Classical learning-based approaches. Learning-based approaches seek to learn the features that correspond to the roadway rather than being hand-tuned to them. To take one example from these early learning-based approaches, [43] presents an adaptive vision system for autonomous navigation. It uses a color transformation method to maximize the differences of colors associated with the roadway and background and then computes the maximum likelihood for pixel classification task. This adaptive technique works by:

1. Updating the parameters of color transformation infrequently for intense color changes.

2. Updating the parameters of a maximum likelihood classifier to deal with the gradual changes in color and lighting intensity.

In [52], Lieb et al. presents a self supervised reverse optical flow technique for road following task using 1-D template matching. The assumption here is that the vehicle is on the road (it will also extract the road properties) as it moves along the road teleoperated by a human supervisor. As another example, in [15], Moghadam et al. presents a self-supervised learning algorithm for terrain classification that uses near-field stereo vision in front of the robot and combines with information about terrain features to train a classifier for classification of roads beyond the range of stereo vision.

Deep learning based approaches

In the last few years, approaches based on Deep Neural Networks (DNN) and specifically Convolutional Neural Networks (CNN) [2] have shown good performance on a range of different image-processing based tasks. In the scope of this thesis we review some of the efforts involving CNN and DNN-based approaches in the task of autonomous driving and road detection and following. In terms of supervised learning, two sets of approaches have emerged in the DNN literature to tackle autonomous driving problem: 1) road surface detection and classification followed by the generation of an appropriate steering angle 2) steering the robot on the road directly from training data. Both of these approaches are discussed below.

Road surface detection. This category of approaches segments image pixels into *road* or *non-road* classes. This information is then used to estimate the best steering angle to keep the robot on the road. These approaches treat the problem as a classification problem (which pixels are road pixels) exploiting the wide range of classification approaches that are based on CNN. For example, in [7], Mohan presents a deep deconvolutional network architecture that incorporates spatial information around each pixel in the labeling

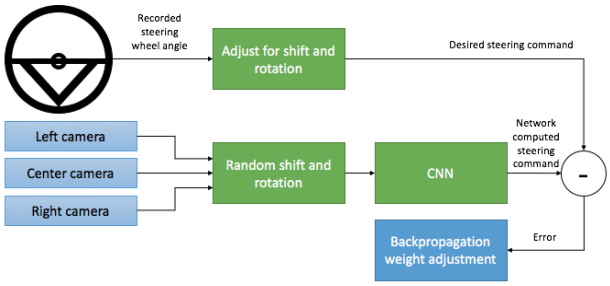


Figure 2.8: *NVIDIA deep learning method for finding the proper steering angle in each frame, redrawn from [19].*

process. [8] and [9] use small image patches from each frame in order to label the center pixel of each patch as road or non-road. These patches are fed into a trained CNN to classify the label of the center pixel of that patch. These algorithms use spatial information associated with each pixel in order to label that pixel with high confidence. In [10] Oliviera et al. present a method for road segmentation with the aim of reaching a better trade-off between accuracy and speed. They introduce a CNN with deep deconvolutional layers that improves the performance of the network. Another advantage of this method is that it uses the entire image as an input; not different patches of each frame and this really helps to make the algorithm to run faster and be more efficient.

Steering the robot directly. In [13] Pomerleau presented an Autonomous Land Vehicle in a Neural Network (ALVINN), using a fully connected network. This approach uses a set of driver's actions captured different roads such as one lane, multi-lane paved roads and also unpaved off-roads, to train the classical neural network architecture of the robot to return a suitable steering command to the robot. Inspired by this work, NVIDIA corporation designed a system to train a CNN on camera frames with respect to the given steering angle of a human driver for each frame [19]. An instrumented car is outfitted with a camera that simulates the human driver's view of the road, and a human drives the vehicle while the camera input and human steering commands are collected. A CNN is then constructed from this dataset using the human steering angle as ground truth. A simplified block diagram of this approach is shown in Figure 2.8. In [19],

Bojarski et al., present an end-to-end learning method for driving a car in public roadways with or without lane marking as well as highways. They claim that using an end-to-end learning approach, system will optimize all the parameters altogether and there is no need for hand tuning any other hyper-parameter for different road types and conditions.

A common problem with the simple model of using a human to train a DNN is that the network only experiences 'nominal' driving conditions. Humans are generally good drivers and thus the network only sees 'good' driving conditions. In order to train the network for less normal conditions it is necessary to include in the training dataset examples that are not 'normal'. The NVIDIA system addresses this in two ways. First, as shown in Figure 2.8, there are three cameras assembled on the vehicle. One of them faces forward and the other two are angled to the right and left of the road. These cameras capture conditions when the driver would be twisted left or right relative to the normal driving directions. Second, the three camera frames are fed into a random shift and rotation unit. This allows the system to be trained on images that might be acquired if the car is perturbed from normal driving conditions. Output of this augmentation then is used to train a CNN using backpropagation using the human steering command as the correct answer. A second key issue in the development of a neural network to provide driving commands is how to encode both the user's responses and the output of the neural network. NVIDIA uses the inverse of the radius of the instantaneous steering circle to encode this information. This allows straight ahead (instantaneous radius of infinity) to be easily encoded for the neural network.

In the development of a CNN for a given task a key question becomes the nature of the structure (the architecture) of the CNN. For NVIDIA's system the CNN is a 9 layer network with 5 convolutional layers and 3 fully-connected layers. Stride sizes in the first three convolutional layers are 2×2 with a kernel size of 5×5 and non-strided convolutional layers in the remaining two layers. This network consists of about $250,000$ parameters.

CNN have also been used to drive a robot on off-road trails as well and one such approach is presented

in [18]. This work uses a machine learning approach for following a forest trail. Rather than directly mapping image to steering angle this approach categorizes the input image into one of 'straight', 'left' or 'right' and then uses the distribution of likelihoods over these three categories to compute both a steering angle as well as an appropriate vehicle speed. In order to collect training data three cameras are mounted on a head of a hiker, one pointing 'forward' and one yawed to the right and the other one yawed to the left. Data is collected while the 'forward' camera is aligned with the trail. These three cameras are set up with a 30 degrees yaw from each other on the head of a hiker to record the dataset. Their dataset consists of 8 hours of video in forest-like trails and images are captured in a way that hiker always looks towards the direction of the trail. Therefor, in a classification task, the central camera is labeled as "go straight", the left camera is labeled as "turn right" and the right camera is labeled as "turn left". These labels are then used to train a CNN in order to do the classification task and outputs a probability of each class as a softmax function. They used a 9 layer neural network in order to do the classification task. These layers consist of 4 back to back convolutional and max-pooling layers followed by a 2000 neuron fully connected layer and finally a classification layer (output layer) with three neurons which returns the probability of occurrence of each label. This DNN is based on the architecture used in [53]. For evaluation purposes, the accuracy of this classifier is calculated based on the maximum probability of softmax function in the output of DNN. The reported accuracy is 85.2% for the classification task between three labels of "go straight", "turn right" and "turn left", but they did not publish any criteria of how well it performs to steer a robot on a real trail.

2.3.2 LIDAR-based approaches

LIDAR-based methods for road following have been studied widely in the literature as well. The main advantages of using LIDAR sensors are that they provide a 3D representation of the environment and operate independently of the lighting conditions. The major disadvantages of LIDAR sensors are their lack of dense resolution, their need to emit energy into the environment and their relatively high price in comparison with visual sensors. Most of the LIDAR sensors are able to return the intensity of the points as well which can

be exploited when detecting different surface features independently from the lightning (see [54], [55] and [56]). As an example, in [55], Kammel use LIDAR intensity values to detect the lane markings due to their cue difference from the background (i.e., road) and to extract the curb's position on the road based on the height change in the laser range finder data. [57] and [58] used LIDAR to detect the curbs and consequently the edges of the road ahead. Single beam LIDAR sensors are only able to detect obstacles and features in the plane of the laser beam. Multiple 2D LIDAR sensors or 3D laser scanners, detect more features of the environment. An improved road detection and following scheme (see [55] and [5], for example) can be achieved with multiple LIDAR scanners in comparison to the single 2D LIDAR approach (see [54] and [59], for example).

2.3.3 Integrated vision-laser-based approaches

Fusing both visual information and laser range information enables a robot to have a more robust perception of the surrounding environment and the trail within it. Rasmussen [60] describes an approach with a laser range finder and vision for the road following task, using four different features; height and smoothness of the points in front of the robot and color and texture features obtained from an on-board camera. Rasmussen used a neural network to classify different kinds of roads (paved or unpaved) using data sets of images and laser information from similar road scenarios. In the DARPA Grand Challenge in 2005 [3], the Stanley robot, which won the competition, used a combination of vision and laser range finder for task of road detection in off-roads and desert terrains [5]. Equipped with five 2D laser range finders, Stanley was able to classify the terrain to three different classes; obstacle, drivable and unknown. An obstacle is defined on a grid cell around the robot which two nearby points of data have a height difference of more than a defined vertical distance threshold. Using this map of the road, Stanley computed a quadrilateral in front of the robot containing all possible drivable grid cells and used this region to perform color classification of the further road ahead to increase the range of road detection. Stanley could maintain high speeds in navigation and was able to handle the sudden changes in the surface type of the road as well. Classic approaches in integrating vision

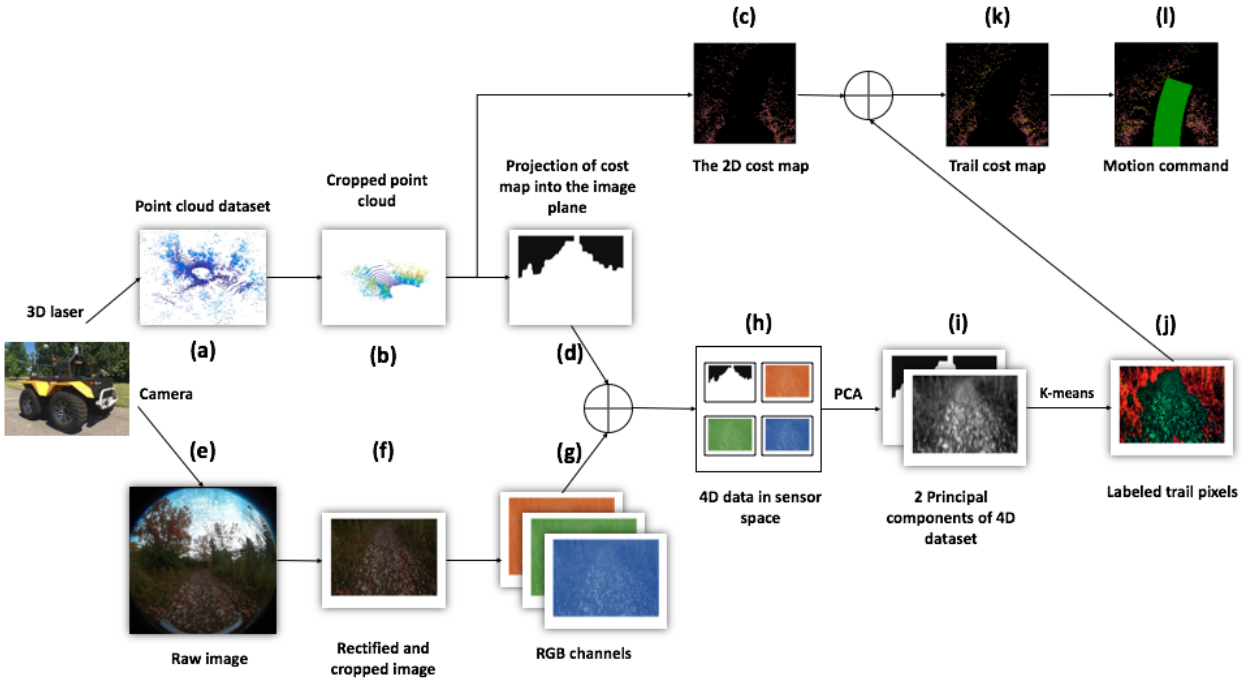


Figure 2.9: *The flow diagram of the trail finding algorithm. Data from the two sensors are processed into a common reference frame. Laser data is re-coded as error relative to the expected ground plane. When integrated with the camera signal this produces a 4D sensor signal (r,g,b,e) which then goes through a PCA process to obtain a 2D signal for segmentation. The k-means clustering algorithm is used to segment regions similar to the region directly in front of the robot (which is assumed to be trail) from other regions.*

and laser information have been very successful along with new efforts on using DNN and CNN to enhance off-road autonomous driving problem. There are a large number of details that must be addressed in the development of an integrated LIDAR-vision trail following algorithm. Here we provide a detailed description of an integrated vision-laser-based approach to highlight the details involved in trail following using multiple sensors.

Hoveidar-Sefid and Jenkin in [61] presented a trail following algorithm using the fusion of 3D laser information and visual data from a front facing omnidirectional camera. For trail following, they assumed that the robot is on the trail at time t, and that the goal is to obtain the local twist vector (V, ω) that moves

the robot along the trail while centering the robot on it. so that the robot has moved along the trail and is (more) centered on it at time t+1. The basic approach in this trail following algorithm is to construct a hyper-dimensional image that characterizes both deviation from the ground plane and per pixel image information, and then assuming that the robot is currently on the trail to utilize the K-means clustering algorithm on a reduced dimensional version of this image to identify trail versus non-trail locations. This information is then used to construct a local motion command (twist vector) that will drive the robot along the trail and keep the robot centered on it (see Figure 2.9).

2.3.4 Systematic evaluation and datasets

Given the commercial importance of autonomous driving and the large number of researchers working on the problem, a number of large-scale evaluations and datasets now exist to compare and benchmark road following algorithms. There have been a large number of effective road following algorithms developed over the years and large-scale systems and their evaluations have been held. The DARPA Grand Challenge Events [3], for example, provided the opportunity for (then) state of the art algorithms to be tested in a common testing environment, and details on many of the algorithms used and their performance can be found in [47]. The KITTI dataset [62], a standard dataset for road following evaluation, based on the benchmark presented in [63] is a modern dataset in order to be used as an evaluation for autonomous road following tasks and it has been widely used in the recent efforts in this field and is explained in full detail in [62].

Evaluating a DNN that is used for driving is a subtle problem. One option would be to use the mean or max deviation in steering angle of some test evaluation dataset, as this is what the CNN was trained to minimize. A more pragmatic error metric is to score the CNN on how well it drives the car, rather than how accurately it mimics human performance. The NVIDIA project [19] chose to use the later performance metric, and this is a commonly used metric in the literature. So for evaluation purposes, accuracy of this algorithm is calculated as a fraction of autonomous operation of the vehicle over total time of driving,

including interventions of a human driver. Therefor $\textbf{Accuracy} = \frac{\textbf{Time of autonomous mode}}{\textbf{Total time of driving}} \times 100\%$. The NVIDIA project in a regular two lane roads in a short trip in New Jersey, reached an accuracy of 98% [19].

2.4 OpenStreetMaps for trail type annotation

Topographical maps are available in printed form and at different scales and accuracy for different regions of the world. Updating printed paper maps is a time consuming and error prone process, and thus prior to the development of modern computer technology maps were routinely out of date. Although it is possible to extract digital information from printed topographical maps, it is also possible to obtain digital versions of topographical maps directly. These maps provide an accurate representation of the environment and points of interest. Today, large numbers of maps are available for public access and are being used in applications including public transportation, traffic control and travel planning. "Google Maps" [64], "Bing Maps" [65],"Yahoo Maps" [66],"OpenStreetMap Project (OSM)" [67],"MapQuest" [68] and "Apple Maps" [69] are among the most common publicly available digital map sources. Some of these map providers only allow the user to utilize a number of specific services and do not publish their map information for free. The OpenStreetMap foundation [67] is an open source mapping community which allows individuals to contribute to the map representation and provides the resulting maps for free. The accuracy of these maps is not well determined, as every mapper may use different tools or provide a different precision during the mapping process. As is the case with many crowd-sourced dataset, inaccuracies can be easily introduced by the contributors. That being said, OpenStreetMaps cover most of the world and gross errors are easily found and corrected by contributors.

Topographical maps of the environment have been exploited for autonomous navigation purposes. A GIS (Geographic Information Systems) human readable map has been used for single or multi-robot crews to function in an urban environment by translating required information of the given map to a 3D model of existing obstacles in robots nearby [70]. In [71] a method that combined OSM data and visual odometry

for global vehicle localization was proposed for robot navigation. The approach described in [72] annotates a given street map manually with three different layers, sidewalks, roadways and buildings and uses the augmented for robot navigation with a multi layer laser range finder, however building these map representations is time consuming and is not practical in larger environments. In [73] Tao et al. present an approach for navigation using digital maps improved by the robot. This approach exploits an available digital map of the environment. In order to increase the accuracy of the map, the robot learns the lane markings and the exact shape of the pathways of interest by traversing the region with a precise localization system and a human operator. After this learning process, low-cost GPS and other sensors on the vehicle are used for localization. Google Maps have been used for autonomous robot navigation as well. Senlet in [74] presents a framework for global localization of a robot using Google Maps, in the presence of real world moving objects in the environment, without enhancing global positioning systems. OpenStreetMap also has been used for urban navigation in different aspects (see [75], [76] and [77] for examples) and for an assisting method for point to point navigation of people with mobility impairments [78]. Brubaker et al. recently exploited OpenStreetMap information and visual odometry data for self-localization of cars in urban environment as two core inputs, using probabilistic methods [79].

2.5 Summary

Road, trail and path following are enabling technologies for autonomous vehicles. A large literature exists for this problem but in the main research has addressed the problem of well marked "road" following rather than the task of following off-road dirt tracks and trails although there have been modest attempts in this domain as well. Solutions can be divided based on the nature of the sensors that are used to identify the roadway in front of the vehicle and the way in which this information is used to control the vehicle.

In terms of visual sensors, a large literature exists on the use of traditional image processing techniques as well as neural-network based approaches. The last few years have shown the development of effective

NN-based techniques, at least for well marked roadways. Can CNN-based techniques be applied to off-road tracks and trails as well? And if so, what is the appropriate way to capture trail data to train such a CNN and to train the CNN itself? These and related questions are considered in the following chapters.

Chapter 3

Convolutional Neural Networks for trail following

3.1 Introduction

A key requirement for off-road autonomous navigation is to be able to follow a certain path or trail in a natural and unstructured environment. The robot is continuously following a trail and at each instant takes a sensor snapshot of the trail "in front" of the robot. The goal is to take this snapshot and to detect the deviation of the robot with respect to the direction of the trail and generate an appropriate motion command that moves the robot along the trail. Trail following in general is a hard problem in comparison with the well known autonomous navigation in urban environments as the urban road fallowing problem is aided by many detectable and well known features available in highways and streets. This is to be compared to the somewhat unstructured nature of trails in rural environments (Figure 3.1). The goal of the trail following algorithm is to detect where to move next in order to follow the trail. The diversity of trail types presents many challenges for the task of fully autonomous navigation. Other factors including uncontrolled lightning, weather conditions and seasonal variations also add to the complexity of the task. The goal of the

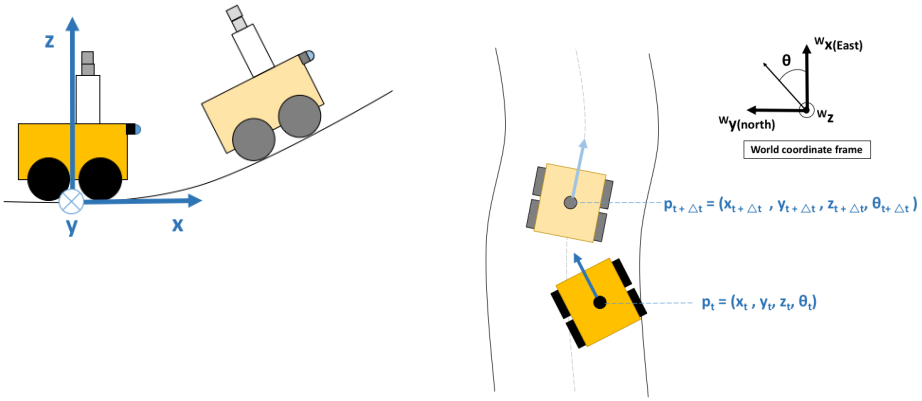
(a) *(b)*

Figure 3.1: *Different roadway types. (a) shows a primary road in urban environment which has a consistent surface color and detectable lane markers and road edges. (b) shows an off-road dirt path covered with leaves which lacks a detectable edge and consistent surface type and color.*

trail following algorithm is to detect the deviation of the heading direction of the robot with respect to the direction of the trail and publish an appropriate motion command to keep the robot on the trail and move it forward.

Although there are many possible approaches to trail following, this work utilizes a CNN in order to compute the appropriate trail following motion command. In particular, this work is inspired by the CNN-based approach presented in [18] and seeks to address the following questions which were first identified in Chapter 1:

1. Is it possible to increase the classification accuracy presented in [18] using "transfer learning" with a pre-trained model of another DNN problem?
2. How many cameras are required to make a robust dataset for training the CNN of the system in order to maximize the accuracy of the trail following algorithm?
3. Is there an advantage to using wide field of view cameras over narrow field of view cameras?
4. Is a specific trained CNN for all the trail types sufficient or is a separate CNN required for each type of trail?



(a) *(b)*

Figure 3.2: *Coordinate frames. (a) shows the robot coordinate frame placed underneath the center of mass of the robot on the ground plane in point $p = [0, 0, 0]$. (b) shows a bird's eye's view of the robot on the path in two subsequent frames at time t and $t + \Delta t$ with its position and orientation in the world coordinate frame. The deviation angle between robot headings in two frames is caused by the correction angle obtained from the motion command of the CNN classifier.*

3.2 Formal definition of the trail following problem

For trail following, it is assumed that the robot is on the trail, and the goal is to estimate the local twist vector (V, ω) that moves the robot along the trail while centering the robot on it. The state of the robot in the world coordinate frame is shown in Figure 3.2. In order to formally define the problem these assumptions are defined as follows:

- The robot is on the trail at all times (p_t is on trail).
- The robot is more or less centered on the trail and looking along the trail ($||\theta_t - \theta_{trail}|| < \tau_{max}$. In the work here $\tau_{max} = 60°$).

- The drivable path on the trail is more or less flat.
- The trail has certain features which differentiate it from the surroundings.

The robot uses visual information from the local environment and feeds it to a trained CNN in order to classify the deviation of the heading of the robot with respect to the trail ahead, and based on the estimated angle generates a motion command to move the robot in the direction of the trail.

3.3 Transfer learning using the Inception-V3 network

Given the success of transfer learning in a range of other applications, in this work transfer learning is used to train the CNN. More specifically here we use "Inception-V3" [80], a well-known DNN architecture for ImageNet Large Scale Visual Recognition Challenge (ILSVRC) challenge [81] which achieved state-of-the-art accuracy in the time of its publication, for the trail classification task.

In the ILSVRC challenge [81], GoogleNet [82] and VGGNet [36] (two of state-of-the-art architecture at their time) achieved very high performance in the image classification task. One advantage of GoogleNet is that it used a new architecture within its CNN, called an Inception module. The basic concept behind an inception module is that processing the visual information at different scales and then aggregating these results together to feed the next layer in CNN helps the network to learn abstract features at different scales [82]. Figure 3.3a shows a visualization of the Inception module. This module helped GoogleNet to reduce the number of parameters related to the competing networks in that time by a factor of 12. The number of exploited parameters in GoogleNet is about 5 million while the VGGNet consists of about 180 million parameters. The large number of parameters in VGGNet makes it unsuitable for real-time applications such as mobile robot applications, due to the very expensive computation required for each forward pass of the network. The basic concept behind GoogleNet is improved in its successors Inception-V2 [80],

Inception-V3 [80], Inception-V4 [83] and Inception-ResNet [83]. More details about these architectures are available in the references given above.

The Inception-V3 network proved very successful on the ILSVRC classification challenge obtaining top-1 accuracy of 78.0% and top-5 accuracy of 93.9% which is very close to the performance of later versions of Inception, i.e., Inception-V4 and Inception-ResNet which have (top-1, top-5) accuracy of $(80.2\%, 95.2\%)$ and $(80.4\%, 95.3\%)$ respectively. In the Inception-V3 network, the Inception module is revised in a way that helps to reduce the computational cost by using smaller convolutional kernels, with more layers. For example, The Inception-V3 network substituted a 5×5 convolutional kernel with two back to back 3×3 kernels which resulted in substantial improvement in performance and computational speed. Three different implementation of this revision are explained in Figure 3.3. Another advantage of the Inception-V3 over later versions of these networks is that Inception-V3 has a less complex network architecture in comparison to its two slightly better successors. One limitation of Inception-V3 is that Inception-V3 accepts input images of a fixed size of 299×299 pixels. Table 3.1 explains the details of Inception-V3 network architecture.

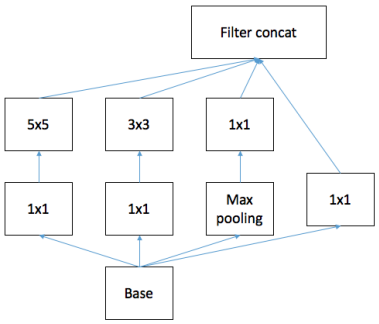
In order to retrain Inception-V3 network architecture for a new task using transfer learning methodology, in this work the number of the last two layers of the network as shown in Table 3.1 (i.e., the fully connected classification layer and the softmax layer) are replaced with with randomly initialized weights for the fully connected classification layer and a softmax layer with an output whose size is equal to the number of classification labels.

3.4 TrailNet dataset

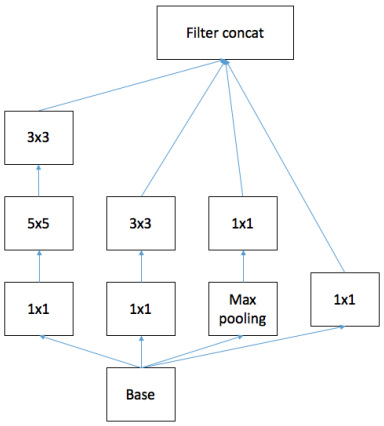
Training a CNN requires an appropriate labelled dataset. For this work we collected a dataset of different trails under various trail and imaging conditions. This dataset is called the TrailNet dataset and its capture is inspired by the approach presented in [18]. TrailNet consist of images captured from a wide field of view

Table 3.1: *Inception-V3 network architecture [1]. The network consists of 13 structural blocks. The input is a 299x299x3 channel image which is then processed by convolution layers (see Chapter 2) with* 3×3 *convolution mask with a stride size of 2 pixels in the first layer, two* 3×3 *convolution masks with a stride size of 1 pixels in the second and third layer,* 3×3 *max pooling mask with a stride size of 2 pixels in layer four, a* 3×3 *convolution mask with a stride size of 1 pixels in layer five, a* 3×3 *convolution mask with a stride size of 2 pixels in layer six, a* 3×3 *convolution mask with a stride size of 1 pixels in layer seven, three Inception modules as shown in Figure 3.3b, five Inception modules as shown in Figure 3.3c, two Inception modules as shown in Figure 3.3d, a* 8×8 *max pooling mask with a stride size of 1 pixels, a fully connected layer with the size of* $1 \times 1 \times 2048$ *and finally a softmax layer with* 1000 *outputs (as the number of class labels in ImageNet dataset).*

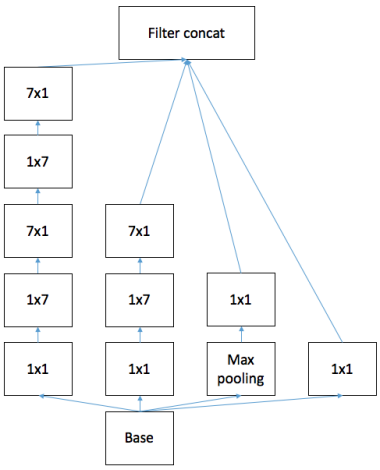
Type	**Patch size/stride or remarks**	**Input size**
conv	$3 \times 3/2$	$299 \times 299 \times 3$
conv	$3 \times 3/1$	$149 \times 149 \times 32$
conv	$3 \times 3/1$	$147 \times 147 \times 32$
pool	$3 \times 3/2$	$147 \times 147 \times 64$
conv	$3 \times 3/1$	$73 \times 73 \times 64$
conv	$3 \times 3/2$	$71 \times 71 \times 80$
conv	$3 \times 3/1$	$35 \times 35 \times 192$
3× Inception	As in Figure 3.3b	$35 \times 35 \times 288$
5× Inception	As in Figure 3.3c	$17 \times 17 \times 768$
2× Inception	As in Figure 3.3d	$8 \times 8 \times 1280$
pool	$8 \times 8/1$	$8 \times 8 \times 2048$
linear	fully-connected layer	$1 \times 1 \times 2048$
softmax	classifier	$1 \times 1 \times 1000$



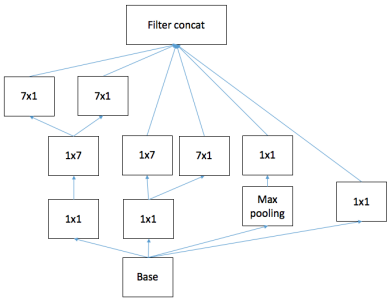
(a) Original inception module



(b) $2 \times 3 \times 3$ *kernels instead* 5×5 *kernels*



(c) 1×7 *and* 7×1 *kernels instead of* $n \times n$ *kernels*



(d) Revised version for capturing higher dimensional representations

Figure 3.3: *The original inception module and revised structures of the original Inception module used in the the Inception-V3 architecture [80]. (a) is the original inception module introduced in [82]. (b) shows that* 5×5 *convolutional kernel of subfigure (a) replaced with two back to back* 3×3 *kernels. (c) introduces a method to factorize convolutional layers with* $n \times n$ *size where* $n \in \{12, 13, ..., 20\}$ *with back to back* 1×7 *and* 7×1 *convolutional kernels. (d) shows a revised version of the original inception module for promoting higher dimensional representations.*

(a) *(b)*

Figure 3.4: *Wide field ('omnidirectional camera'). (a) Kodak Pixpro SP360 omnidirectional camera [84]. (b) Shows a sample image taken by the camera.*

cameras of different trail types, where each class of images has a certain deviation angle from the heading direction of the trail. TrailNet is available for public use at "http://vgrserver.eecs.yorku.ca".

3.4.1 Dataset capture

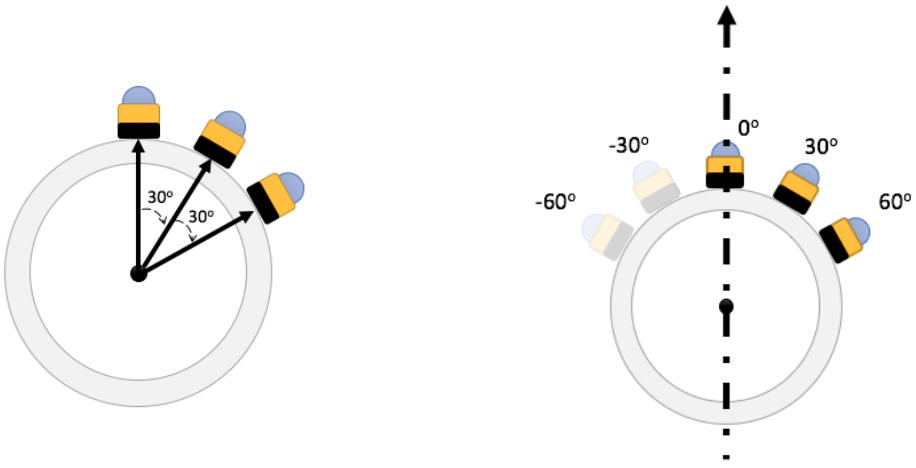
TrailNet was captured with three omnidirectional cameras. Each of the cameras is a Kodak Pixpro SP360 [84], which is a 360° degree fish eye omnidirectional camera with a 214° degree of vertical field of view with the video capturing ability of 30 frames per second with a resolution of 1072×1072 pixels (see Figure 3.4). This large field of view provides considerable flexibility when working with the dataset as it is then possible to use the whole image or crop it in order to simulate narrower field of view images.

Camera mounting setup. A Pioneer 3-AT robot [11] was used as a mobile base in this work. During training the three cameras were mounted on a detachable circular mount on top of the robot, while during trail following only the centre camera was used. The mounting setup on top of the Pioneer robot is shown

Figure 3.5: *Pioneer robot with the camera mounting setup on it. As shown here the cameras are separated from each other by* $30°$*. The two off-axis cameras can be mirrored to simulate the cameras on the opposite side of the robot.*

in Figure 3.5. Three omnidirectional cameras were used in order to record multiple frames of the trails; each with a different deviation angle from the trail heading. As a practical point of view to the problem of autonomous navigation on trails, the deviation of the robot on a trail is assumed to be in the range of $[-60°, 60°]$ with respect to the heading direction of trail. Trails are assumed to be approximately symmetric, i.e., if an image frame shows a $\theta°$ deviation from trail direction, then horizontally flipped version of the same frame indicates a $-\theta°$ deviation angle. This assumption allows us to "simulate" more camera frames with the three mounting cameras shown in Figure 3.6(a).

The camera orientation range of $[-60°, 60°]$ is divided into five different camera positions, with $30°$ difference between adjacent orientations. That is, the cameras are aligned at $-60°, -30°, 0°, 30°$and $60°$ relative to straight ahead. In order to record a dataset with these five different angles, using only three available cameras, a single piece of the trail needs to be captured once with the setup shown in Figure 3.6(a).



(a) *(b)*

Figure 3.6: *Camera mounting setup. (a) is a circular setup that holds three cameras with* 30° *difference between them and (b) shows the mirroring process of the main setup in order to capture all* 5 *angle classes for the TrailNet dataset. Note that the dotted-line arrow is pointing to the direction of the trail and transparent cameras with faded color are the simulated cameras obtained by flipping the images of their corresponding camera on the right side of the circle.*

3.4.2 Individual trail datasets

Within the TrailNet dataset there exists a number of different datasets that can either be used separately or combined together in order to train the network. Table 3.2 summarizes the complete collection of datasets within TrailNet. Here we identify specific subsets within the full dataset and highlight their purpose in evaluating the CNN in terms of its ability to perform trail recognition.

Camera geometry. These datasets represent different camera geometries in terms of training the CNN.

- **Five cameras**: all the 5 frames with 30° angle separation between adjacent cameras (i.e., $[-60°, -30°, 0°, 30°, 60°]$),

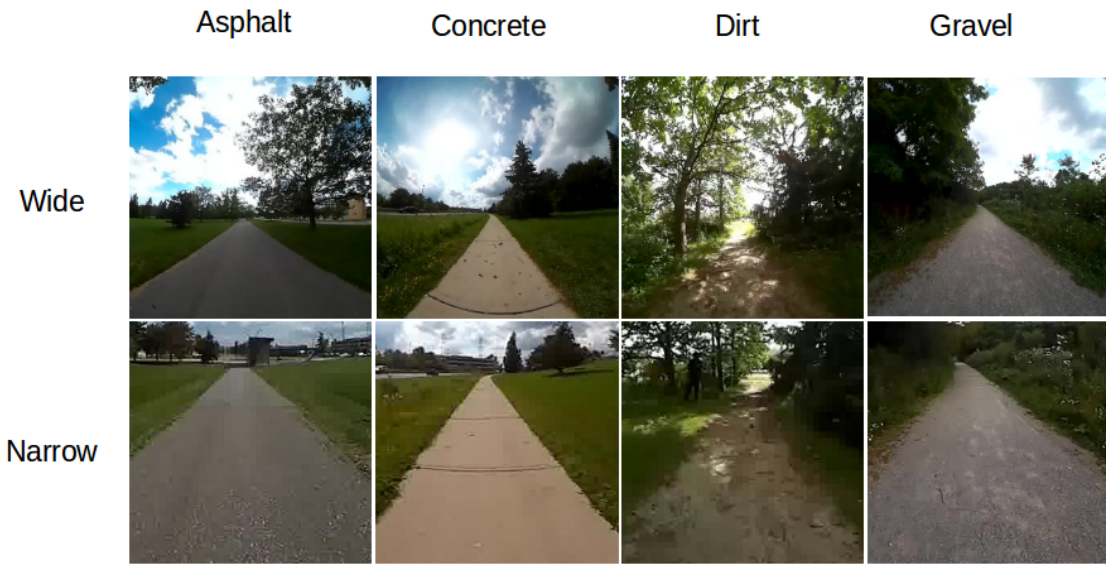


Figure 3.7: *Wide field of view versus narrow field of view. The first row contains examples of wide datasets on all trail types (from left: asphalt, concrete, dirt and gravel) and the second row shows images of the same trail types with a narrow field of view retina.*

- **Three cameras with** $60°$ **angle separation between adjacent cameras**: frames with $60°$ difference in between (i.e., $[-60°, 0°, 60°]$).

Figure 3.7 shows a single image tuple captured from each of the datasets in both their wide and narrow field settings.

Camera field of view. In order to analyze the effect of the field of view of the camera, each subset of the TrailNet dataset is further divided to two different versions. One version uses the entire image and is labeled "Wide". In order to build the "Wide" datasets, the central square in the original omnidirectional images of 740×740 is selected and resized to 299×299 pixels as this resolution corresponds to the size of input that Inception-V3 accepts. These frames have a field of view of $150°$. The other version is called "Narrow". In order to build the "Narrow" datasets, a rectangular region with the size of 299×299 pixels was selected from the lower-center of each image frame. The field of view of frames with this cropping methodology is

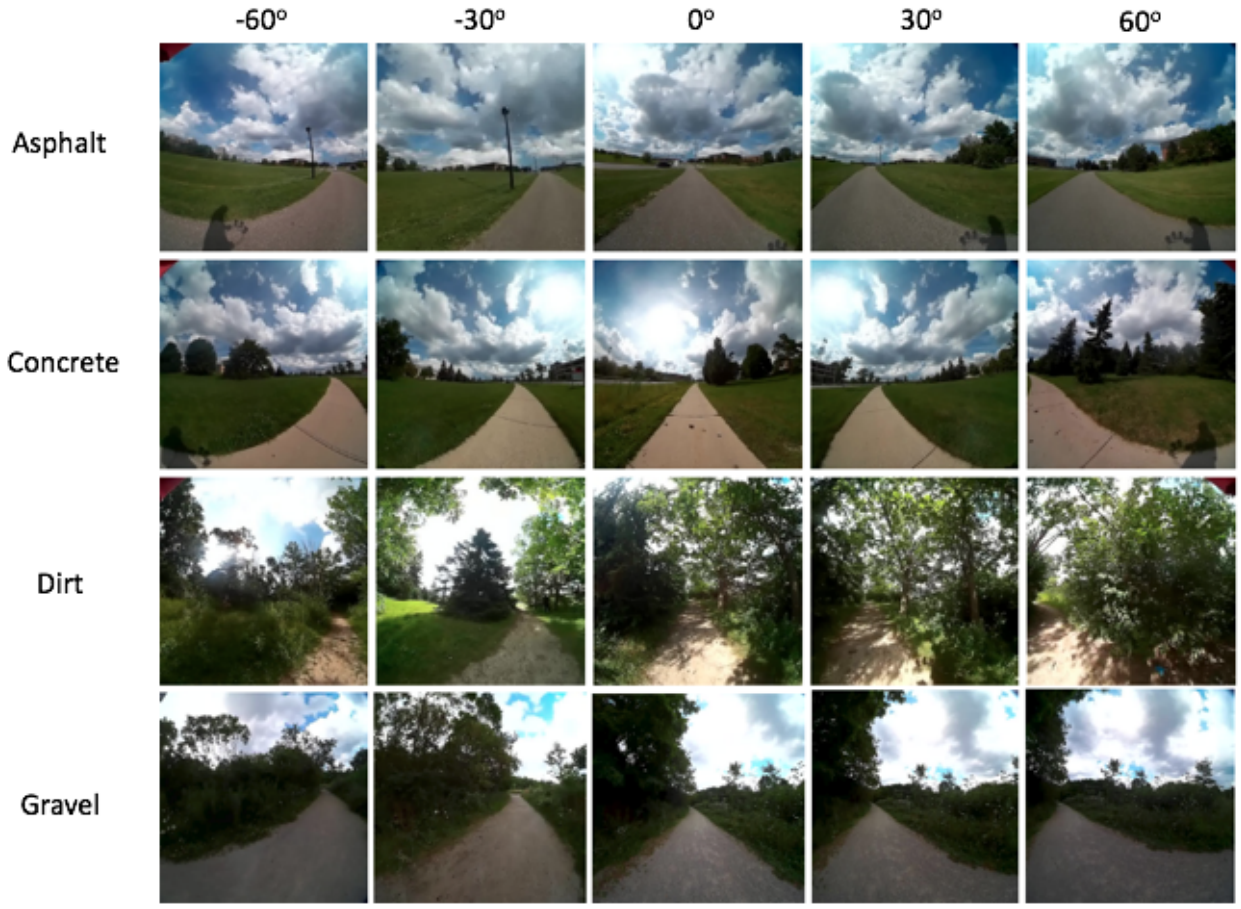


Figure 3.8: *A stack of the wide field of view with the five camera setup on all the trail types. First to the last row show asphalt, concrete, dirt and gravel trails respectively. Columns consist of the deviation from the heading of the trail in sorted as this vector:* $[-60°, -30°, 0°, 30°, 60°]$.

approximately 60° which corresponds to the nominal field of view of a regular camera.

Trail-type annotated dataset. In order to study the effect of surface type of the trails, each of the TrailNet datasets are further divided by trail types: (1) asphalt, (2) concrete (3) dirt and (4) gravel. Figure 3.8 shows a sample image snapshot for the five element wide dataset for each of the trail types.

All the aforementioned subset versions together constitute the full TrailNet dataset. A complete list of all TrailNet dataset elements is given in Table 3.2. The total number of images in the processes of

Table 3.2: *TrailNet dataset details and versions.*

Number	Wide/Narrow	# of cameras	Camera angles	Trail type	# of images
1	Wide	5	$[-60°, -30°, 0°, 30°, 60°]$	Asphalt	31250
2	Wide	3	$[-60°, 0°, 60°]$	Asphalt	18750
3	Narrow	5	$[-60°, -30°, 0°, 30°, 60°]$	Asphalt	31250
4	Narrow	3	$[-60°, 0°, 60°]$	Asphalt	18750
5	Wide	5	$[-60°, -30°, 0°, 30°, 60°]$	Concrete	31250
6	Wide	3	$[-60°, 0°, 60°]$	Concrete	18750
7	Narrow	5	$[-60°, -30°, 0°, 30°, 60°]$	Concrete	31250
8	Narrow	3	$[-60°, 0°, 60°]$	Concrete	18750
13	Wide	5	$[-60°, -30°, 0°, 30°, 60°]$	Dirt	31250
14	Wide	3	$[-60°, 0°, 60°]$	Dirt	18750
15	Narrow	5	$[-60°, -30°, 0°, 30°, 60°]$	Dirt	31250
16	Narrow	3	$[-60°, 0°, 60°]$	Dirt	18750
9	Wide	5	$[-60°, -30°, 0°, 30°, 60°]$	Gravel	31250
10	Wide	3	$[-60°, 0°, 60°]$	Gravel	18750
11	Narrow	5	$[-60°, -30°, 0°, 30°, 60°]$	Gravel	31250
12	Narrow	3	$[-60°, 0°, 60°]$	Gravel	18750

training, validation and testing, for each label is 6250. In order to be consistent throughout the training and comparison in this thesis, 10% of these images is used for validation purposes and another 10% is dedicated to the test set. Therefore, 80% of the images (i.e., 5000 images) are used for training. This is summarized in Table 3.2.

3.5 Encoding trail types

Trails can be characterized in terms of their type. Fortunately, it is not necessary to manually select the trail type as there already exist standard trail labels and for many trails the assignment of label to specific portions of a trail network have already been performed. Such representations can be found in topographical maps like the OpenStreetMap databases that are available online.

Formally, the OpenStreetMap format is fully described in [85]. A short review of relevant portions of

the representation is described below. For full details on OpenStreetMap see [85]. From a data model point of view, an OpenStreetMap consists of three basic components, called 'Elements'. Each Element has an unlimited number of associated tags for defining properties of the element. The three basic elements are 'Node', 'Way' and 'Relation'.

- Node: A Node defines a specific point on the earth and in its most simple version is comprised of a latitude and longitude and an id number. Nodes may have some tags to indicate more information about a particular point in the environment, for instance a traffic signal (highway=traffic-signals) and are also used to define the shape of Way elements of the map.
- Way: A Way defines a linear feature in the environment such as a road, trail, or river. Ways are ordered collection of nodes. The maximum number of exploited nodes in a way is 2000, although multiple ways can be extended via the relation element to represent larger features. the way element is also used to describe solid polygons such as the shape of a building or the coast of a lake. Such ways are known as closed ways.
- Relation: A Relation is a data structure to relate two or more elements on the map. They can be used to bond a number of ways together to make a bicycle route or a bus route, giving information about restricted turns in an intersection or making a multi-polygon shape of a building with its perimeter as 'outer way' and its holes as 'inner way'.

Tags add information to an element and consist of a (key,value) pair written as (k,v) in OpenStreetmap. For instance to define a way as a footway, one would add a tag to the corresponding way as shown below:

k ="highway" v ="footway"

Here k refers to 'key' component and v defines the 'value' of introduced key.

```
<way id="204416108" version="2" timestamp="2014-09-19T00:17:43Z">
 <nd ref="653634313"/>
 <nd ref="837788249"/>
 <nd ref="2144235246"/>
 <tag k="surface" v="dirt"/>
 <tag k="highway" v="footway"/>
 <tag k="name" v="Trans Canada Trail"/>
</way>

                .
                .
                .

<node id="653634313" version="2" timestamp="2013-02-06T17:09:26Z" lat="43.8615192" lon="-80.0599583"/>
```

Figure 3.9: *A sample block of .osm file defining a way and a node reference.*

OpenStreetMap maps are stored in an XML formatted file with the .osm extension. A snippet from such a file is shown in Figure 3.9. Although they are stored in an XML representation, one difference between XML and the representation used by OSM is that in OSM the order of the nodes in an element can have meaning. For example, a Way is an ordered collection of nodes. Maps for many environments can be found on-line and the OpenStreetMap project maintains a large database of maps on its web page www.openstreetmap.com.

All pathways in an OSM map are defined by a tag with the key 'highway' and different values for different types of paths. Some of these path types are accessible for robot navigation. Table 3.3 shows different road types of interest for robot navigation. Some of the useful surface types for robot navigation are described in Table 3.4.

Extracting highways from OSM maps. There exist a number of open source tools that can manipulate OSM maps. JOSM[86] is a Java-based tool that simplifies the processing of OSM maps. For our point of view JOSM can be used to select map features matching a particular set of properties (e.g., within a given area and matching a particular map feature property, such as being a highway). The result of such selection is shown in Figure 3.11. Figure 3.11a shows a portion of York University in OSM format. Figure 3.11b shows the same sub-region but with all features except highways removed.

Table 3.3: *Tags for different road types in OSM maps reprinted from [85]. The image on the right suggests an example of the given road type.*

Key	Value	Notes	Image
highway	pedestrian	For roads used mainly/exclusively for pedestrians.	
highway	track	Roads for mostly agricultural or forestry uses.	
highway	path	A non-specific path.	
highway	footway	For designated footpaths; i.e., mainly/exclusively for pedestrians. This includes walking tracks and gravel paths.	
highway	bridleway	For horses.	
highway	cycleway	For designated cycleways.	
highway	unclassified	The least most important through roads in a country's system. Often link villages and hamlets.	
highway	service	For access roads to, or within an industrial estate, camp site, business park, car park etc.	

Table 3.4: *Tags for different surface types of the roads in OSM maps reprinted from [85]. The image on the right suggests the type of surface given.*

Key	Value	Notes	Image
surface	asphalt	This is actually short for asphalt concrete.	
surface	sett	Sett surface is formed from stones quarried or worked to a regular shape.	
surface	concrete	Cement based Concrete, forming a large, continuous surface, typically cast in place.	
surface	paving-stones	Paving stones are equally sized concrete stones, with a flat top.	
surface	dirt	Some compacted roads are sometimes called dirt as well.	
surface	grass	Grass covered ground.	
surface	gravel	Broken/crushed rock with sharp edges, known as ballast on railways. Usually loosely arranged.	

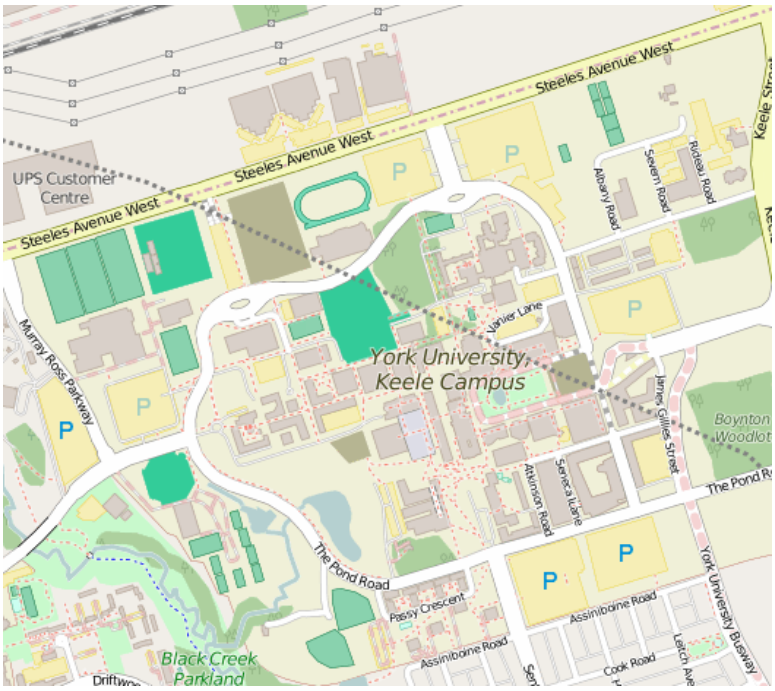
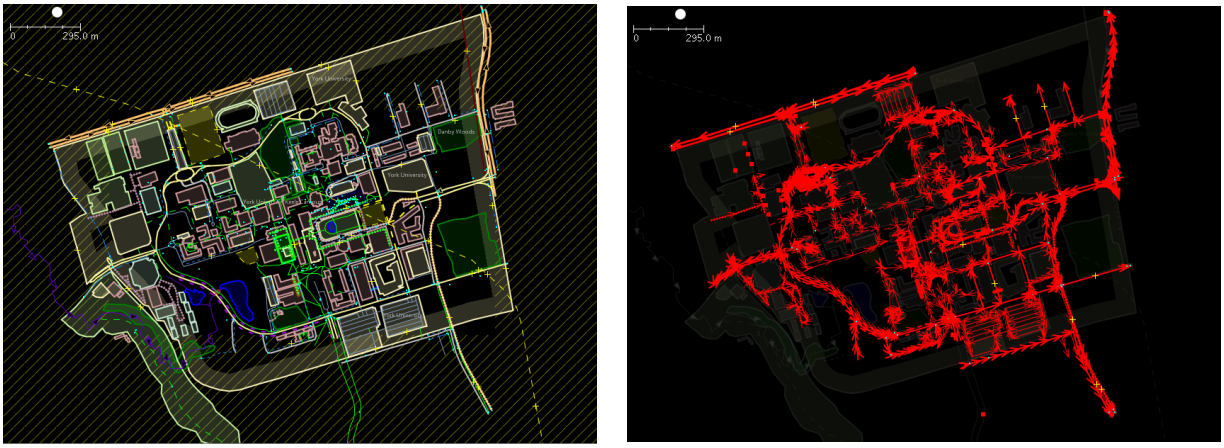


Figure 3.10: *OSM map of York University campus [67].*



(a) *A portion of York University in OSM format*

(b) *Same sub-region only with highway features extracted*

Figure 3.11: *York University OSM map in JOSM software and the extracted road network of its map.*

3.6 Example training

In this section we walk through the training process of one small dataset to illustrate the basic algorithm and its performance. All the training use the Inception-V3 network provided by the open source library

Table 3.5: *Hyper-parameters of the training example introduced in this sections.*

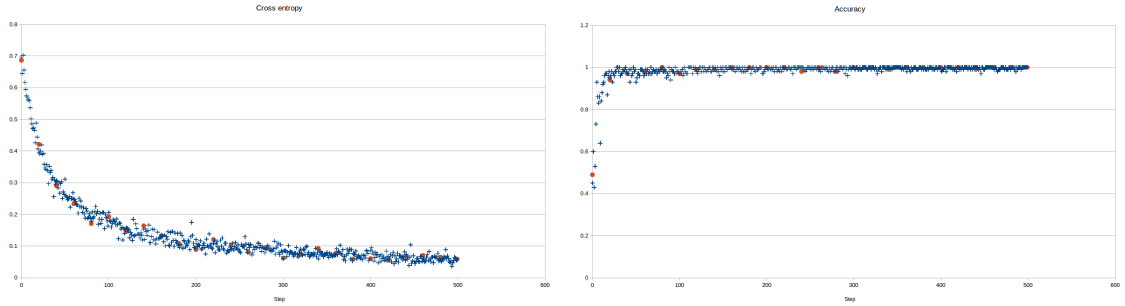
Training hyper-parameters	**Value**
epochs	5
learning rate	0.02
train batch size	100
validation batch size	100
random brightness	±10%
# images per label for training	5000
# images per label for validation	625
# images per label for testing	625

TesnsorFlow [87].

In order to apply transfer learning method on a pre-trained model of Inception-V3 on an ImageNet dataset for the target labels of TrailNet, the classification layer and softmax layer of the Inception-V3 model were replaced with a new classification layer which includes 2048 nodes and a softmax layer with n outputs, where n is the number of classes in each version of the TrailNet dataset. This follows the same model as shown above in Table 3.1 with the softmax layer with the size of $1 \times 1 \times 2$ as this example only consist of two classes of data.

For the purpose of this sample training process, a simple classification task on a subset of the TrailNet dataset is described. For exposition purposes, here we classify a subset of the TrailNet dataset which has been labeled as being either 'straight' or 'curved' and this is used to retrain the Inception-V3 network architecture with the same hyper-parameters and number of images per class as the main classification task described in the following chapter. Therefore, this subset has 5000, 625 and 625 images per class label for training, validation and testing, respectively. The number of images in each training step is called the "batch-size". The number of steps for training a network for n epochs, with m images per class and k labels is calculated as:

$$\textbf{Number of steps} = \frac{(m \times k)}{\textbf{batch-size}} \times n \tag{3.1}$$



(a) *Cross-entropy* ***(b)*** *Accuracy*

Figure 3.12: *Inception-V3 retraining results the sample straight versus curved subset of the TrailNet dataset. Blue point show the training results and orange points depict the validation results during training. Horizontal axis in all the charts is the number of steps in the training process. Here you can see that the accuracy of the classification task is saturated at approximately* 99% *after about* 200 *steps, while the cross entropy term continues to decline out until around the* 500^{th} *step.*

To make our training approach more robust to the different lightning conditions, in the process of training, the brightness of each image is randomly modified through the addition of random Gaussian noise with a sigma equal to 10 percent of the original image brightness. In training the DNN's, other parameters rather than weights and bias values of the nodes, are called hyper parameters. These hyper-parameters include learning rate and epochs (number of iterations on the whole training dataset). Hyper-parameters used in this evaluation are chosen to be the same as those that will be used later in Chapter 4 in order to be able to compare different TrailNet versions in a fair manner. The learning rate was set to 0.02 and for each training exercise, and the training process was repeated for 5 epochs. The network was trained on 5000 images per each class of the image dataset and 625 images per each class were used for validation set. The test set for each version also contains 625 images per each class. Table 3.5 shows a list of the hyper-parameters used in this training example.

The training process took place using the Elastic Cloud Computing (EC2) computers of the Amazon Web Services (AWS) online computing facilities [88]. Amazon provides such online computing for different heavy

Table 3.6: *Examples of correct and wrong classification results of the sample training on the TrailNet dataset.*

Straight	**Not-straight**	**Missclassified as straight**	**Missclassified as not-straight**
			
			

computational tasks; specifically there are some servers providing high-throughput GPU's for DNN training purposes. For this study we exploited the "p2xlarge" module of AWS which has a single NVIDIA K80 Tesla GPU [89]. Based on the memory capacity of the GPU used in this work, in each step of training the CNN, only a batch of 100 images was given to the network at a time. Therefore, in this example as there are two classes with 5000 images per class, and 100 images per step for training. 500 iterations were required to go over all the training files for 5 epochs following Equation 3.6: $(5000 \times 2)/100 \times 5 = 500$. All required files for the the training, validation and testing, along with the pre-trained model of Inception-V3 network were uploaded to the EC2 computer and after installation of all the required libraries for running TensorFlow using GPU (more details on installing TensorFlow and dependent libraries can be find in [87]), training takes place. In parallel every 20 steps, validation of a batch of 100 images is processed. The cross-entropy and accuracy charts for this training and validation example are shown in Figure 3.12. After the training on the dataset is processed for 5 epochs (500 steps), the accuracy of the retrained network is tested on the test set. The final accuracy of the classification task on the test set is 99.2%. Table 3.6 shows some of the correctly

classified images along with examples of missclassification of the network.

3.7 Summary

This chapter described the basic CNN developed for this work and how output for this CNN can be used to construct an appropriate steering angle for the robot itself. This chapter also described the TrailNet training dataset that will be used in the following chapter to guide camera geometry, number of cameras, camera field of view, and trail-type specificity in terms of driving a robot along trails. The chapter also described a worked example of the modified network on straight versus curved roads. The following chapter uses this network to identify the camera geometry and camera field of view most appropriate for trail following.

Chapter 4

Experimental results

This chapter describes the retraining process of the Inception-V3 network on different versions of TrailNet and evaluates their performances on multiple offline criteria. In addition to this laboratory-based evaluation, the performance of the algorithm for the task of autonomous trail following is also evaluated.

4.1 CNN recognizing straight ahead, turning left and turning right trails

In order to evaluate the effect of different number of cameras, the nature of the camera (e.g., wide or narrow field of view) and also the camera geometry, the classification layer of the pretrained model of Inception-V3 CNN was retrained on different TrailNet trail types (described in Table 3.2). The training step follows the procedure discussed for the simple example described in Section 3.6. Hyperparameters used in the following training steps were common over the different trail types and are provided in Table 4.1.

The full set of tests would include four different trail types (asphalt, concrete, dirt and concrete) ×

Table 4.1: *Hyperparameters of the training steps on TrailNet dataset.*

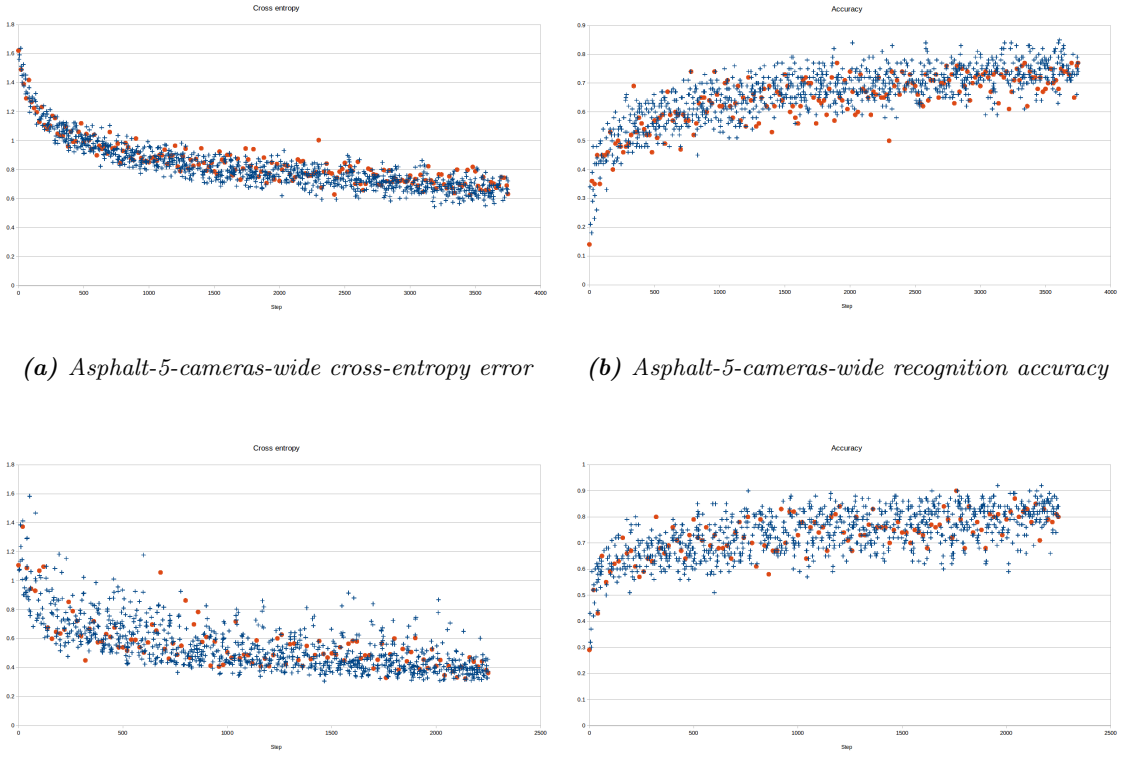
Training hyperparameters	**Value**
epochs	5
learning rate	0.02
train batch size	100
validation batch size	100
random brightness	±15%
# images per label for training	5000 ± 250
# images per label for validation	625 ± 100
# images per label for testing	625 ± 100

narrow versus wide camera field of view × different numbers of cameras and specific camera geometries. Evaluating all possible combinations is clearly quite expensive and beyond the scope of this work. In order to reduce the number of potential tests, here we first concentrate on the asphalt trail type and then once a mechanism for trail following for asphalt trails has been identified, these results are applied to the other trail types considered here.

4.1.1 Camera number and geometry

Three cameras or five and their field of view. For trail following should cameras with a wider field of view be preferred over narrower ones? Wider fields of view will capture more of the scene, but does this wider field of view correspond to improved trail heading recognition? We begin by considering the camera geometry and then move on to question of field of view.

Although it might be desirable to try every possible camera arrangement in every possible orientation to find an optimal camera geometry, this is not a practical option. In order to reduce the possible combinations, we first consider the question of suitable camera separation for training. In [18] training was performed with cameras pointing straight ahead and at ±30 degrees offset for an aerial vehicle. Although this may have



(a) Asphalt-5-cameras-wide cross-entropy error

(b) Asphalt-5-cameras-wide recognition accuracy

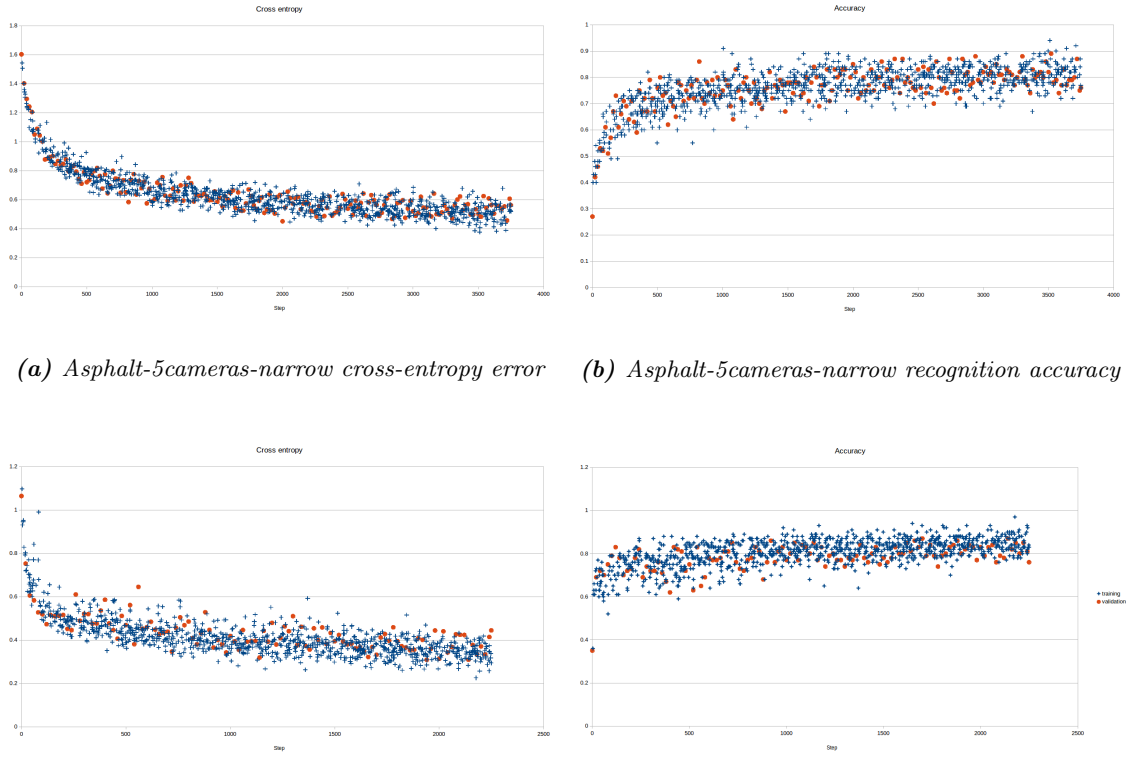
(c) Asphalt-3-cameras-wide cross-entropy error

(d) Asphalt-3-cameras-wide recognition accuracy

Figure 4.1: *Inception-V3 retraining results summary on Asphalt-5-cameras-wide and Asphalt-3-cameras-wide datasets. Blue points show the training results and orange points depict the validation results during training. Horizontal axis is the number of steps in the training process.*

been an appropriate geometry for the camera and vehicle used in [18], is a similar geometry suitable for a ground-contact robot and the cameras used in this work? The assumption here is that the range required to control a land autonomous robot on a trail is from -60 to $+60$ degrees with respect to the heading direction of the trail. In order to cover this geometry, two different geometry setups are evaluated here: 1) five cameras with a 30 degrees offset between adjacent cameras in the range of -60 to $+60$ degrees inclusive, and 2) three cameras with a 60 degrees offset between adjacent cameras in the range of -60 to $+60$ degrees inclusive.

As the number of labels increases in a CNN, the accuracy of the network tends to decrease, specially if there exists a correlation between the labels. There are more distractor classes so a reduction in accuracy



(a) Asphalt-5cameras-narrow cross-entropy error *(b) Asphalt-5cameras-narrow recognition accuracy*

(c) Asphalt-3-cameras-narrow cross-entropy error *(d) Asphalt-3-cameras-narrow recognition accuracy*

Figure 4.2: *Inception-V3 retraining results summary on Asphalt-5cameras-narrow and Asphalt-3-cameras-narrow datasets. Blue points show the training results and orange lines depict the validation results during training. Horizontal axes are the number of steps in the training process.*

is to be expected generally. In the case for trail following this accuracy will be decreased further as the various classes have a high similarity and correlation in their information. For the datasets considered in this thesis, the images of different labels have a number of features in common as they are images of a trail with just a rotation of the camera and there exist portions of overlap between them. Table 4.2 shows the average recognition and maximum accuracy for different combinations of number of cameras (three versus five) and the nature of the cameras used (wide versus narrow field of view). The process of the training the Inception-V3 network to process these datasets is shown in Figures 4.1 and 4.2. Although having five cameras instead of three increases the resolution of the dataset, the additional classification classes increases

Table 4.2: *Inception-V3 test accuracy on asphalt versions of TrailNet dataset. The three camera geometry outperforms the five camera geometry using both he narrow and wide field of view as measured by average performance on the TrailNet dataset.*

Dataset Name	**Average**	**Maximum**	**Field of view**	**Camera Geometry**
Asphalt-5-cameras-wide	71.77%	74.00%	wide	5 cameras (0, ±60° and ±30°)
Asphalt-3-cameras-wide	82.07%	83.70%	wide	3 cameras (0 and ±60°)
Asphalt-5-cameras-narrow	80.35%	81.40%	narrow	5 cameras (0, ±60° and ±30°)
Asphalt-3-cameras-narrow	82.32%	82.70%	narrow	3 cameras (0, ±60°)

the opportunity of the algorithm to miss classify the input image. As can be seen in Table 4.2, the three camera geometry out-performs the five camera geometry both as measured by the mean recognition rate for both the narrow and wide field of view cameras. Indeed the only time that the wide field of view camera out-performed the narrow is for the asphalt-3-cameras-wide dataset versus the asphalt-3-cameras-narrow one.

Camera field of view Table 4.2 also shows that wide field of view and narrow field of view camera arrangements have very similar recognition rates for the 3 cameras version dataset, however, on average the narrow field of view camera average recognition rate out-performs the wide field of view recognition rate. This may be due to the fact that when wide field of view cameras are used that then the similarity between the +/-60 camera and the straight ahead camera is increased. In any event, the best average performance is found with the -60, 0, +60 camera geometry using the narrow field of view cameras.

Summary Given the poorer performance on the five camera system relative to the three camera system, further work concentrated on the three camera setup with cameras at -60, 0 and +60 exclusively, using a narrow field of view. The reported test accuracy results are based on the recognition rate of the trained network on a test set of images selected from the dataset, which was not shown to the network during the training process. However, there is always a correlation between images used for training and testing as

Table 4.3: *Independent dataset test accuracy. Although the asphalt-3-cameras-wide and asphalt-3-cameras-narrow DNN performed well during training, performance on an independent dataset was not as successful with lower overall accuracy and, perhaps more critically, systematic difficulty in recognizing one of the -60 or +60 classes.*

Training dataset	**Independent test accuracy**				**Test accuracy during training**
	Overall	$+60°$	$-60°$	$0°$	
Asphalt-3-cameras-wide	57.90%	84.65%	**00.89%**	100.0%	83.70%
Asphalt-3-cameras-narrow	60.56%	62.87%	**35.71%**	89.71%	82.70%

they are from the same dataset and have been assigned randomly into training and testing groups. In order to evaluate the performance of the trained networks for a real world application, the trained networks were re-evaluated on data streams obtained from cameras mounted on the robot operating on similar asphalt trails on a different day. The results are summarized in Table 4.3. Surprisingly, the experimental results outside on asphalt trails suggested that the trained network is only able to recognize the straight ahead images (i.e., $0°$) and either $-60°$ or $+60°$ images correctly. But the DNN is not able to distinguish between $-60°$ or $+60°$.This dataset was collected by the robot over 100 meters of an asphalt trail near the Sherman building on York University's Keele campus. This result was found for both the narrow and wide field of view cameras. The performance rates highlighted in bold illustrate these cases.

In order to investigate the reduction in recognition performance in comparison with the training/testing dataset collected earlier, the network was retrained several times with different augmentations on this new dataset. For data augmentation purposes, instead of the original retina size of 299×299 pixels, a larger retinal with the size of 349×349 pixels was chosen. In order to make the training results more robust towards small rotations and translations of the camera with respect to the trails, a 299×299 retina was chosen randomly from within the images of 349×349 pixels in each training step. The same training process was repeated three times for re-randomization of the training data. Results are shown in Table 4.4. A breakdown in the test recognition accuracy for different labels (i.e., $0°$, $-60°$ and $+60°$) for these three DNN

Table 4.4: *A breakdown of test accuracy with 50 pixels data augmentation on the narrow dataset with three cameras (−60°, 0° and +60°) on three different trials on the same datset with an independent test set from the training dataset. Anonymously low accuracy rates are highlighted in bold.*

Trial	**Independent test accuracy**				**Test accuracy during training**
	Overall	+60°	−60°	0°	
1	62.39%	**12.87%**	99.55%	72.00%	91.10%
2	73.90%	**33.13%**	92.26%	99.40%	95.10%
3	70.94%	100.0%	**13.53%**	98.80%	96.30%

Table 4.5: *Independent dataset test accuracy on only ±60° angles.*

Training dataset	**Independent test accuracy**			**Test accuracy during training**
	Overall	60°	−60°	
Asphalt-left-right	58.64%	99.62%	**17.66%**	92.90%

on the same dataset suggests that the network is unable to correctly recognize −60° and +60° angles, while in all the tests, the recognition rate for the 0° label was reasonably high (see Table 4.4). Interestingly each of the tests for one of the +60° or −60° tests showed a reasonably high accuracy while the other one did not. In order to investigate the issue more fully, a new network was trained with only −60° and +60° angle labels of the latter dataset. Table 4.5 depicts the label-wise and overall accuracy of this test. Results confirmed the earlier suspicion that the trained dataset had difficulties differentiating between the −60° and +60° conditions. Taken together, these results suggest that although Inception-V3 showed good accuracy during training, the actual errors are not evenly distributed throughout the data. Rather, the trained network had some difficultly distinguishing between the −60° and +60° categories when presented with novel data.

It would appear that Inception-V3 pretrained with ImageNet dataset is not suitable for distinguishing mirrored features (i.e., the left and right views are mirrored versions of each other). It might be argued that this confusion arose from our use of mirrored images for both the left and the right view, and that

Table 4.6: *A breakdown of test accuracy with 50 pixels data augmentation on a dataset with real left and right narrow images (i.e., not mirroring right images to get left images) with 3 cameras on two different trials with an independent test set from the training dataset. Anonymously low accuracy rates are highlighted in bold. For the first trial 50 pixels augmentation and for the second trial 100 pixels augmentation is used.*

	Independent test accuracy				
Trial	**Overall**	60°	-60°	0°	**Test accuracy during training**
1	66.88%	**15.84%**	99.55%	84.00%	91.30%
2	66.88%	**07.92%**	98.21%	94.85%	91.00%

data obtained from real cameras might not exhibit such regularity. In order to consider this a new dataset with two cameras pointing right ($+60^\circ$) and left (-60°) was captured on a 100 meters asphalt trail on York University campus near the Sherman Building. The same training hyperparameters were used for this training session. Table 4.6 shows the results of this test. As it is shown in Table 4.6, even with this new dataset the same deficiency in recognition of $+60^\circ$ and -60° was observed. One possibility for the fine tuned Inception-V3 network being unable to distinguish between $+60^\circ$ and -60° classes on novel dataset is that training with ImageNet dataset using random flipping of training images for data augmentation purposes may have introduced an insensitivity to horizontal reflection of features. Such views, say of objects like cars, include views from both sides, which blur left and right views into a single class. It is also possible that the achieved good performance on the training dataset on these mirrored classes (during test evaluation of the novel dataset) was due to the high level of correlation in the training and test dataset.

Summary Although recognition accuracy can be quite good for a single camera, a key issue is that the confusion between $+60^\circ$ and -60° will be catastrophic for a robot if motion commands are generated based on the output of this stage. Given this, it is worthwhile considering other approaches in terms of utilizing classification to control the vehicle.

Table 4.7: *Confusion matrix for the straight versus not-straight trained networks on the test sets from their corresponding trails and other trail types. Poorly performing network-trail type combinations are highlighted in bold.*

	Trail type			
NN	asphalt	concrete	dirt	gravel
asphalt	99.90%	96.69%	**63.78%**	87.26%
concrete	85.34%	99.75%	**66.60%**	**70.10%**
dirt	92.40%	93.95%	97.77%	95.15%
gravel	84.72%	98.83%	84.33%	99.13%

4.2 Are different CNN's required for different trail types?

In order to study the effect of the trail type in the task of deviation angle recognition on trails, the performance of each road-type (asphalt, concrete, dirt, gravel) network was tested on a test set from each of the trail types. Table 4.7 shows the confusion matrix of these performances. First, the "correct" recognizer works well on the "correct" trail type (the diagonal in Table 4.7). Second, the "wrong" recognizer shows reasonably good accuracy on the "wrong" trail type with some exceptions. The dirt network shows a reasonably good performance on all the other trail types, the asphalt network is reliable for asphalt and concrete, the concrete network shows a good performance only on concrete, and the gravel network also has a good ability to recognize the correct labels for the concrete dataset and the gravel dataset itself. But for best performance, a properly tuned trail-specific CNN performs best.

4.3 CNN recognizing straight ahead versus turning on trails

In order to deal with the inability of the pre-trained Inception-V3 to reliably differentiate between the robot deviating to the left and turning to the right, here we explore using the pre-trained Inception network to differentiate between straight ahead trails and trails that are bending away from the straight ahead direction.

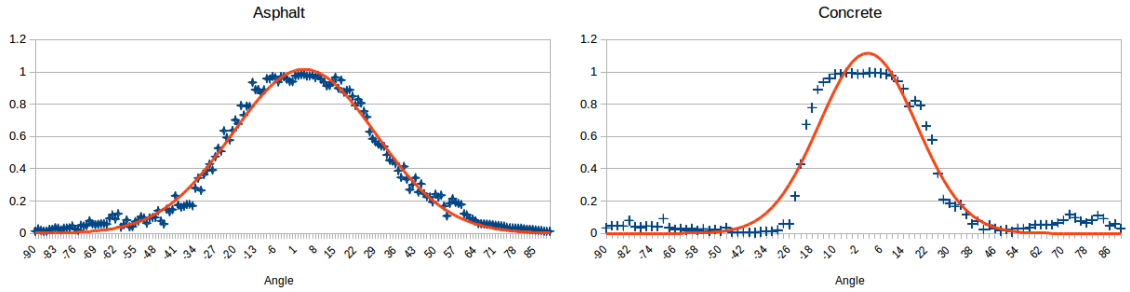
Table 4.8: *Detailed test accuracy results of the straight/not-straight method with the three narrow camera geometry setup on all trail types. The* 0° *and overall columns show recognition rate for the two classes. To verify that there was no bias in terms of the* -60° *and* $+60^\circ$ *categories which made up the not-straight category and* 0° *which is the straight category.*

	Test accuracy			Subcategories accuracy		
Training dataset	**Overall**	**Not-straight**	**Straight**	$+60^\circ$	-60°	0°
Asphalt-3-cameras-narrow	99.90%	99.95%	99.90%	100.0%	99.90%	99.90%
Concrete-3-cameras-narrow	99.75%	100%	99.20%	100.0%	100.0%	99.20%
Dirt-3-cameras-narrow	97.77%	100%	99.20%	100.0%	100.0%	99.30%
Gravel-3-cameras-narrow	99.13%	99.88%	98.14%	100.0%	99.24%	98.14%

In this method, instead of training the CNN to learn three different angles of 0°, -60° and $+60^\circ$, labels are either straight (i.e., 0°) or not straight (i.e., -60° and $+60^\circ$). The hyperparameters of this training session are the same as those given in Table 4.1, but augmented by doubling the number of training images and test images of not-straight labels in order to increase the diversity of trail features introduced in those different angles. The recognition rate on this new training experiment were considerably higher than that found with the CNN when it was forced to choose between straight ahead, left and right (Table 4.8). This is, of course, to be expected in part due to the reduced number of possible labels that can be associated with a given input image. Unlike the earlier approach, the recognition rate (the overall column in Table 4.8) for the not-straight label does not drop for mirrored angles (not-straight label) and it stays high for both +60 deg and -60 deg within the not-straight category.

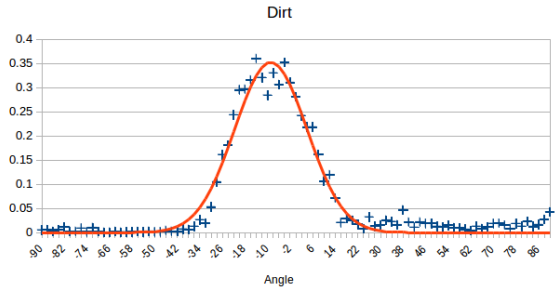
4.4 Sensitivity evaluation

Although the Inception-V3 classification network for straight ahead versus not-straight roadway performance is quite good, the Inception-V3 classification accuracy alone is not enough to exploit CNN's in the task of autonomous trail following. Can we utilize multiple cameras each trained to identify 'straight head' versus 'curved' roadways to drive the robot? Here we explore the sensitivity of trained networks to different trail

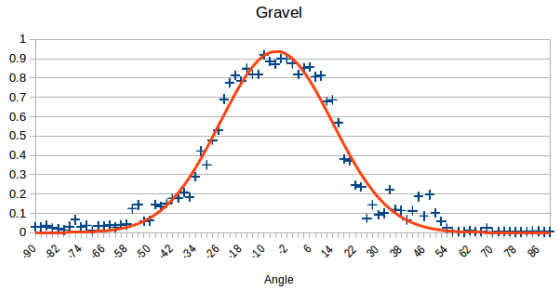


(a) Asphalt

(b) Concrete



(c) Dirt



(d) Gravel

Figure 4.3: *Blue points in the charts depict the* $0°$ *label output results of the trained networks on images between* $-90°$ *and* $+90°$ *on their corresponding trail type. Orange lines show a normalized to 1 Gaussian function fitted to the plotted results. The Gaussian fit is normalized to the maximum of the response curve. Note the different vertical axes for the various plots.*

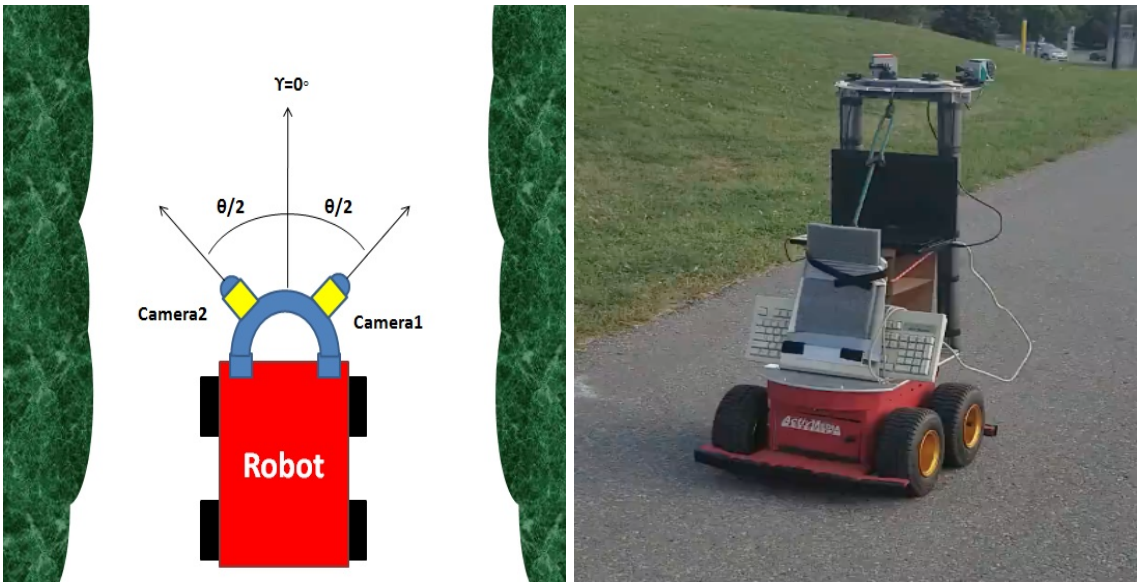
angles in the range of $[-90°, 90°]$. A new set of images was captured from trails of different types. This dataset was captured from five different spots on each trail type. At each capture point images were captured pointing along the trail in the range of $[-90°, 90°]$ degrees with respect to the heading direction of the trail in 2 degrees increment (i.e., angles of captured images on each spot are $[-90°, -88°, -86°, ..., , -2°, 0°, 2°, ..., 88°, 90°]$). Therefore, this dataset consist of a total of $90 \times 5 = 450$ images for each trail type. This allows us to estimate the directional sensitivity function for each of the trained networks on their corresponding trail (see Figure 4.3). For each response curve a normalized Gaussian is fit. This Gaussian is normalized to the maximum response of the CNN for the given trail type. The nature of the fits suggest that the networks

have a symmetrical sensitivity for the positive and negative deviation angles from the trail. The Gaussian-like response fit also suggests that a classical difference of offset Gaussian (DOOG)[90] approach might be exploited to combine multiple CNN response curves to estimate the direction of the trail relative to the robot.

4.5 Differential method

The previous section shows that it is possible to train a CNN based on the Inception V3 pre-trained CNN to identify if the robot is following the trail or if it has deviated to the left or the right from the trail, but not which way it has deviated. Note, unfortunately, that the absolute maximum of the CNN is not a reliable estimate of the robot's direction with respect to the trail given the change in magnitude of the response curve for different trail types. Given that one camera is insufficient, one idea is to use a camera pair to estimate the directions. The approach developed here utilizes a difference of offset Gaussian (DOOG) [90] approach to the problem. The sensitivity curves suggest a Gaussian response for each detection and by combining the detection radially we can distinguish between deviation of the robot to the left or to the right. Figure 4.4 shows the two cameras mounted onboard the robot. Using this approach, CNN are applied simultaneously to the image streams of the two cameras, and the output of these CNN are combined using their differences and this is used to compute the necessary steering angle for the robot.

In order to find the optimal offset angle between the cameras, the sensitivity functions of both cameras with respect to the deviation angle of the robot γ is computed. Assume that the sensitivity function of the right and left cameras are modeled by the two Gaussian distributions are G_1 and G_2, respectively, where they have the same standard deviation of σ and means of $\mu_1 = \frac{\theta}{2}$ and $\mu_2 = -\frac{\theta}{2}$, respectively. The combined sensitivity function of both cameras is computed as the addition of G_1 and G_2 as $G_+ = G_1 + G_2$. The difference between G_1 and G_2 is denoted as G_- which is given by $G_1 - G_2$. In order to have a stable overall sensitivity function G_+ and preventing it from having a ripple in the center, θ should be chosen in a way

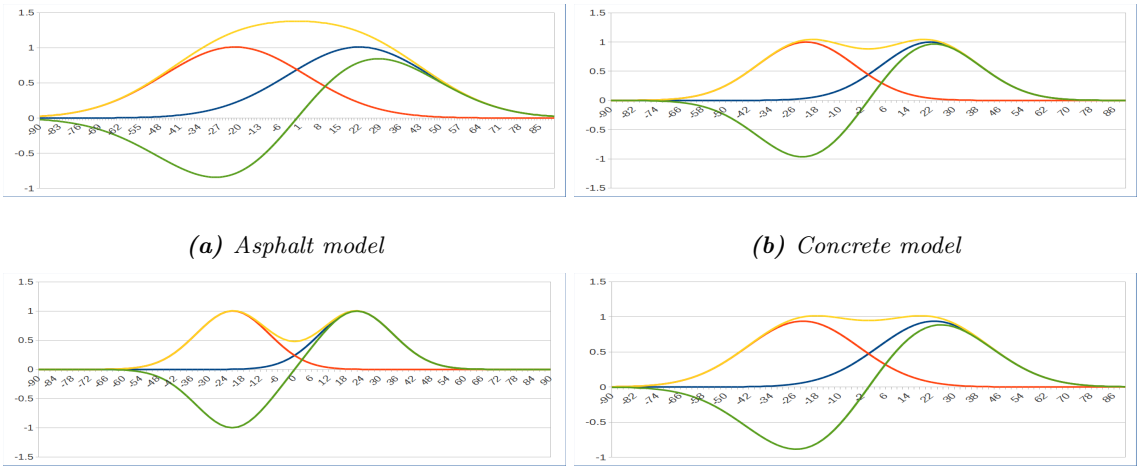


(a) *(b)*

Figure 4.4: *Conceptual visualization of using two cameras on the robot (a) and the actual robot on an asphalt trail on the the campus of York University.*

that at $\gamma = 0$, $G_1 = G_2 \geq 0.5$. A Gaussian distribution reaches the half of its amplitude $\sqrt{2ln(2)}\sigma$. This happens approximately when $\mu_1 = -\mu_2 = 1.17 \times \sigma$. Therefore, the best offset angle between the cameras is to separate the cameras from each other one on the right and the other one on the left by this value. Figure 4.5 shows G_1, G_2, G_+ and G_- for the ideal σ for each trail type as well as the average across trail types.

Table 4.9 illustrates the characteristics of the Gaussian distributions of all of the networks on their corresponding trail type and their averages across the different trail types. The average standard deviation $\sigma_{\mathbf{avr}} = 18.59$ of all the trail types Gaussian model was used for the trails and tests that follow. Figure 4.5 depicts the G_+ charts for all of the networks with the corresponding generic camera offset angle $\theta/2 = 21.75°$ (see Figure 4.4).



(a) Asphalt model *(b) Concrete model*

(c) Dirt model *(d) Gravel model*

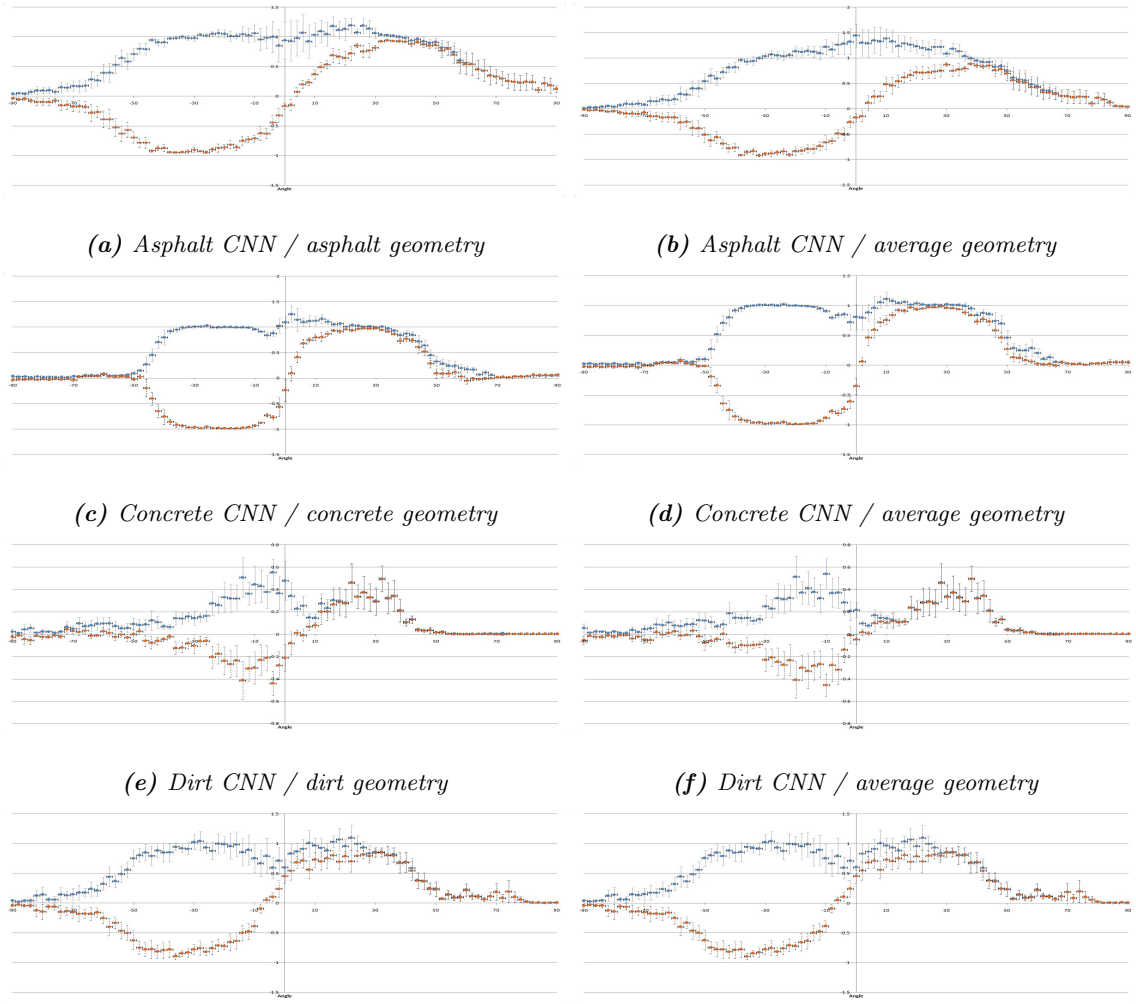
Figure 4.5: *Charts include the distributions of G_1, G_2, G_+ and G_- with blue, orange, yellow and green curves, respectively. G_1 and G_2 are the Gaussian fits of each network and are shifted to $\sqrt{2ln(2)}\sigma$ and $-\sqrt{2ln(2)}\sigma$ mean in the charts with their ideal σ. The horizontal axis of all charts is the angle of the heading of the robot γ with respect to the trail direction (see Figure 4.4).*

Table 4.9: *Gaussian model parameters for each network.*

NN	μ	σ
Asphalt	3.96	24.85
Concrete	1.47	16.99
Dirt	-8.86	12.85
Gravel	-5.68	19.69
Average	-2.27	18.59

After maximizing the sensitivity function G_+, the next step is to compute a proper function with G_1 and G_2 functions in order to compute the steering angle command for the robot. Subtracting two identical Gaussian distributions with an offset between their mean values, results in a Gabor-shaped filter G_-, known as the Difference Of Offset Gaussians (DOOG) [90]. (see Figure 4.5).

Figure 4.6 shows the performance of the differential CNN on the different trail types including the



(a) *Asphalt CNN / asphalt geometry*

(b) *Asphalt CNN / average geometry*

(c) *Concrete CNN / concrete geometry*

(d) *Concrete CNN / average geometry*

(e) *Dirt CNN / dirt geometry*

(f) *Dirt CNN / average geometry*

(g) *Gravel CNN / gravel geometry*

(h) *Gravel CNN / average geometry*

Figure 4.6: *Charts show experimental values of G_+ (blue) and G_- (orange) with standard errors. (a),(c),(e) and (g) show performance with the trail specific camera separation angles while (b),(d),(f) and (h) show performance with the average camera separation angle. The horizontal axis of all charts is the angle of the heading of the robot γ with respect to the trail direction (see Figure 4.4). The graphs are quite similar to the theoretical G_+ and G_- plots shown in Figure 4.5. However, due to the low standard deviation of Gaussian response of dirt CNN, with the ideal angle, the G_+ graph provides lower quality response close to $0°$ angle.*

performance of the average geometry CNN on the trails as well. For each differential CNN, its performance computed over the different angles for each of the five sample points on each trail, and the average results and

the standard errors are plotted. For each G_+ and G_- are shown. Several observations can be made. First, the curves plotted in Figure 4.6 are similar to those shown in Figure 4.5. Second, the average differential camera geometry given the values in Table 4.9, coupled with tuned CNN provide satisfactory performance eventhough the tuned differential CNN show superior performance in general. Finally, it is not possible to use the magnitude of G_- independently of G_+ when estimating the appropriate steering angle.

From trail orientation to steering angle. Computing the steering angle command $\dot{\theta}_{\mathbf{steering}}$ to drive the robot along the trail involves estimating $\dot{\theta}_{\mathbf{steering}}$ from G_- and G_+. There are a number of practical issues involved here. First, if the detector cannot detect a road, then some default steering angle should be chosen or in a full application this information should also be sent to higher level control. Second, there is a desire to not oversteer (an overshoot) the vehicle motion, and finally, if the steering angle is only changing by a very small amount the robot should not introduce small zero-mean fluctuations into the steering angle (the controller should have a deadband region). Properly addressing all of these issues is beyond the scope of this thesis, so here we adopt a very simple strategy for obtaining a steering angle change from the (G+, G-) pair. If $|G_-| < th_-$, then we assume that the robot is in a deadband region, and set the change in steering angle to be zero. If $|G+| < th_+$, then the detector does not detect a trailway, and again we set the change in steering angle to be zero. For other angles we clamp the output change in steering angle to +/- 0.1 rad/s as:

$$\dot{\theta}_{\mathbf{steering}} = \begin{cases} 0.1(\frac{rad}{s}) & \text{if } G_+ > th_+ \text{ and } G_- > th_- \\ 0(\frac{rad}{s}) & \text{if } |G_-| < th_- \text{ or } G_+ < th_+ \\ -0.1(\frac{rad}{s}) & \text{if } G_- < -th_- \text{ and } G_+ > th_+. \end{cases}$$

In all the operating states of the robot, for the experiments described in the following section, the robot has a constant velocity of $20cm/s$.

4.6 Driving the robot with the differential CNN

In order to test the performance of the proposed algorithm for the task of autonomous driving on trails, a field trial was conducted on a range of trail types. Due to the battery capacities of the robot and hardware used in this experiments, tests have been broken down to 10 distances of 20 meters long. The following evaluations were used based on criteria typically use for autonomous driving systems (see [19]);

- The maximum distance travelled over the the 20 meter test ranges before human intervention was required.
- The minimum autonomous distance travelled over each of the ten 20 meter test ranges before human intervention was required.
- The average autonomous distance traveled over each of the ten 20 meter test ranges before human intervention was required.
- The average number of human operator interruptions required in order to drive the vehicle 20 meters.

Figures 4.6-4.9 show the paths chosen for the test on a map and a film strip of the robot in operation on all trail types. The chart of the results of the test on four different trails is shown in Table 4.10. Figure 4.11 plots a summary of these results.

The tests showed a number of interesting aspects of the algorithm. First, it performed very well on asphalt and concrete paths with the robot operating perfectly on asphalt and performing over 18m (in a 20m trial) without failure on average. Performance on dirt and gravel trails was less successful, but still showed quite good performance. It is interesting to note that the dirt trail was much narrower than the other trails tested (see Figure 4.9) but that even here performance was quite good. Worst performance was found on the gravel trail. One possibility here is that the control algorithm developed for gravel trails might need to be more tightly tuned for the gravel as the robot's tires have poor grip on this surface.

Table 4.10: *Autonomous field trial results for different trail types on 200 meters of paths shown in Figures 4.6-9 (a). The average values are computed based on the performance of the algorithm over ten different 20 meters pieces of path. Performance of the algorithm on asphalt and concrete paths was quit good, but deteriorated on dirt and gravel trails.*

Parameters	**Asphalt**	**Concrete**	**Dirt**	**Gravel**
Maximum distance travelled (m)	20	20	19	20
Minimum distance travelled (m)	20	12	8	5
Average autonomous distance travelled (m)	20	18.6	15.6	13.1
Average number of human operator interruptions	0	0.2	1.2	1.4

(a) *Map* ***(b)*** *Robot operation*

Figure 4.7: *Asphalt experiment. (a) shows a satellite view of the paths used for the field trial on asphalt trails available on campus of York University and (b) shows a film strip of the robot in operation. Green and red triangles show the start point and finish points of tests, respectively. The test was conducted on two different trails where the combined distance was 200 meters.*

(a) *Map* ***(b)*** *Robot operation*

Figure 4.8: *Concrete experiment. (a) shows a satellite view of the paths used for the field trial on concrete trails available on campus of York University and (b) shows a film strip of the robot in operation. Green and red triangles show the start point and finish points of tests, respectively. This trail provided 10 independent 20 meter test regions.*

4.7 Summary

This chapter described the process of evaluating the use of Inception-V3 network for off-road trail following. Experiments evaluated the nature, geometry and number of cameras that might be applied to the problem. Experiments demonstrated the following:

- Training on recognition tasks with wide and narrow field of view cameras on three cameras geometry datasets had average recognition rates of 82.07% and 82.32%, suggesting that narrower field of view cameras (here 60 degree) are as effective as larger field of view cameras (here 180 degrees).

(a) Map *(b) Robot operation*

Figure 4.9: *Dirt experiment. (a) shows a satellite view of the paths used for the field trial on dirt trails available on Crothers Woods Trails, Toronto, ON. (b) shows a film strip of the robot in operation. Green and red triangles show the start point and finish points of tests, respectively. One 100 meters path was used in two directions for test of the dirt trail.*

- Training with five orientation categories versus three found that the use of five cameras was not as effective in terms of recognition than was the three camera geometry (see Table 4.2).
- Although inception-V3 has been trained to successfully learn straight ahead versus twisted left and twisted right, we were unable to obtain satisfactory results for novel datasets in operation. It may be the case that the Inception-V3 network is unable to generalize the subtle differences between the +60 degrees and -60 degrees categories, or that some more sophisticated augmentation algorithm might have enabled this encoding.
- Although differentiating between -60, 0 and +60 degrees datasets did not generalize well, differentiating between 0 and ±60 degrees as two categories was successful. Based on this result a differential

(a) *Map* ***(b)*** *Robot operation*

Figure 4.10: *Gravel experiment. (a) shows a satellite view of the paths used for the field trial on gravel trails available on Crothers Woods Trails, Toronto, ON. Note that the trail region near the intersection with the larger trail was used in the tests but that performance in this region was good. (b) shows a film strip of the robot in operation.*

application of this system was developed. Using a DOOG model a trail following algorithm was successful in navigating different trail types using a common camera geometry and using tuned CNN for different trail types. End-to-end testing of the algorithm showed very good performance on asphalt and concrete paths, with poorer performance on dirt and gravel. Performance on dirt was still quite impressive given the narrow nature of such paths.

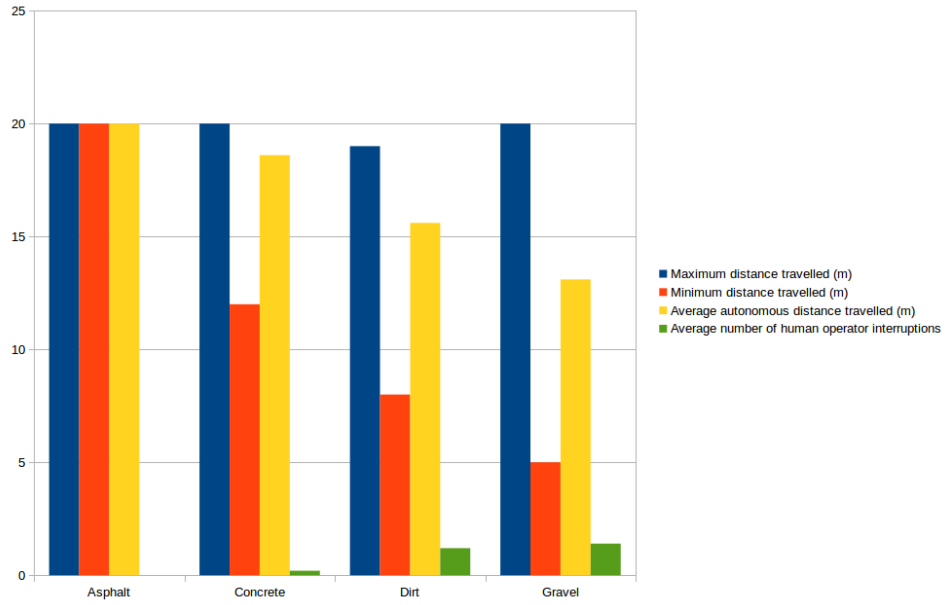


Figure 4.11: *Autonomous test results plot on different trail types (see Table 4.10 for details). Vertical axis is in meters for distance traveled and count for average number of human operator interruptions.*

Chapter 5

Summary and future work

5.1 Summary

Autonomous driving on off-road trails is a hard problem in robotics. Instead of redesigning a new architecture of CNN for the task of autonomous driving on trails, this work considered a transfer learning approach using a pretrained model of the Inception-V3 network.

An approach with a single Inception-V3 trained CNN showed that although it was possible to train the neural network to classify $0°$, $-60°$ and $+60°$ steering classes for asphalt trail type, this training did not generalize well for an independent test set. In fact, even with exploiting a range of different augmentations for the training, the resulted CNN were not able to distinguish between $-60°$ and $+60°$ classes. This misclassification makes the approach with a single Inception-V3 CNN dysfunctional for steering the robot on trails.

Further experimentation showed that the Inception-V3 network could be used to classify straight ahead

versus not-straight trail views, and that this classification was robust for both $-60°$ and $+60°$ trail angles. Based on the results, a differential CNN system was developed using a pair of cameras attached to the right and left of the vehicle. Sensitivity testing of the tuned CNN demonstrated that a differential (Difference of Offset Gaussian's) approach could be applied to the recognition score of the two CNN. Analysis of the sensitivity functions of the various CNN demonstrated that a camera separation of approximately 21 degrees to the left and right was an appropriate camera geometry. The trained CNN coupled with the two cameras were found to be successful in driving the robot for a range of different trail types. Tests were performed on a range of different trail types and demonstrated that the approach worked very well on asphalt and concrete trails with reduced but still good performance on dirt and gravel trails. Of some surprise was the good end-to-end performance on the dirt trails tests, given their narrow width size.

In training of this differential model, this work considered a number of different cameras geometries in order to tackle the trail following problem. To summarize these results here:

1. The G_+ and G_- values computed from G_1 and G_2 provide certainty and direction values. In the work presented here G_+ is given by the simple sum of G_1 and G_2. As shown in Figures 4.5 and 4.6 the G_+ signal is quite broad and perhaps a more tuned G_+ signal (e.g., $\sqrt{(}G_1^2 + G_2^2)$ might be considered) as an alternative.

2. This work considered the use of wide versus narrow field of view cameras. It was initially hypothesizes that the wide field of view cameras would be more effective than the narrow field of view cameras. However, recognition accuracy results suggest that narrow field of view cameras performed, on average, better than the wide field versions. One reason for this result may be that the wider field of view introduces a higher level of similarity in the images between the various cameras.

3. In development of a recognition system for roadway steering angle, a five cameras system was evaluated against a three cameras system. It was hypothesized the five cameras system would provide enhanced granulation in terms of recognition. However, the use of more recognition classes resulted a decrease

in the recognition accuracy.

4. The variety of the nature of the trail surfaces (e.g., asphalt, concrete, dirt and gravel), defines the need for the development of trail-type tuned CNN. Such training resulted in much higher recognition rates on their corresponding trail types. Although some of the trained networks seemed to perform acceptable on some of the other trails, but none of the trained network was strong enough to be used for all trail types. Therefore, knowing the trail type is clearly helpful. It would see that a map (containing information about the position of trails and their type) is a useful resource to be used for following. different trail types. Although recognition rates with Inception-V3 were acceptable once properly trained, the sensitivity of the resulting classifier to changes in the environment made the resulting detector unreliable. Fundamentally, the detector was not reliable in detecting -60 or +60 classes, although detection for straight ahead was quite good. Further evaluation of the detector found that it was quite good at discriminating between straight ahead and deviation from straight ahead. Based on this result, a differential model was developed with cameras pointing to the left and right of straight ahead and then integrating the output of these two sensors within a differential model. This model is similar to the 'difference of offset Gaussian' model [90].

5. Sensitivity analysis of the resulting detector showed good performance with the G+ and G- channels indicating 'strength of response' and 'deviation from straight ahead' over the various trail types.

6. Utilizing a simple control regime, the G_+ / G_- signals can be used to develop a control signal that drives the robot along various off-road trails using CNN tuned to the appropriate trail type and using a shared camera geometry.

5.2 Future work

This research developed a differential method for driving a robot along different trail types with two cameras onboard with an offset angle from each other. Future work in this area can concentrate on a number of

different directions including:

- The current end-to-end system utilizes a very simple control law. Clearly more sophisticated approaches are possible. For example, at the moment the control twist velocity is chosen based on the G_+ / G_- values falling into wide bins. Clearly a more nuanced controller is possible.

- The current recognition system does not encode trail width in any manner. A parallel CNN system that estimated trail width could be used to provide useful information for the controller.

- It would be interesting to explore the end-to-end performance of the system using different trail type CNN on the "wrong" trail type. Recognition rates suggested that it might be possible to compress the class of classifiers. Is such a compression possible in terms of end-to-end performance?

- Can non-visual information be integrated within the approach. For example, could 3D LIDAR increase the accuracy and reliability of the system?

- Exploiting high-throughput processing units on board can extensively help to speed up the response of the robot to the incoming stream of images and this can help for having a faster autonomous robot for off-road trails. What is the best way of exploiting commodity graphics boards to speed up the implementation of CNN in the field?

- Knowing that the lighting conditions and season variation are problematic for a classification based algorithm which has been trained only on a dataset collected in a specific season, gathering a year-round dataset with lots of diversity in lightning and features, might help the algorithm to be more robust for unseen trails and regions.

Bibliography

[1] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 2818–2826.

[2] 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.

[3] (2016) The Defense Advanced Research Projects Agency (DARPA) website. [Online]. Available: http://www.darpa.mil.

[4] L. D. Jackel, E. Krotkov, M. Perschbacher, J. Pippine, and C. Sullivan, "The DARPA LAGR Program: Goals, challenges, methodology, and phase I results," *Journal of Field Robotics*, vol. 23, no. 11-12, pp. 945–973, 2006.

[5] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann *et al.*, "Stanley: The robot that won the DARPA Grand Challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[6] M. Bajracharya, A. Howard, L. H. Matthies, B. Tang, and M. Turmon, "Autonomous off-road navigation with end-to-end learning for the LAGR Program," *Journal of Field Robotics*, vol. 26, no. 1, pp. 3–25, 2009.

[7] R. Mohan, "Deep deconvolutional networks for scene parsing," *arXiv preprint arXiv:1411.4101*, 2014.

[8] 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.

[9] C. C. T. Mendes, V. Frémont, and D. F. Wolf, "Exploiting fully convolutional neural networks for fast road detection," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, 2016, pp. 3174–3179.

[10] G. L. Oliveira, W. Burgard, and T. Brox, "Efficient deep models for monocular road segmentation," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, South Korea, 2016, pp. 4885–4891.

[11] (2017) The Pioneer 3-at robot website. [Online]. Available: http://www.mobilerobots.com/ResearchRobots/P3AT.aspx.

[12] (2016) The The Robot Operating System website. [Online]. Available: https://www.ros.org.

[13] D. A. Pomerleau, "ALVINN, an autonomous land vehicle in a neural network," Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, Tech. Rep., 1989.

[14] H. Jeong, Y. Oh, J.-H. Park, B. Koo, and S. W. Lee, "Vision-based adaptive and recursive tracking of unpaved roads," *Pattern Recognition Letters*, vol. 23, no. 1, pp. 73–82, 2002.

[15] P. Moghadam, W. S. Wijesoma, and M. Moratuwage, "Towards a fully-autonomous vision-based vehicle navigation system in outdoor environments," in *Proceedings of the IEEE International Conference on Control Automation Robotics & Vision (ICARCV)*, Singapore, 2010, pp. 597–602.

[16] A. Deshpande, J. Rock, and D. Forsyth, "Learning large-scale automatic image colorization," in *Proceedings of the International Conference on Computer Vision*, Santiago, Chile, 2015, pp. 567–575.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[18] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, and G. Di Caro, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2016.

[19] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[20] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016.

[22] C. M. Bishop, "Pattern recognition," *Machine Learning*, vol. 128, pp. 1–58, 2006.

[23] S. Haykin and N. Network, "A comprehensive foundation," *Neural Networks*, vol. 2, no. 2004, p. 41, 2004.

[24] Y. Bengio, "Learning deep architectures for AI," *Foundations and trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[25] C. M. Bishop, *Neural networks for pattern recognition*. New York, NY: Oxford University Press, 1995.

[26] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. New York, NY: John Wiley & Sons, 2012.

[27] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks." in *Aistats*, vol. 9, 2010, pp. 249–256.

[28] G. E. Hinton, J. L. Mcclelland, and D. E. Rumelhart, "Distributed Representations, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations," Cambridge, MA, 1986.

[29] N. Gershenfeld, "An experimentalist's introduction to the observation of dynamical systems," *Directions in Chaos*, vol. 2, pp. 310–384, 1988.

[30] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[31] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, and T. Chen, "Recent advances in convolutional neural networks," *Pattern Recognition*, 2017.

[32] K. Jarrett, K. Kavukcuoglu, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Dhaka, Bangladesh, 2009, pp. 2146–2153.

[33] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, Paris, France, 2010, pp. 253–256.

[34] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," *Artificial Neural Networks (ICANN)*, pp. 92–101, 2010.

[35] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *In Proceedings of the European Conference on Computer Vision (ECCV)*. Zurich, Switzerland: Springer, 2014, pp. 818–833.

[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[37] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[38] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, Columbus, OH, 2014, pp. 1717–1724.

[39] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, Miami, FL, 2009, pp. 248–255.

[40] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

[41] S. Yan, J. Dong, Q. Chen, Z. Song, Y. Pan, W. Xia, Z. Huang, Y. Hua, and S. Shen, "Generalized hierarchical matching for sub-category aware object classification," in *Proceedings of the European Conference on Computer Vision (ECCV)*, vol. 5, no. 6, Florence, Italy, 2012.

[42] A. Waxman, J. Moigne, and B. Srinivasan, "Visual navigation of roadways," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, St. Louis, MO, 1985, pp. 862–867.

[43] D. Kuan, G. Phipps, and A. C. Hsueh, "Autonomous robotic vehicle road following," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 10, no. 5, pp. 648–658, 1988.

[44] S.-P. Liou and R. C. Jain, "Road following using vanishing points," *Journal of Computer Vision, Graphics, and Image Processing (CVGIP)*, vol. 39, no. 1, pp. 116–130, 1987.

[45] G. N. DeSouza and A. C. Kak, "Vision for mobile robot navigation: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 24, no. 2, pp. 237–267, 2002.

[46] A. B. Hillel, R. Lerner, D. Levi, and G. Raz, "Recent progress in road and lane detection: a survey," *Machine Vision and Applications*, vol. 25, no. 3, pp. 727–745, 2014.

[47] M. Buehler, K. Iagnemma, and S. Singh, *The 2005 DARPA Grand Challenge: The Great Robot Race*. Berlin, Germany: Springer Science & Business Media, 2007, vol. 36.

[48] S. Tsugawa, T. Yatabe, T. Hirose, and S. Matsumoto, "An automobile with artificial intelligence," in *Proceedings of the International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 1979, pp. 893–895.

[49] C. Thorpe, M. H. Hebert, T. Kanade, and S. A. Shafer, "Vision and navigation for the carnegie-mellon navlab," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 10, no. 3, pp. 362–373, 1988.

[50] P. Moghadam and J. F. Dong, "Road direction detection based on vanishing-point tracking," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, Vilalmoura-Algarve, Portugal, 2012, pp. 1553–1560.

[51] P. De Cristóforis, M. A. Nitsche, T. Krajník, and M. Mejail, "Real-time monocular image-based path detection," *Journal of Real-Time Image Processing*, vol. 11, no. 2, pp. 335–348, 2016.

[52] D. Lieb, A. Lookingbill, and S. Thrun, "Adaptive road following using self-supervised learning and reverse optical flow." in *Robotics: Science and Systems*, 2005, pp. 273–280.

[53] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, Providence, RI, 2012, pp. 3642–3649.

[54] A. von Reyher, A. Joos, and H. Winner, "A LIDAR-based approach for near range lane detection," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, Las Vegas, NV, 2005, pp. 147–152.

[55] S. Kammel and B. Pitzer, "Lidar-based lane marker detection and mapping," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, Eindhoven, Netherlands, 2008, pp. 1137–1142.

[56] T. Ogawa and K. Takagi, "Lane recognition using on-vehicle lidar," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, Tokyo, Japan, 2006, pp. 540–545.

[57] J. Hernández and B. Marcotegui, "Filtering of artifacts and pavement segmentation from mobile lidar data," in *Proceedings of the ISPRS Workshop Laserscanning*, Paris, France, 2009.

[58] L. B. Cremean and R. M. Murray, "Model-based estimation of off-highway road geometry using single-axis ladar and inertial sensing," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Orlando, FL, 2006, pp. 1661–1666.

[59] W. Zhang, "Lidar-based road and road-edge detection," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, La Jolla, CA, 2010, pp. 845–848.

[60] C. Rasmussen, "Combining laser range, color, and texture cues for autonomous road following," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, vol. 4, Washington DC, USA, 2002, pp. 4320–4325.

[61] M. Hoveidar-Sefid and M. Jenkin, "Autonomous trail following," in *Proceedings of the International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, Madrid, Spain, 2017.

[62] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[63] J. Fritsch, T. Kuhnl, and A. Geiger, "A new performance measure and evaluation benchmark for road detection algorithms," in *Proceedings of the IEEE International Conference on Intelligent Transportation Systems (ITSC)*, The Hague, Netherlands, 2013, pp. 1693–1700.

[64] (2016) The Google Map website. [Online]. Available: http://www.maps.google.com.

[65] (2016) The Bing Map website. [Online]. Available: https://www.bing.com/mapspreview.

[66] (2016) The Yahoo Map website. [Online]. Available: https://maps.yahoo.com.

[67] (2016) The OpenStreetMap project website. [Online]. Available: http://www.openstreetmap.org.

[68] (2016) The MapQuest website. [Online]. Available: https://www.mapquest.com.

[69] (2016) The Apple Map website. [Online]. Available: https://mapsconnect.apple.com.

[70] J. M. Mirats-Tur, C. Zinggerling, and A. Corominas-Murtra, "GIS map based mobile robot navigation in urban environments," in *Proceedings of the IEEE International Conference on Advanced Robotics (ICAR)*, 2009, pp. 1–6.

[71] G. Floros, B. van der Zander, and B. Leibe, "OpenStreetSLAM: Global vehicle localization using openstreetmaps," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, 2013, pp. 1054–1059.

[72] K. Irie and M. Tomono, "Localization and road boundary recognition in urban environments using digital street maps," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, MN, 2012, pp. 4493–4499.

[73] Z. Tao, P. Bonnifait, V. Fremont, and J. Ibanez-Guzman, "Mapping and localization using GPS, lane markings and proprioceptive sensors," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, 2013, pp. 406–412.

[74] T. Senlet and A. Elgammal, "A framework for global vehicle localization using stereo images and satellite and road maps," in *Proceedings of the IEEE International Conference on Computer Vision Workshops (ICCV)*, Barcelona, Spain, 2011, pp. 2034–2041.

[75] M. Hentschel and B. Wagner, "Autonomous robot navigation based on openstreetmap geodata," in *Proceedings of the IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Madeira Island, Portugal, 2010, pp. 1645–1650.

[76] E. Pereira, D. A. Lima, and A. C. Victorino, "Autonomous vehicle global navigation approach associating sensor based control and digital maps," in *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Bail, Indonesia, 2014, pp. 2404–2409.

[77] P. Ruchti, B. Steder, M. Ruhnke, and W. Burgard, "Localization on openstreetmap data using a 3d laser scanner," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, 2015, pp. 5260–5265.

[78] A. Mancini and P. Zingaretti, "Point to point navigation for people with mobility impairments," in *Proceedings of the IEEE International Conference on Mechatronic and Embedded Systems and Applications (MESA)*, Ancona, Italy, 2014, pp. 1–6.

[79] M. A. Brubaker, A. Geiger, and R. Urtasun, "Map-based probabilistic visual self-localization," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 38, no. 4, pp. 652–665, 2016.

[80] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2818–2826.

[81] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *arXiv preprint arXiv:1409.0575*, 2014.

[82] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, 2015, pp. 1–9.

[83] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning." in *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI)*, San Francisco, CA, 2017, pp. 4278–4284.

[84] (2016) The Kodak camera website. [Online]. Available: https://www.kodakpixpro.com.

[85] (2016) The OpenStreetMap project Wiki website. [Online]. Available: https://wiki.openstreetmap.org/wiki.

[86] (2016) The JOSM software website. [Online]. Available: https://josm.openstreetmap.de.

[87] (2017) The TensorFlow website. [Online]. Available: https://www.tensorflow.org.

[88] (2017) The Amazon Web Services website. [Online]. Available: aws.amazon.com.

[89] (2017) The NVIDIA website. [Online]. Available: https://www.nvidia.com.

[90] R. A. Young, R. M. Lesperance, and W. W. Meyer, "The Gaussian derivative model for spatial-temporal vision: I. Cortical model," *Spatial Vision*, vol. 14, no. 3, pp. 261–319, 2001.