TÉCNICO LISBOA

Generate a Birds Eye View from Fisheye Cameras using Generative Adversarial Networks

Ricardo Branco Lucas

Thesis to obtain the Master of Science Degree in

Mechanical Engineering

Supervisor: Prof. José Raul Carreira Azinheira

Examination Committee

Chairperson: Prof. Carlos Baptista Cardeira
Members of the Committee:
Prof. Alexandra Bento Moutinho
Prof. Susana Margarida da Silva Vieira

January 2021

Acknowledgements

I would like to thank Amer Mustajbasic for supervising my thesis work at Volvo Cars and for all the help and feedback throughout it. I also want to thank Daniel Larsson for overseeing my stay at Volvo Cars and for all the assistance, in particular during the data collection sessions. Additionally, I would like to thank Tom Bruls, author of [1], for taking the time to clarify details regarding the implementation of their paper. Finally, I want to thank José Raul Azinheira, for taking on the responsibility of being my master thesis supervisor at Instituto Superior Técnico.

Resumo

Métodos tradicionais para gerar uma *Bird's Eye View* (BEV) são fiáveis quando utilizados para superfícies planas, mas deixam de o ser quando utilizados para inclinações ou objetos salientes estão presentes. Esta dissertação tem como objetivo investigar diferentes métodos para gerar uma BEV, a partir de 4 câmaras *fisheye* montadas num veículo, utilizando Redes Adversárias Generativas e como melhorar sobre os métodos atuais. Duas abordagens diferentes com Redes Adversárias Generativas são apresentadas, juntamente com um procedimento de recolha e processamento de dados. Na primeira abordagem, modelos *state-of-the-art* (Pix2pixHD, CycleGAN, AttentionGAN) são utilizados para gerar a BEV. Na segunda, um modelo com múltiplas entradas é proposto, não só para gerar a BEV, mas também para estimar homografias corretivas. As abordagens tomadas são testadas utilizando dados reais e demonstram resultados promissores, sendo que a o modelo proposto puderá ser desenvolvido em estudos futuros. Não foi possivél gerar uma BEV, com atuais modelos de Redes Adversárias Generativas, apenas a partir de imagens *fisheye*. No entanto, sem aumentar significativamente a complexidade destes métodos, abordagens semelhantes apresentam resultados promissores. Em particular, módulos *Spatial Transformer* demonstram um alto potencial em complementar as Redes Adversárias Generativas para a geração da BEV.

Palavras-chave: *Bird's Eye View*, Redes Adversárias Generativas, Homografia, *Spatial Transformer Networks*, Homografia.

Abstract

Traditional methods for generating a Bird's Eye View (BEV) are accurate for flat surfaces, but errors are introduced when slopes or protruding objects are present. This thesis aims to investigate different methods using Generative Adversarial Networks (GAN) to generate a BEV from 4 vehicle-mounted fisheye cameras, and to improve on traditional methods. I present two different model approaches, and a data collection and processing procedure. In the first approach, state-of-the-art models (Pix2pixHD, CycleGAN, AttentionGAN) are used to generate the BEV images. In the second, I propose a multi-input model, not only to generate the BEV images, but also to estimate corrected homography matrices. It was not possible to generate a BEV with current state-of-the-art Generative Adversarial Networks, given only fisheye images. Nonetheless, it is shown that without greatly increasing the methods' complexity, similar approaches yield promising results. In particular, Spatial Transformer modules demonstrate a strong potential in aiding a GAN to learn a correct mapping to the BEV.

Keywords: Bird's Eye View, Generative Adversarial Networks, Spatial Transformer Networks, Homography estimation.

Contents

List of Tables

List of Figures

Nomenclature

Acronyms

ANN	Artificial Neural Network
BEV	Bird's Eye View
CNN	Convolutional Neural Network
FID	Frechet-Inception Distance
GAN	Generative Adversarial Network
KID	Kernel-Inception Distance
MLP	Multi-Layer Perceptron
ST	Spatial Transformer

Other Symbols

γ	Angle relative to vertical axis
λ	Eigenvalue
$\mathbb{E}$	Expected Value
$\mathcal{L}$	Loss function
τ	Network parameters
θ	Geometrical transformation parameters
a	Horizontal pixel coordinate
b	Vertical pixel coordinate
C	Midpoint
D	Discriminator network
F	Generator network with inverse mapping
G	Generator network

k	Empirically determined constant
l	Length between points
M	Image gradients matrix
n	Number of intermediate layers
P	Data distribution
R	Corner detection score
t	Time
u	Horizontal pixel gradient
v	Vertical pixel gradient
w	Weight
x	Ground truth/Target image
y	Conditional data/Label or input image
z	Noise

Chapter 1

Introduction

Increasing concern with safety in mobility, together with costumers' interest in connectivity [2], has been shifting automotive manufacturers' efforts towards research and development in information technologies. As a result, many of them have led the field in intelligent technology solutions for safe mobility and ease of driving, in particular in autonomous driving.

Autonomous systems in vehicles need to accurately perceive and understand their surrounding environment in order to perform safe and efficient driving. Cameras are an inexpensive and popular sensor choice for these systems, and if paired with computer vision techniques, they can provide a great amount of information about the environment. Among the different types of cameras, fisheye cameras stand out for achieving high angles of view. This makes them ideal to capture the close proximity environment. By mounting four 180 degrees wide fisheye cameras on each side of the vehicle, it is easy to gain 360 degrees coverage of the surrounding environment, with some overlapping areas between adjacent cameras.

Although a camera can provide information in the 2-dimensional image plane, it lacks information about the real-world coordinates. By assuming a flat earth approximation, a perspective transform is often applied to project image pixels to the ground plane. The method of applying this perspective transform is commonly referred to as Inverse Perspective Mapping (IPM) [3]. And this results in a top-down view, also known as Bird's Eye View (BEV). This resulting projection provides information on the real-world coordinates of the elements on the ground plane. As the IPM method assumes a flat earth approximation, the objects protruding out of the ground plane, slopes and irregularities in the road surface will be incorrectly mapped to the BEV image during the IPM projection. Additionally, when using the IPM for the generation of a BEV in multi-camera systems other issues also arise, such as misalignments between the different views, mostly due to accuracy errors in camera calibration.

BEV images can be utilised within tasks such as lane detection [4], road marking detection [5], free space computation [6], and path planning [7]. Since the relevant elements for these tasks are mostly present in the ground plane, the IPM approximation is usually sufficient enough to be used in these applications. An accurate mapping to the BEV could not only improve the performance and reliability of the previously mentioned tasks, but also allow tasks where objects like other vehicles, obstacles and pedestrians are accurately detected and located in the real-world coordinates [8].

With the advancement of deep learning techniques, Generative Adversarial Networks (GANs) [9] have been able to learn the mapping between images of two different domains. By exploring similar methods for the generation of the BEV from 4 fisheye camera images, a more accurate mapping could be achieved.

Several works have taken a geometry-based approach to the generation of the BEV. In [10], [11] and [12] the authors proposed methods to align and stitch the different views in multi-camera systems. In [13], data from a laser range finder is fused with images from the cameras, so that the IPM is not computed in the regions where obstacles are present. And in [14], a mono visual odometry algorithm is used to obtain the vehicle motion, in order to correct the IPM.

In the last few years many works emerged on Image Generation and Translation with the rise of GANs [15]. Some of the most relevant being the ones on conditional GANs [16], on image-to-image translation [17] and on unsupersivioned GANs [18]. With the latest works employing attention modules that allow for the translation of images that require holistic and large shape changes [19], [20], but are still restricted to perform aligned appearance transformations. BridgeGAN [21] uses a GAN in conjunction with the IPM to bridge the large gap between the frontal view and the BEV.

In [22], the authors proposed a Spatial Transformer module that transforms the input images to improve performance on classification tasks. In [23] and [24] similar ideas to the Spatial Transformer were used to perform novel view synthesis. In [25], the authors use estimations of the depth and normals of the images to predict homographies. The work presented in [1] employs Spatial Transformer modules together with a GAN model to generate a BEV from a single frontal view, resulting in a higher quality BEV compared with the one generated solely through IPM.

The works [21] and [1] are the closest to ours, however these only generate the BEV for a single view. As far as I am aware, our work is the first to use multi-input GAN models to generate a single BEV from 4 different views.

1.1 Objectives

The main objective of this thesis is to investigate different methods using Generative Adversarial Networks to generate a Bird's Eye View from 4 vehicle-mounted fisheye cameras, and evaluate the capabilities of these networks to learn the correct mappings. In particular, I aim to generate a BEV image where all objects are correctly represented without distortions, and the stitching between the different fisheye images is seamless. This thesis also aims to find an efficient data collection procedure and evaluate which data is best suited for learning.

Furthermore, with this work I aim to find methods that reduce the need for time-consuming tasks, such as the estimation of parameters necessary to compute the Inverse Perspective Mapping, the alignment and the stitching of the different views.

1.2 Scope and Limitations

The generation of a BEV image is a problem which can be approached with many different methods or combination of methods. This thesis is limited to the exploration of methods that make use of neural networks, in particular GANs.

The collection of the data required to train GAN models is part of the work carried out during the course of this thesis. Due to time and budget constraints, the collected data has limitations in terms of quality and scenario variation. The scenarios covered by the data are limited to parking lots during daytime, with clear weather.

This thesis studies different GAN models and variations in the architecture and building blocks of the models. Due to the training time of the models, it is not part of this thesis to perform hyperparameter tuning on all models. Models are also limited to use a maximum of 24GB on the available NVIDIA Titan RTX GPU and this also restricts the size of the model and of the input images.

All the work was conducted at Volvo Cars in Lund, Sweden, within the autonomous driving department.

1.3 Thesis Outline

This thesis is divided into five chapters:

- The first chapter introduces the problem, the motivations and the objectives. It also includes an overview of related work to the generation of BEV images.
- The second chapter acts as a foundation for the reader, and provides the technical and theoretical knowledge of the concepts within Computer Vision and Neural Networks, which are used in this thesis.
- The third chapter covers the approaches taken in this thesis. It explains the data collection and processing process used to build the necessary datasets, as well as the reasons and details of the chosen and built neural network models, and steps taken to reach the objectives.
- The fourth chapter presents the results produced by the neural network models, which are analysed and discussed.
- The fifth chapter presents the conclusion to the thesis and possible future work.

Chapter 2

Background on Computer Vision and Neural Networks

To reach the objectives of this thesis, several concepts from Computer Vision and Machine Learning are used. In this chapter we give a brief overview of these concepts, starting with the ones taken from classic Computer Vision, followed with an introduction of the concepts of Neural Networks, and then with the Machine Learning frameworks more specific to this thesis.

2.1 Computer Vision

2.1.1 Optical Flow and The Lucas–Kanade Method

Optical flow is "the apparent motion of brightness patterns observed when a camera is moving relative to the objects being imaged" [26]. To estimate motion, first we need to define the brightness constancy constraint, Eq. 2.1. For a pixel in a frame that has intensity I at location (a, b, t), and for the next frame moves by distance $(\Delta a, \Delta b)$ and time Δt, to a new location $(a + \Delta a, b + \Delta b, t + \Delta t)$, with the same intensity, we can write:

$$I(a, b, t) = I(a + \Delta a, b + \Delta b, t + \Delta t) \tag{2.1}$$

By assuming the movement to be small, we can linearise the right side of Eq. 2.1 using Taylor series:

$$I(a + \Delta a, b + \Delta b, t + \Delta t) = I(a, b, t) + \frac{\delta I}{\delta a}\Delta a + \frac{\delta I}{\delta b}\Delta b + \frac{\delta I}{\delta t}\Delta t \tag{2.2}$$

and by substitution in Eq. 2.1, we get:

$$\frac{\delta I}{\delta a}\Delta a + \frac{\delta I}{\delta b}\Delta b + \frac{\delta I}{\delta t}\Delta t = 0 \tag{2.3}$$

and dividing by Δt :

$$\frac{\delta I}{\delta a}\frac{\Delta a}{\Delta t} + \frac{\delta I}{\delta b}\frac{\Delta b}{\Delta t} + \frac{\delta I}{\delta t}\frac{\Delta t}{\Delta t} = 0 \tag{2.4}$$

resulting in the optical flow equation:

$$\frac{\delta I}{\delta a}u + \frac{\delta I}{\delta b}v + \frac{\delta I}{\delta t} = 0 \tag{2.5}$$

$$I_a u + I_b v + I_t = 0 \tag{2.6}$$

where u and v are the a and b components of the velocity or optical flow, respectively. For simplicity of representation, we also present the partial derivatives of the intensity I with respect to position a, b and time t as (I_a, I_b, I_t).

In Eq. 2.6 we can calculate I_a and I_b, as they are image gradients. Similarly I_t is the gradient along time. But u and v are unknown. Therefore, this equation is not a solvable equation, as it has these two unknowns, u and v. In order to solve it, some assumptions have to be made:

- The pixel intensities of an object do not change between consecutive frames.
- Neighbouring pixels have similar motion.

Taking into account the previous assumptions, the Lucas-Kanade method [27] takes a 3x3 patch around the pixel, meaning that we can write an optical flow equation for each of these 9 points, resulting in the following system in matrix form:

$$Ds = E \tag{2.7}$$

where:

$$D = \begin{bmatrix} I_{a_1} & I_{b_1} \\ I_{a_2} & I_{b_2} \\ \vdots & \vdots \\ I_{a_9} & I_{b_9} \end{bmatrix}, \quad s = \begin{bmatrix} u \\ v \end{bmatrix}, \quad E = \begin{bmatrix} I_{t_1} \\ I_{t_2} \\ \vdots \\ I_{t_9} \end{bmatrix}$$

However, this is a system with 9 equations and two unknown variables, meaning it is over-determined. Therefore, the Lucas–Kanade method obtains a compromise solution by using the least squares principle. Specifically, it solves the following 2x2 system:

$$D^T Ds = D^T E \quad <=> \quad s = (D^T D)^{-1} D^T E \tag{2.8}$$

of which the final solution is:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i^9 I_{a_i}^2 & \sum_i^9 I_{a_i} I_{b_i} \\ \sum_i^9 I_{a_i} I_{b_i} & \sum_i^9 I_{b_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i^9 I_{a_i} I_{t_i} \\ -\sum_i^9 I_{b_i} I_{t_i} \end{bmatrix} \tag{2.9}$$

2.1.2 Corner Detection

Corners are regions with at least two different directions in the image gradients. When choosing tracking points to use with the Lucas–Kanade method, it is beneficial to choose corners, as these avoid the aperture problem [28], which can lead to unreliable tracking. To tackle this and several other problems, methods to find corners in images were developed.

In [29], an early attempt to find these corners is presented: the Harris Corner Detector. Here, a sliding window approach is taken and a score function, Eq. 2.1.2, is used to determine if the window, W, location corresponds to a corner.

$$R = det(M) - k(trace(M))^2 \tag{2.10}$$

where,

$$M = \sum_{(a,b)\in W} \begin{bmatrix} I_a^2 & I_a I_b \\ I_a I_b & I_b^2 \end{bmatrix} \tag{2.11}$$

and where k is an empirically determined constant; $k \in [0.04, 0.06]$. For the full derivation of M, please refer to [29].

If the score, R, is greater than 0 and above a defined threshold, then the region corresponds to a corner.

In [30], the Shi-Tomasi Corner Detector is introduced, here the authors altered the score function from the Harris Corner Detector, , to become:

$$R = min(\lambda_1, \lambda_2) \tag{2.12}$$

In Eq. 2.12, λ_1 and λ_2 are the eigenvalues of M. As in the Harris Corner Detector, if the score R, is above a defined minimum threshold, then the region is classified as a corner.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are one of the most popular machine learning methods. ANNs play a key role in many complex tasks, such as image recognition, segmentation and classification, as well as approximating functions.

ANNs are said to have been inspired by biological brains in which a network of millions of neurons create intricate signal patterns that process information. Here an artificial neuron is modelled to receive several inputs, in the form of a real number, process them and create an output to pass on to other neurons. And when the information has passed by enough neurons, the output from the network will have a meaningful signal. For ANNs, it is often the case that more neurons can solve more complex tasks, but it is also heavily dependent on how these neurons are put together. In this section we will give a brief overview of the underlying theory behind how an ANN operates and is optimised. For a more comprehensive explanation on Neural Networks, please refer to [31].

2.2.1 The Artificial Neuron

The artificial neuron is, at it core, a function that computes the dot product between an input vector x and the weight vector w of the neuron, resulting in a weighted sum. A neuron has the output y, which is computed in the following way,

$$y = \sigma(x \cdot w + b) \tag{2.13}$$

The weight vector w is what gives the neuron its computational characteristic and can emphasise different dimensions in x when computing the weighted sum. Also, a bias term b is added to the weighted sum. This weighted sum is then passed through a non-linear function σ, known as an activation function, to produce the output. Historically, a common choice of activation function is the sigmoid function, since it takes a real-valued input (the sum) and bounds it to a range between 0 and 1.

2.2.2 The Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP) is the simplest type of ANN. MLPs are modelled as collections of artificial neurons that are connected in a graph, where the outputs of some neurons can become inputs to other neurons. MLP models are often organized into distinct layers of neurons, the first layer is called the input layer and the last layer is called the output layer, all the layers in between are said to be hidden layers. The most common layer type is the fully-connected layer in which neurons between two adjacent layers are fully pairwise connected, but neurons within a single layer share no connections.

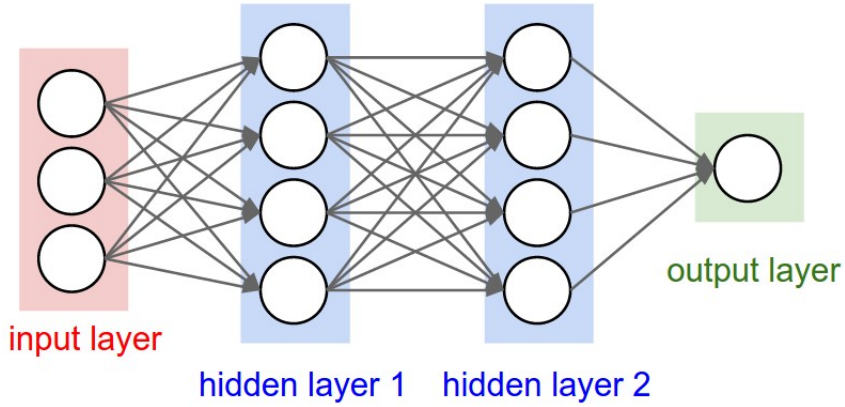

Figure 2.1: A MLP with two hidden, fully-connected layers of 4 artificial neurons each.

The MLP network architecture can carry out more complicated calculations than a single neuron, and can in fact represent any function on a compact set [32].

2.2.3 Learning

In order to the output of an ANN have any meaning, just as biological brains needs to train on specific tasks to become proficient on them, an ANN needs to train on data. Learning occurs by updating the weights and biases of the network after each piece of data is processed, to improve the accuracy of the result.

To be able to improve the performance of a network, one first needs to have a way to evaluate it. This is done by a loss function, that for each data point compares the output of the network with a ground truth value, to calculate a loss value. Now the goal is to find the weights that minimise the loss. To do this, backpropagation is done, it calculates the gradients of the loss function in the network, by recursively applying the chain rule all the way to the inputs. Then, the weights can be updated via stochastic gradient descent.

2.2.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are very similar to the typical ANNs from the previous section. They are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and follows it with an activation function. And they also have a loss function on the last layer. On the other hand, CNN architectures make the explicit assumption that the inputs are images, which allows to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

2.2.4.1 Convolutional Layer

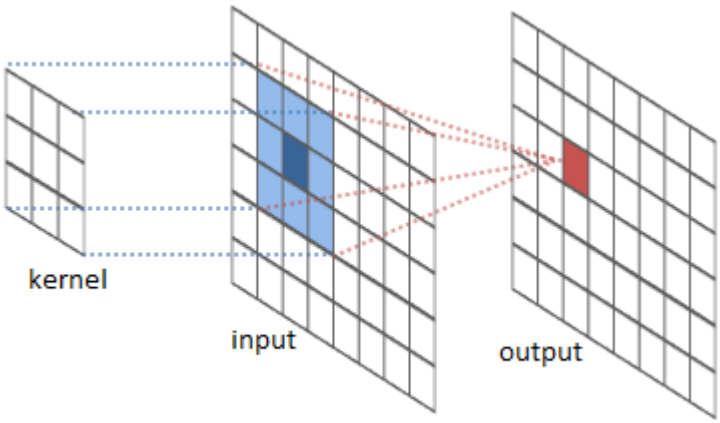

Figure 2.2: Visualization of the convolutional operation, where the output results from the dot products between the kernel/filter and the input.

The convolutional layer's parameters consist of a set of learnable filters (or kernels). Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a CNN might have size 5x5x3 (i.e. 5 pixels width and height, and 3 because images have depth 3, the colour channels). During the forward pass, each filter is convoluted across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As the filter is slided over the width and height of the input volume, it will produce a 2-dimensional activation map (also referred to as feature map) that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature, such as an edge of some orientation or a blotch of some colour on the first layer. The number of filters used for an input volume, will determine the number of feature maps, or the depth, in the output volume.

2.2.5 Generative Adversarial Networks

Generative Adversarial Networks (GANs), [9], are models composed of two networks, a generator and a discriminator, that contest against each other, in order to learn to generate new data.

The purpose of GANs is to be able to generate new samples from a given, complex and high dimensional, training distribution. A GAN achieves this by sampling from a simple distribution, e.g. Gaussian noise, and learning a transformation to the training data distribution, by using a neural network.

The purpose of the generator is to approximates a mapping to the training data distribution P_{data}, and the discriminator is to estimate the probability that a sample came from the training data rather than the generator. Fig. 2.3 represents the typical structure of a GAN.

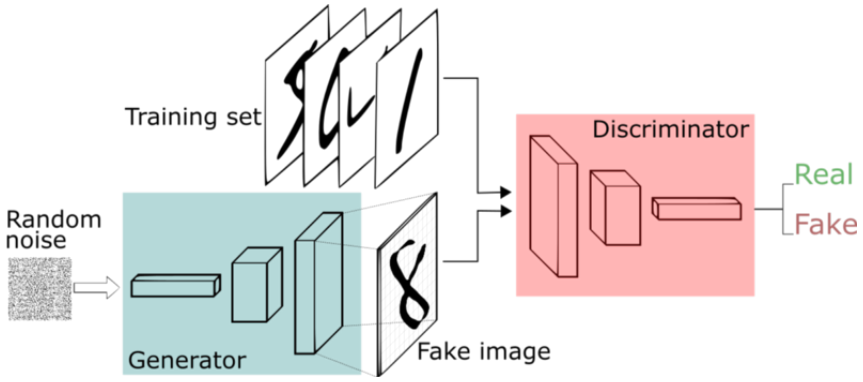

Figure 2.3: Generative Adversarial Network framework. Figure taken from [33].

To learn the generator's, G, distribution, a mapping function $G(z;\tau_g)$, between a prior noise distribution, P_z, and a training data distribution, P_{data}, is built, where G is a differentiable function represented by a neural network (which can be a MLP or a CNN) with parameters τ_g. The discriminator, $D(x;\tau_d)$, also represented by a neural network with parameters τ_d, outputs a single scalar representing the probability that it's input came from training data, P_{data}, rather than from generated data, P_g. Here, z represents a sample from the noise distribution and x a sample from the training data, which we will also refer to as ground truth or target image.

The Generator and Discriminator are both trained simultaneously using backpropagation: the parameters of G are updated to minimise $log(1 - D(G(z))$ and the parameters of D are updated to maximise $logD(x)$ and $log(1 - D(G(z))$. This can be defined as a two-player min-max game with loss function $\mathcal{L}(D, G)$:

$$\min_G \max_D \mathcal{L}(D,G) = \mathbb{E}_{x\sim P_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim P_z(z)}[log(1 - logD(G(z)))] \tag{2.14}$$

Since noise is the only input to the generator, it is not possible to control the modes of the generated data. As an answer to this, Conditional GANs [16] were introduced. Here the model is conditioned on additional data, y, (for example on digits class labels of the MNIST dataset) which allow to control the generative process to the desired modes. This conditioning is done in the generator G by combining the prior input noise, z, with y in a joint hidden representation, which is composed by, again, a neural network (commonly referred to as encoder network), and in the discriminator by inputting y and x concatenated. The loss function for the

conditional GAN is:

$$\min_G \max_D \mathcal{L}(D,G) = \mathbb{E}_{x\sim P_{data}(x)}[logD(x|y)] + \mathbb{E}_{z\sim P_z(z)}[log(1-logD(G(z|y)))] \tag{2.15}$$

More recent works on Image-to-image translation tasks [17], where y is an image, have completely removed the prior noise input, and instead relied on other regularisation methods, like dropout [34]. Here, the translation is considered to be done between two domains, X and Y. In other words, the training dataset is given as a set of pairs of corresponding images $\{(x_i, y_i)\}$, where $(x_i \in X), (y_i \in Y)$, and for any pair, x_i and y_i correspond to each other; for example the following pair: an image of a landscape during summer and an image of the same landscape, with the same framing, during winter. Here, we define y_i as the label image and x_i as the target image. The loss function in this case becomes:

$$\min_G \max_D \mathcal{L}(D,G) = \mathcal{L}_{GAN}(G,D) \tag{2.16}$$

where:

$$\mathcal{L}_{GAN}(G,D) = \mathbb{E}_{(x,y)\sim P_{data}(x,y)}[logD(y,x)] + \mathbb{E}_{y\sim P_{data}(y)}[log(1-logD(y,G(y)))] \tag{2.17}$$

2.2.6 Unsupervised Generative Adversarial Networks

Image-to-image translation problems using conditional GANs require paired training data, but this is very expensive to collect and sometimes even impossible. Therefore, an unsupervised approach to GANs is necessary.

In [18], the authors proposed a new model that uses two generators, two discriminators and a cycle consistency loss, which is referred to as CycleGAN. In this model, the two generators, G and F, learn the translations between the two domains X and Y:

$$G : X \rightarrow Y$$

$$F : Y \rightarrow X$$

And the two discriminators, D_X and D_Y, learn to evaluate the generated fake samples, respectively for each domain.

The reason to translate both ways between the domains is to create a cycle: each sample x is translated into the domain Y and then back to X, where the result should be a reconstruction of the original image, i.e., $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$. From here, the researchers introduced the cycle consistency loss, that is calculated with the L1 norm:

$$\mathcal{L}_{Cycle}(G,F) = \mathbb{E}_{x\sim P_{data}(x)}[||F(G(x)) - x||] + \mathbb{E}_{y\sim P_{data}(y)}[||G(F(y)) - y||] \tag{2.18}$$

Which, is then combined with the adversarial losses to form the full objective:

$$min_G max_D \mathcal{L}(G,F,D_X,D_Y) = \mathcal{L}_{GAN}(G,D_Y) + \mathcal{L}_{GAN}(F,D_X) + \mathcal{L}_{Cycle}(G,F) \tag{2.19}$$

2.2.7 Spatial Transformer Networks

Spatial transformer modules, introduced in the paper "Spatial Transformer Networks" [22], are differentiable modules that can be added into existing CNN architectures and are able to geometrically transform an input into a pose that will ease recognition and classification further in another network.

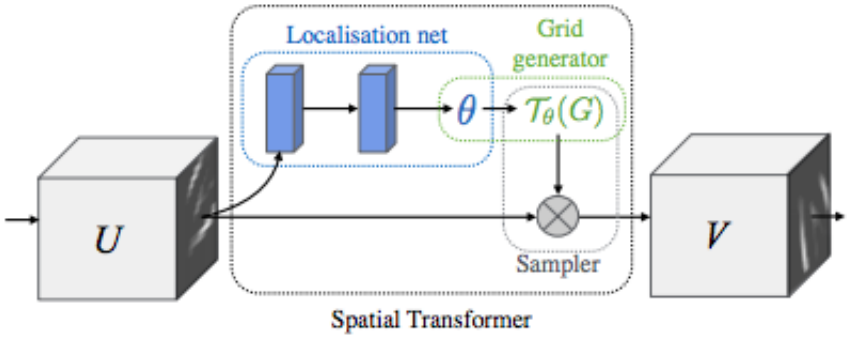

Figure 2.4: Architecture of a spatial transformer module. The input feature map U is passed through the spatial transformer, producing the warped output feature map V. Figure taken from [1].

A spatial transformer module is composed of a localisation network, a grid generator and a sampler, as seen in Fig. 2.4. The task of the localisation network is to take a tensor of feature maps and through a CNN architecture regress the transformation parameters θ. The task of the grid generator is to take the predicted transformation parameters θ and create a new sampling grid, from which the sampler will produce a new wrapped output tensor from the input tensor.

Chapter 3

Proposed Methods

Most CNN models take only one image as input. In order to combine images from multiple cameras mounted on a vehicle, two procedures can be taken: use as input the 4 camera images concatenated along their channel dimension; or combine the 4 images in a 2 by 2 grid, as to create a single image. However, for the task at hand, this would result in spatial inconsistency between the input and the output images, because of the way convolutional layers operate (information in particular locations of the input is mapped to approximately the same location of the output). Therefore, other solutions had to be found. The first solution was to use popular GAN models, which are all single-input, and use as input a pre-processed BEV image from the 4 camera images by using the IPM technique. The second was to create a multi-input GAN model, with integrated Spatial Transformer modules to ensure spatial consistency. In this chapter, we will present these two different GAN-based approaches. But before, the data collection and processing procedures used to create a dataset are explained in detail.

3.1 Dataset Creation

In order to train the GAN models to generate the BEV images, a dataset was necessary. It needed to contain images from the 4 fisheye cameras in one domain and on the other domain the ground truth, which needed to be real BEV images. No publicly available dataset meets these requirements, therefore the work developed in this thesis also encompasses the creation of this required dataset.

3.1.1 Data Collection

The first step in creating a dataset from scratch is the data collection. In order to collect the necessary images we used a Volvo XC90 test car and a DJI Mavic drone. The Volvo test car was equipped with four fisheye cameras, one in the front badge, one in each of the side-mirrors and one in the tailgate, meaning that each camera is pointing in a different direction and 360 degrees coverage can be achieved. The DJI drone was equipped with a camera mounted on a stabilisation gimbal, which was set to point directly down.

The test car fisheye cameras synchronously recorded images at a resolution of 1280 by 1080 pixels and at an irregular frame rate of 3 to 4 frames per second (fps); the field of view of the fisheyes is approximately 180

degrees. While the drone was set to record video at a resolution of 4096 by 2160 pixels and at a fixed frame rate of 23.98 fps.

The recording procedure involved driving the test car at relatively slow speeds i.e., less than 30 km/h, on various parking lots, and having the drone following the car directly above, with the camera pointed straight down, at an altitude relative to the ground of around 50 meters.

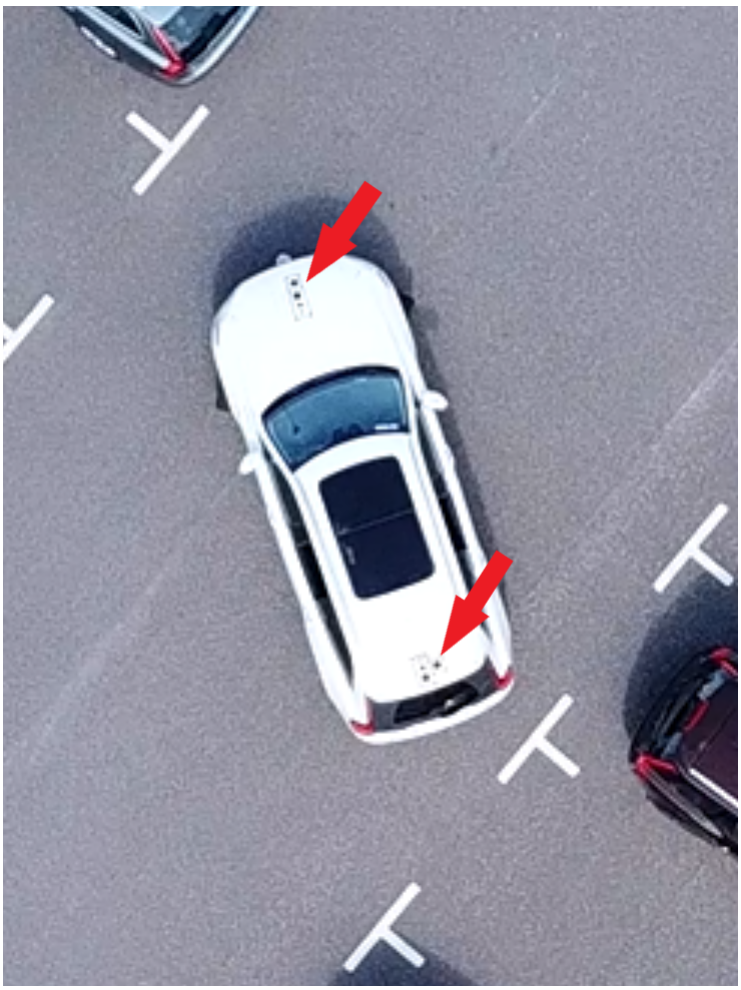

Figure 3.1: Image of the Volvo test car with tracking markers (pointed out by the red arrows), placed in the front and back of the vehicle. Image taken with the DJI drone during a data collection session.

The DJI Mavic drone had to be controlled manually, which meant that it wasn't always directly over the test car nor at the same altitude. That lead to the need of rectifying the drone frames in a later processing stage. To help with processing, markers for tracking were placed on the test car, as seen in Fig. 3.1.

3.1.2 Data Processing

Due to the drone related limitations explained in section 3.1.1, the resulting frames from the drone video had severe variations in the orientation, position and scale relative to the test car. In order to use these images to train GAN models, these parameters had to be normalised, and rectification of the drone images was necessary. The goal of this processing was: to correct each frame to centre the test car in the frame; align the test car

symmetry line with the vertical axis of the frame; keep the scale of the image fixed in relation to the test car; and to synchronise the frames with the images from the test car fisheye cameras, since they ran at different and unstable frame rates. To achieve this, an algorithm using the OpenCV library, was developed.

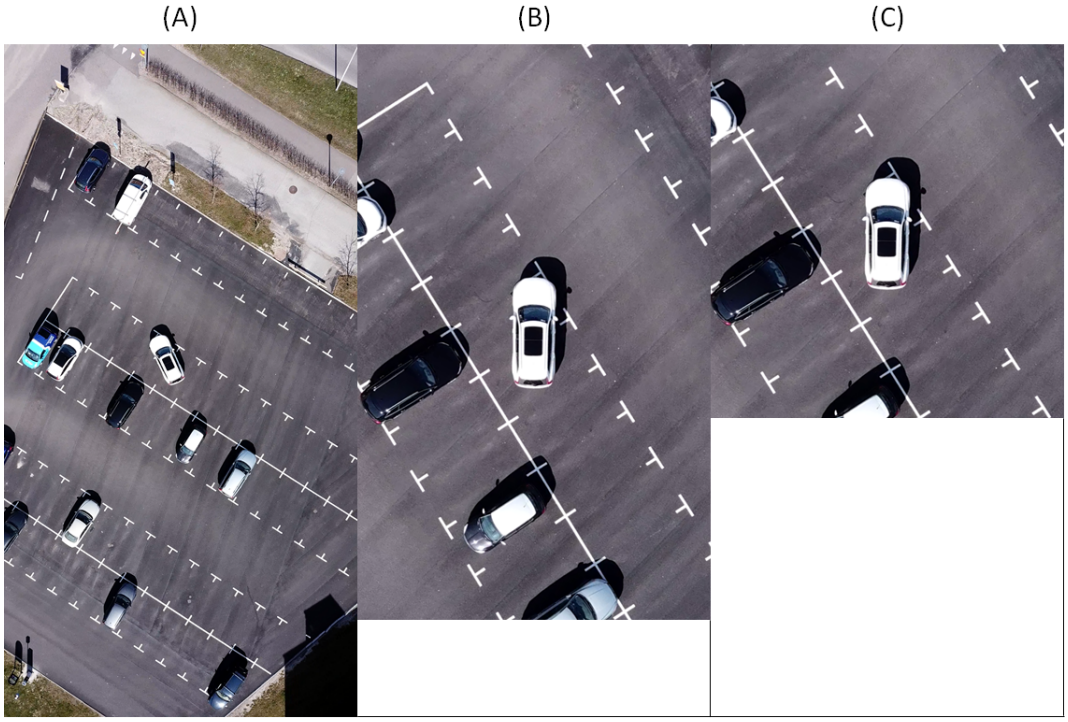

Figure 3.2: Images of the same frame after different processing steps. (A) - Original frame; (B) - Frame after alignment; (C)- Frame after cropping.

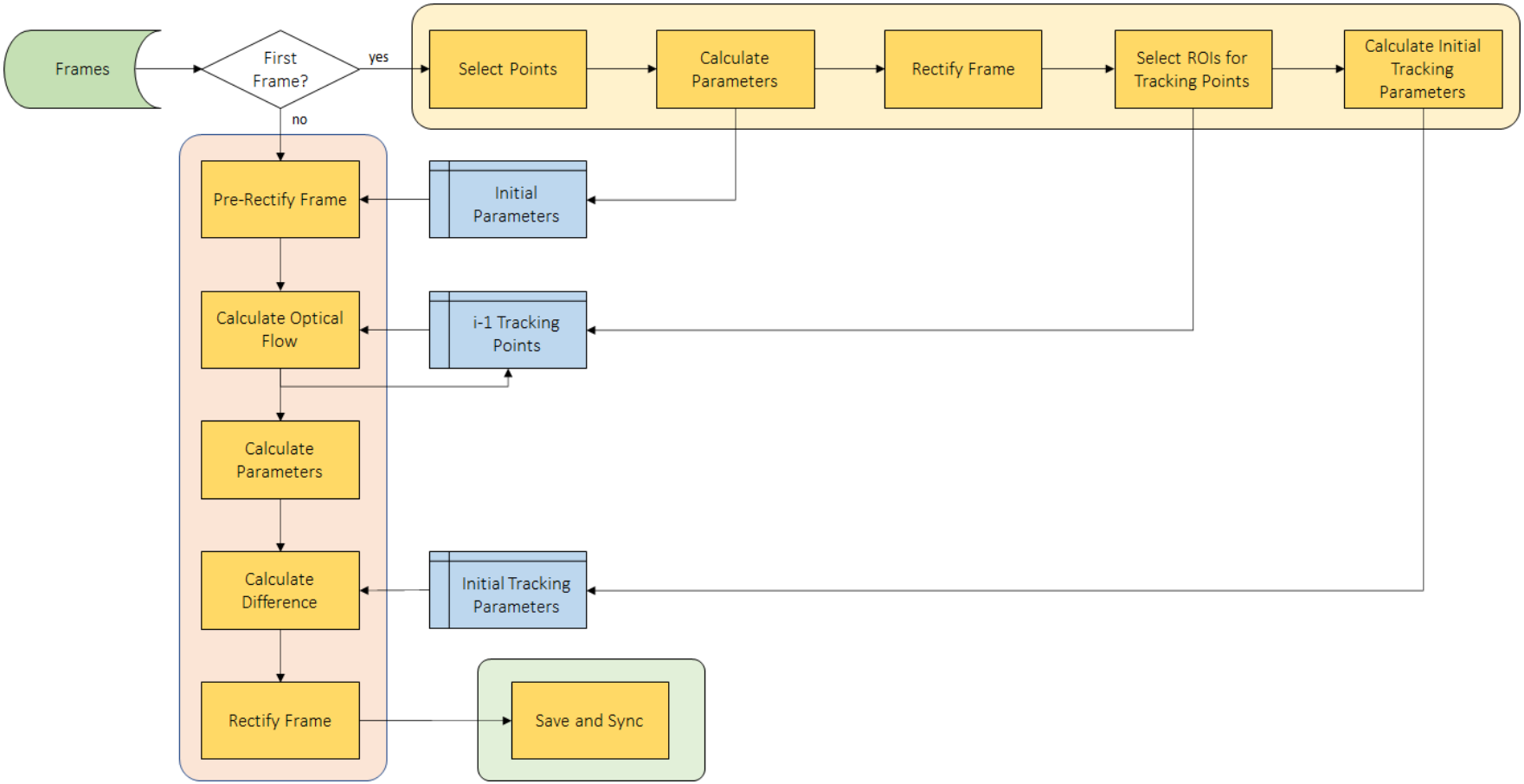

Figure 3.3: Algorithm flowchart for rectifying the frames from the drone video.

The algorithm to rectify the drone images and synchronise them with the test car images can be portioned in 3 main stages: parameter extraction by manual point and Region of Interest (ROI) selection; tracking with optical flow and rectifying; synchronising and saving. In Fig. 3.3 an overview of the algorithm is presented.

The first frame that is fed to the algorithm will follow the first stage, parameter extraction by manual point and ROI selection. Here the user has to identify and manually select two points on the frame that correspond to the test car extremities that belong to its symmetry line. These points will then be fed into the process Calculate Parameters, which will output the angle γ relative to the vertical axis, the length (in pixels) l and the midpoint pixel coordinates C between the points. In this process, given two points in the image coordinate system, $A(a_A,\, b_A)$ and $B(a_B,\, b_B)$, that form the line segment AB, represented in Fig. 3.4, the parameters γ, l and C can be easily calculated:

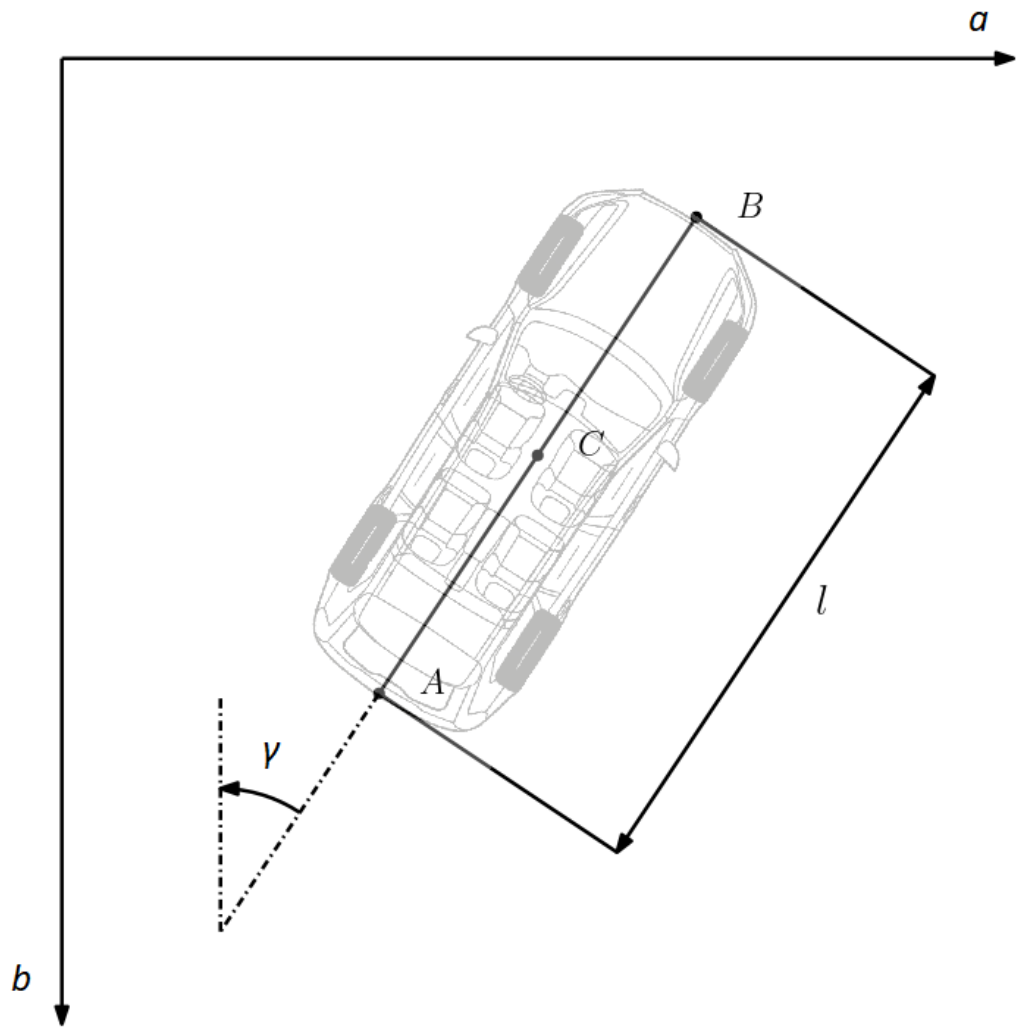

Figure 3.4: Definition of the image coordinate system, representation of the points A and B, and parameters γ, l and C in relation to the test car.

$$l = \sqrt{(a_B - a_A)^2 + (b_B - b_A)^2} \tag{3.1}$$

$$C = (\frac{a_A + a_B}{2},\ \frac{b_A + b_B}{2}) \tag{3.2}$$

$$(a_B,\ b_B) = (a_A + l\sin(\gamma),\ b_A - l\cos(\gamma)) \tag{3.3}$$

$$\gamma = \arctan2(\frac{a_B - a_A}{b_A - b_B}) \tag{3.4}$$

To these first calculated parameters we will refer as Initial Parameters and they will serve to rectify the first frame and to pre-rectify all of the following frames. This means that now only the changes from this first rectified frame need to be tracked, and that can be done automatically.

In order to track the changes in orientation, position and scale of the test car relative to the first frame, a new set of good points to track needs to be selected. Here the user will not select the points, but instead will define two ROIs. The ShiTomasi Corner Detection algorithm will ran on the ROIs and select the best points for tracking. These points will usually be on the markers placed on the test car, but can also be any other good feature to track on the test car when markers are not available. The coordinates of these new tracking points are used to calculate the Initial Tracking Parameters, from which, changes in orientation, position and scale in new frames can be measured and rectified accordingly.

After the first frame, all the frames will follow the procedures in stage two. Here a frame will firstly be rectified according to the Initial Parameters. Then Optical Flow is calculated on the current frame with the coordinates from the previous set of tracking points (for the second frame will be the ones previously selected by the ShiTomasi Corner Detection algorithm), and output a new set of tracking points. From this new set, new parameters are calculated and compared with the Initial Tracking Parameters, and then rectification of the frame can be completed.

At stage three the drone frames are matched with a set of 4 fisheye camera images that correspond to the same scene. Here, we will need to resort to timestamps. Each set of images from the fisheye cameras has a timestamp associated with it. While the frames from the drone don't have timestamps, it is possible to calculate them. For this, we first need to find the timestamp of a single frame, this is possible by manually matching it with a set of images from the fisheye cameras. From here, and knowing the the frame rate of the drone camera, it is possible to calculate the timestamp for all the other frames.

Now, having the timestamp of every recorded frame and image, and knowing that the drone camera has a much higher frame rate, we will use all the recorded fisheye images. For each fisheye images set, the drone image that is closest to it, is rectified, paired and saved. To measure how close an image is to another, we keep an ordered list of the timestamps of the fisheye images sets. When the timeframe from the drone frame is greater or equal to the first timeframe from the fisheyes list, the frame is saved, and the next timeframe from the fisheyes list is loaded. If the timeframe from the frame is lower, it is simply discarded.

For use in different experiments, the images from the test car fisheye cameras were pre-processed into two more sets of different views, which we will refer to as: Undistorted and Classic BEV. In the Undistorted views the distortion caused by the fisheye camera lenses is removed and in the Classic BEV view the images from each individual camera are projected onto the ground plane, stitched together and blended to form a

single BEV image. This pre-processing is outside of the scope of this thesis, as these processes were already developed and implemented by Volvo Cars.

In a later stage, the saved frames from the drone are cropped, Fig. 3.2 (C). This is done to match the same ground plane area as the Classic BEV, therefore, both are aligned, we will refer to this view simply as Drone view. Despite our best efforts to align both domains, and achieving very reasonable results, the alignment will never be perfect, as seen in Fig. 3.5, without resorting to other methods during recording.

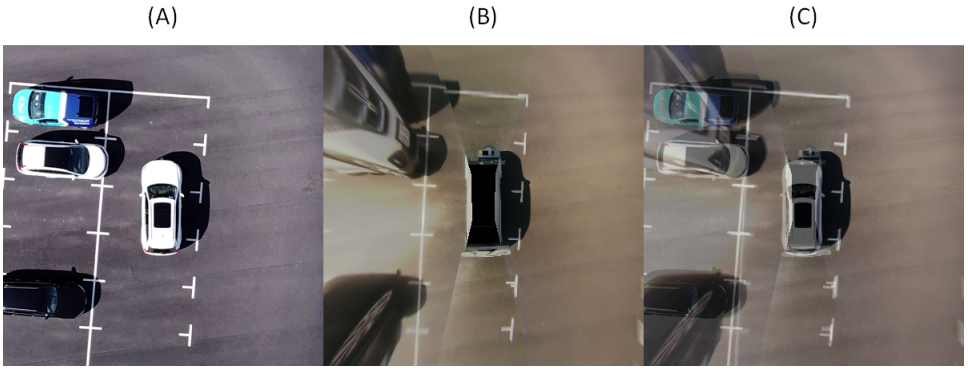

Figure 3.5: Pair of images from the same scene. (A) - Drone image; (B) - Classic BEV; (C) - Overlayed Classic BEV and Drone image.

3.1.3 Data Details and Specification

After processing all data, in total, 10 different views were stored:

- 4 Fisheye images of size 1280×1080, Fig. 3.6;

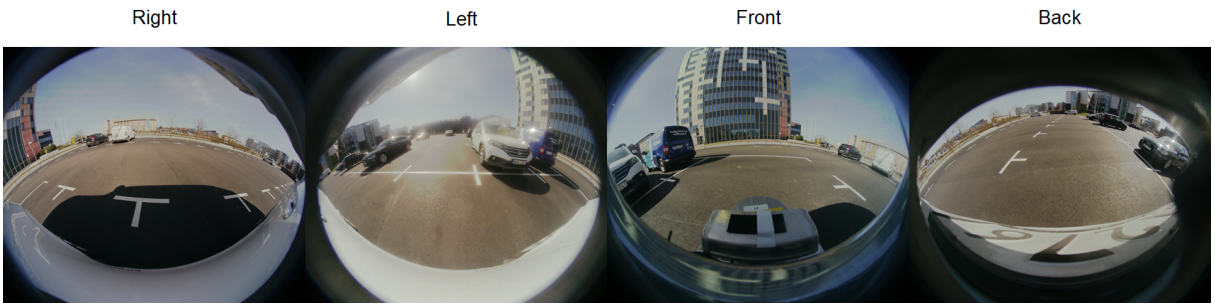

Figure 3.6: Images taken by the 4 fisheye cameras, corresponding to the same scene.

- 4 Undistorted images of size 1280×1080, Fig. 3.7;

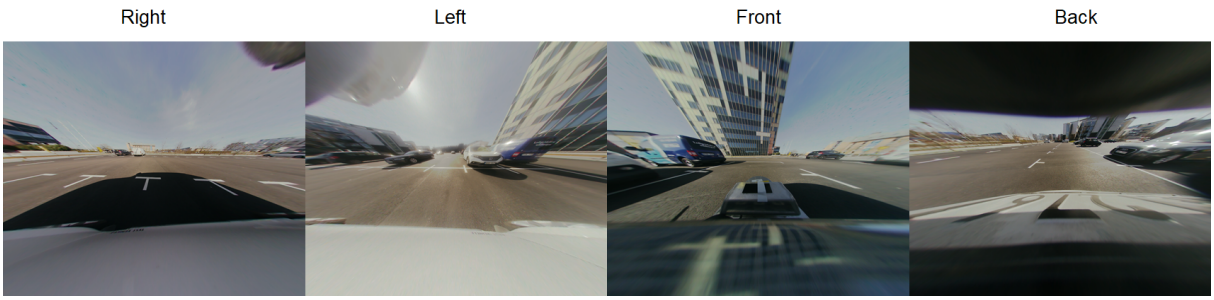

Figure 3.7: Undistorted images taken by the 4 fisheye cameras, corresponding to the same scene.

- Classic BEV images of size 400×400, Fig. 3.8;

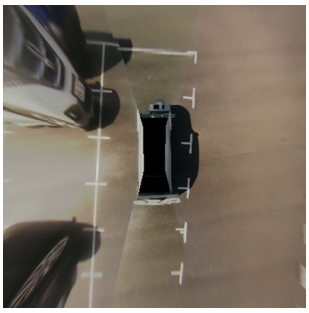

Figure 3.8: Classic BEV image

- Drone images of size 978×1034, at $59.56\ pixels/m$, Fig. 3.9;

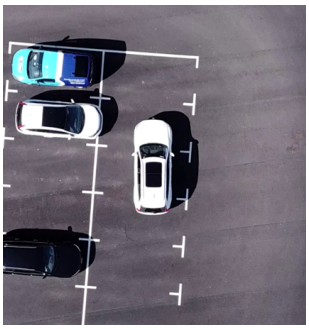

Figure 3.9: Drone Image

From all the total collected data (6276 scenes), three datasets were created: **Dataset 1**, **Dataset 2** and **Dataset 3**.

Dataset 1 contains 6276 scenes from parking lots, which include road markings, other parked cars, poles, overhead trees and slopes.

Dataset 2 contains 5364 scenes from parking lots that include road markings, other parked cars, poles, overhead trees and no slopes.

Dataset 3 contains 4904 scenes from parking lots that include road markings, other parked cars, poles, no overhead trees and no slopes.

3.2 Proposed Approaches

In this section we will go through the methodology and approaches taken to generate a BEV using GAN models. We will explain what architectures were chosen, how the data was fed to the models and how the training procedures were done.

3.2.1 Approach 1: Single-Input Model

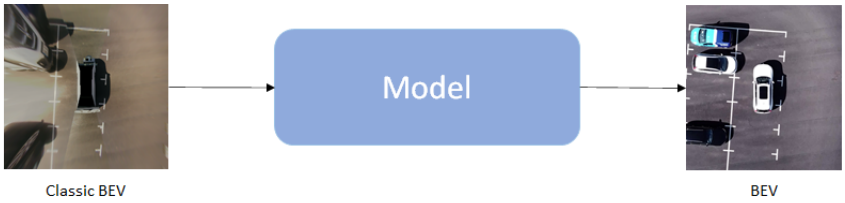

Figure 3.10: Single-input model representation

State-of-the-art GAN models in Image-to-Image Translation have mostly dealt with tasks where the images on both domains are aligned. This means that these models have difficulties learning the geometric transformations necessary to generate a BEV image, due to the large gap between the fisheye camera views and the BEV. Therefore, instead of trying to make these models learn the mapping directly from the fisheye images, we used an pre-generated BEV, through IPM, as input to these models to learn the mapping to the target output BEV, Fig. 3.10. This input will be the Classic BEV, mentioned in the previous section, as it covers the same spatial region as the desired target output image.

A study using several models was conducted to determine if the current state-of-the-art GAN models could correct the large distortions from the Classic BEV images, and at the same time retain the capacity to generate the ground plane details and improve on the overall image quality.

Four models were trained: Pix2pix [17], Pix2pixHD [35], CycleGAN [18] and AttentionGAN [19]. Each of these models fall into one of three different categories. Pix2pix and Pix2pixHD are supervised GAN models, CycleGAN is an unsupervised GAN model and AttentionGAN is a unsupervised GAN model that uses attention. With this, we aimed to evaluate which type of GAN model is best suited to the problem in question and to the given data.

Each model was either implemented according to the respective paper or trained using the official publicly available implementation. For training we used all the default hyperparameters and no alterations to the architecture or loss functions of the models were done.

3.2.1.1 Models Overview

Here we present a brief overview of each of the models used.

1. Supervised GAN models:

    - **Pix2pix**: Introduced conditional adversarial networks as a general-purpose solution to image-to-image translation problems. It uses U-Net [36] based architecture for the generator, and a con-

volutional "PatchGAN" classifier for the discriminator. In addition to the adversarial loss it also employs the L_1-distance .[17]

- **Pix2pixHD**: Building upon Pix2pix, Pix2pixHD generates higher resolution images, by using a ResNet based generator and a multi-scale discriminator. For the objective function, besides the adversarial loss it adds a perceptual loss with the VGG model network [35].

2. Unsupervised GAN models:

   - **CycleGAN**: Using 2 Discriminators and 2 Generators, it introduces a cycle consistency loss that allows the network to generate new images without the need for paired data.[18]

3. Unsupervised GAN models with attention:

   - **AttentionGAN**: This model generates attention masks via a built-in attention mechanism, which it uses to combine with the input and outputs from the image generator to create a final output. With this approach, the network can learn to ignore background areas of an image that remains unchanged and only handle the areas that require change, resulting in reduced artifacts and higher quality images.[19]

3.2.1.2 Implementation Details

The details relative to the implementation of the models are the following:

- Data: The dataset that was used to train the model was Dataset 3. As input/label images we used the Classic BEV images, and as target images we used the Drone images.
- Pre-processing: For the Classic BEV input and Drone ground truth images, a black rectangle was placed over the ego car, as it was not in our interest to learn its mapping.
- Iterations: Each model was trained for 500000 iterations.
- Image Size: We resized the images to 512 x 512 for pix2pixHD, while resizing them to 256 x 256 for all the other models.
- Data Augmentation: We performed Random Horizontal Flipping with a probability of 0.5.
- Batch size: The batch size was kept at 1 for all the models.

3.2.2 Approach 2: Multi-Input Model

In the first approach, presented in section 3.2.1, we used the Classic BEV as a single-input to the model. This means that the fisheye images still need to go through a process of undistorting, geometrical transforming, stitching and blending, to generate the Classic BEV. We want to reduce the number of pre-processing steps before feeding the data to the model, therefore for Approach 2 we propose a multi-input model, Fig. 3.11, which takes as input 4 Undistorted images from the 4 Fisheye cameras, and outputs a BEV image. Consequently we reduced the pre-processing steps to just the undistorting step.

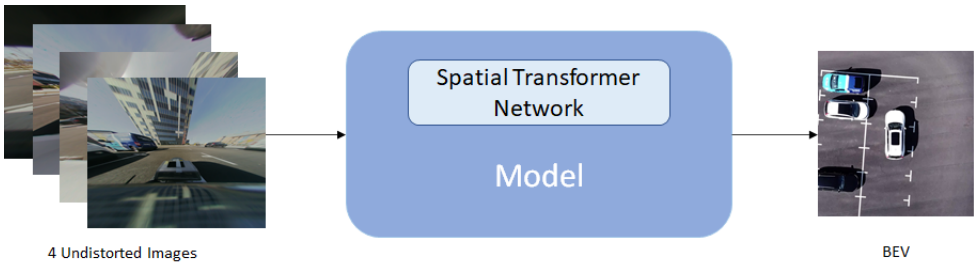

Figure 3.11: Multi-input model representation

Furthermore, when generating a Classic BEV image, a lot of information is lost in the process, as many pixels from the original Fisheye images are not present on the composition of the Classic BEV, including pixels from the ground plane where different views overlap. These lost pixels can be valuable for the model to learn the necessary features to perform an accurate mapping. Because of this, it is in our best interest to feed the models the highest amount of information. This way, all 4 Undistorted images are used, which, even though provide much more information than just using the Classic BEV, it still missing some information present in the Fisheye images. The reason that we don't use the full Fisheye images as input is because we believe that the increase in information that these would provide is negligible, and the ability for the model to correctly learn a good mapping would greatly decrease.

3.2.2.1 Generator

The authors of [1] developed a model that generates BEV images from a single frontal view image. Their model architecture is based on [35] and takes advantage of spatial transformer modules [22] to perform the necessary geometrical transformations to ensure spatial consistency with the ground truth.

Drawing inspiration from [1], our model also resorts to spatial transformer modules to perform the required perspective transformations, and due to its simplicity and extensive use in image-to-image translation tasks we also based our core architecture on the popular pix2pixHD model [35].

While our generator follows the traditional downsample-bottleneck-upsample architecture, in order to build an architecture for multiple inputs and one output, our model has separate input paths for each input image. Each path is composed of a series of downslampling layers and at least a Spatial Transformer Residual (STRes) block [1], that contains a Spatial Transformer module [22] followed by at least a single ResNet block [37]. A ResNet block is a module that has been widely used as part of CNN models and it consists of two convolutional layers and a skip-connection, where the unaltered input to the first layer is added to the output of the second layer, by means of the skip-connection. The intuition behind the use of the STRes block, was "that the slight blurring that occurs as a result of each perspective transformation is restored by the ResNet block that follows it" [1]. The Spatial Transformer module can be placed at any location in the network, therefore several locations were considered, as seen in Fig. 3.12. These locations will be later discussed.

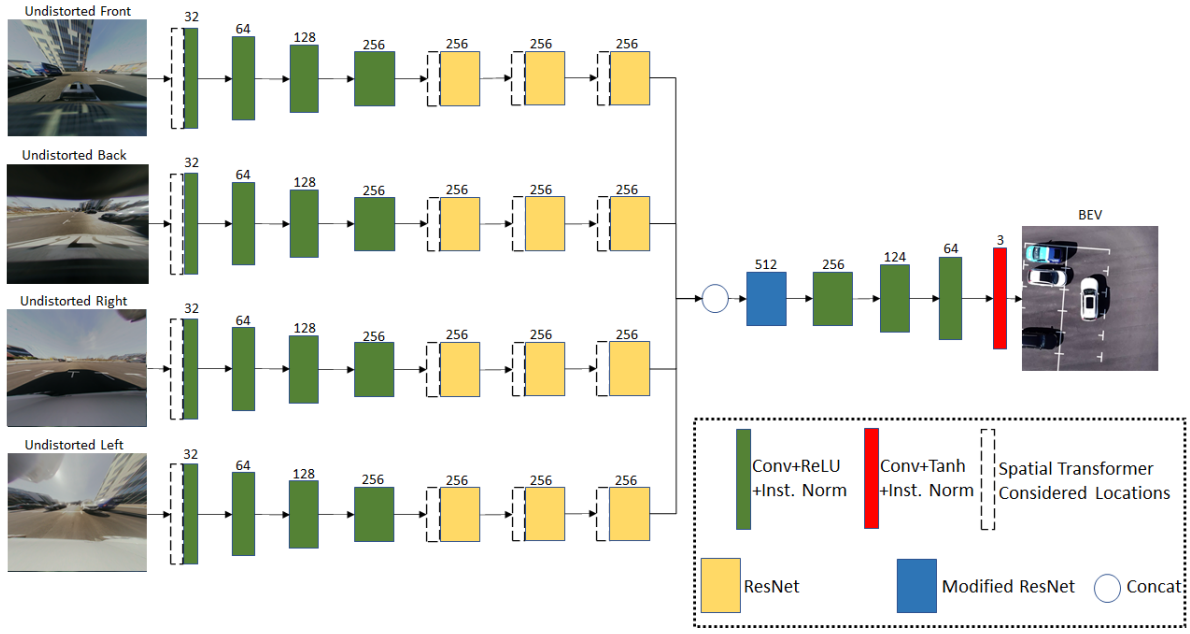

Figure 3.12: Architecture of the generator network. The number of filters is represented for each block.

The features from each input path are then concatenated into a single feature map, which is then convoluted through a modified ResNet block to reduce the number of channels of the output feature map. From here the feature maps goes through a series of upsampling layers until the network ends with the tanh() activation function.

3.2.2.2 Discriminator

The discriminator used is the same as in [35]. It is a multi-scale discriminator, meaning that it is actually composed of 3 discriminators that have the same architecture. Each discriminator will be trained for a different scale, "specifically, we downsample the real and synthesised high-resolution images by a factor of 2 and 4 to create an image pyramid of 3 scales" [35]. This downsampling is achieved through a simple average pooling operation, with stride of 2. The discriminators have a fully convolutional architecture [38], which is also commonly referred to as "PatchGAN", because the elements of the discriminators output will correspond to overlapping patches on the discriminators input. A patch is equivalent to the receptive field of the element. In the used discriminators, each output element corresponds to a 70x70 size patch.

In our model the input to the discriminator is a 15 channel wide feature map which is the result of concatenating the 4 Undistorted images and either the real Drone image or the generated image.

3.2.2.3 Spatial Transformer Networks

The goal of using the Spatial Transformer (ST) module is to ensure spatial consistency between the feature maps and the respective ground truth. The ST module tries to ensure this spatial consistency by learning transformations that are approximate to a projection of the image pixels from the Undistorted images to the ground plane. As seen in section 2.2.7, a ST is a fully differential module that can be easily integrated into

any existing model and is composed by three components: The Localisation Network, the Grid Generator and the Sampler.

The Localisation Network task is to estimate transformation matrix parameters θ_{estim}, from an input volume I . It is represented, in our case, by a simple Convolutional Neural Network (CNN), but it could also be any state-of-the-art classification network, for example EfficientNet [39], as long as we regress to the right number of parameters needed for the transformation we want to do. In our case, we want to estimate an homography matrix, therefore we regress the CNN to 8 parameters. We only need to estimate 8 parameters because the value of the matrix element at the third row, third column position is always 1 in a homography matrix, therefore we don't need to estimate it. The Grid Generator generates a grid of coordinates in the input I, corresponding to each pixel from the output O. The Sampler uses the parameters of the transformation and applies it to the input I, using bi-linear interpolation.

A ST module is not trivial to train for large perspective transformations. To stabilise its training, we define an initial transformation matrix θ_{init}, with an approximate parameterisation of the desired homography matrix, and multiply it with θ_{estim}, which is the matrix estimated by the ST, to calculate a final homography, θ. The following equation represents the matrix multiplication:

$$\theta = \theta_{estim} \cdot \theta_{init} \tag{3.5}$$

An θ_{init} was estimated for each fisheye camera. This estimation was done by selecting four approximately corresponding points on a matching pair of images, composed of an Undistorted image and a Drone image (we assume that the Drone images plane is a good approximation to the ground plane). Using these points, it is possible to arrive at a homography estimation. The same θ_{init} was used for each model.

This way, θ_{estim} is actually learning a perturbation to correct θ_{init}. Which means that θ_{estim} needs to take the form of an identity matrix at initialisation. In order to get this initialisation, the weights of the last layer of the Localisation Network are initialised with zeros and the bias with the identity matrix.

3.2.2.4 Losses

Our objective function is the same as in [1], which in term stems from [35]. It is composed of an adversarial loss $\mathcal{L}_{GAN}$, a feature matching loss $\mathcal{L}_{FM}$ and a perceptual loss $\mathcal{L}_{VGG}$.

The adversarial loss $\mathcal{L}_{GAN}$, is defined as:

$$\mathcal{L}_{GAN}(G, D) = \mathbb{E}_{(x,y)\sim P_{data}(x,y)}[logD(y, x)] + \mathbb{E}_{y\sim P_{data}(y)}[log(1 - logD(y, G(y)))] \tag{3.6}$$

In the perceptual loss, Eq. 3.7, a VVG16 [40] pre-trained network is used. Feature maps from intermediate layers of the network are extracted for both the real, and generated images, and the difference between them is calculated using the L_1-distance. This encourages the generated image and real image to have similar low and high-level feature representations extracted from loss network.

$$\mathcal{L}_{VGG}(G) = \mathbb{E}_{(x,y)\sim P_{data}(x,y)} \sum_{i=1}^{n} \frac{1}{w_i} ||VGG^{(i)}(x) - VGG^{(i)}(G(y))|| \tag{3.7}$$

Here, y is the input/label image and x is the real/target image. The weight variable, $w_i = 2^{n-i}$, is used to scale the importance of each layer, i, used in the loss, both in Eq. 3.7 and 3.8, where n is the number of intermediate layers utilised, and it was set at 4.

The feature matching loss, Eq. 3.8, is related to perceptual losses, but instead of using an auxiliary network, the feature maps are directly extracted from intermediate layers of the discriminators.

$$\mathcal{L}_{FM}(G, D_j) = \mathbb{E}_{(x,y)\sim P_{data}(x,y)} \sum_{i=1}^{n} \frac{1}{w_i} ||D_j^{(i)}(y, x) - D_j^{(i)}(y, G(y))|| \tag{3.8}$$

By using the multi-scale discriminators, D_1, D_2, D_3, together with the generator, G, the full learning objective becomes:

$$\mathcal{L}_{total} = \min_G \left(\left(\max_{D_1,D_2,D_3} \sum_{j=1,2,3} \mathcal{L}_{GAN}(G, D_j) \right) + \lambda_{FM} \sum_{j=1,2,3} \mathcal{L}_{FM}(G, D_j) + \lambda_{VGG} \mathcal{L}_{VGG}(G) \right) \tag{3.9}$$

where λ_{FM} and λ_{VGG} are empirically set at 5 and 2, respectively.

3.2.2.5 Architecture Variations

Three different variations of the generator architecture were studied. In each of different architectures, the variations consisted of modifying where the spatial transformer modules were located. In order to maintain the same learning capacity across the different models, the number of ResNet Blocks was kept constant.

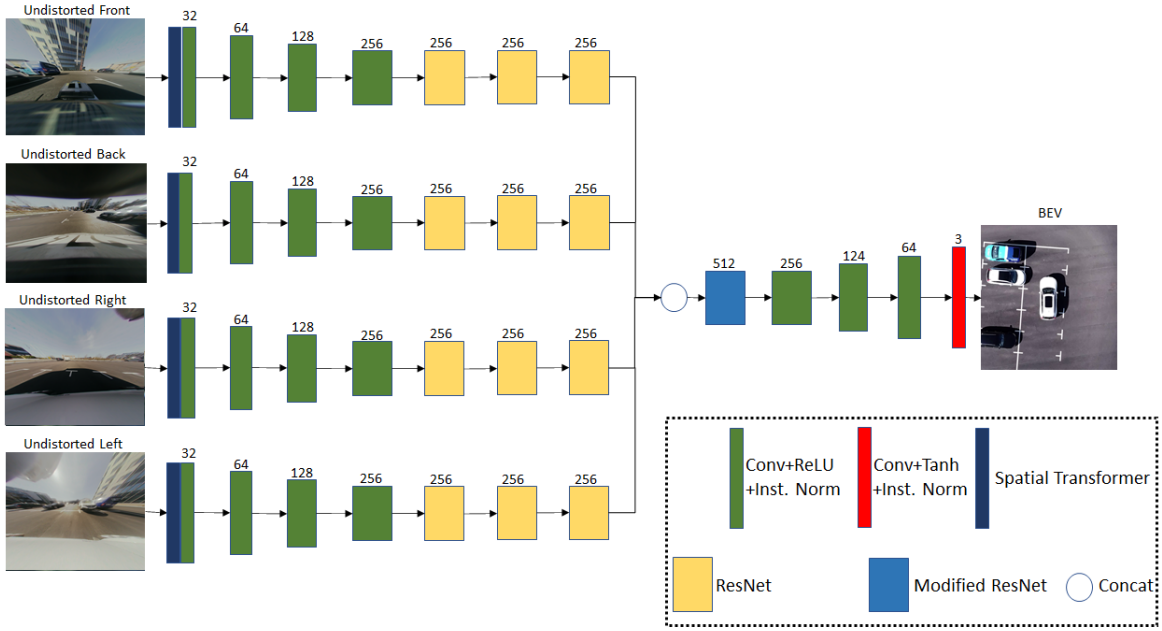

Figure 3.13: STN1-Encoder: Architecture of the generator network. The number of filters is represented for each block.

In the first variation, Fig. 3.13, the ST module is placed at the start of the network, thus transforming the input image before any convolution operation. Meaning that the network will only learn features from a BEV. Therefore, the input to the rest of the network will already be aligned with the ground truth, but it misses features that are only present in the Undistorted image. We will refer to this variation as STN1-Encoder.

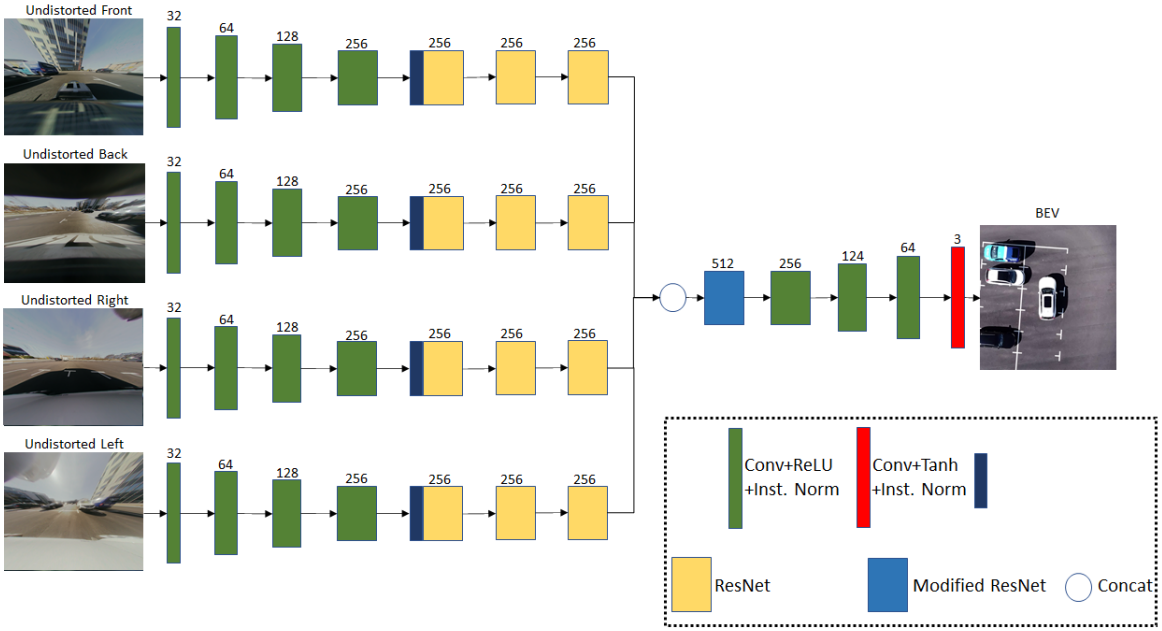

Figure 3.14: STN1-Bottleneck: Architecture of the generator network. The number of filters is represented for each block.

In the second variation, Fig. 3.14, we place the ST module at the start of the bottleneck, to perform the appropriate transformation of the feature maps encoded from the input images. We will refer to this variation as STN1-Bottleneck.

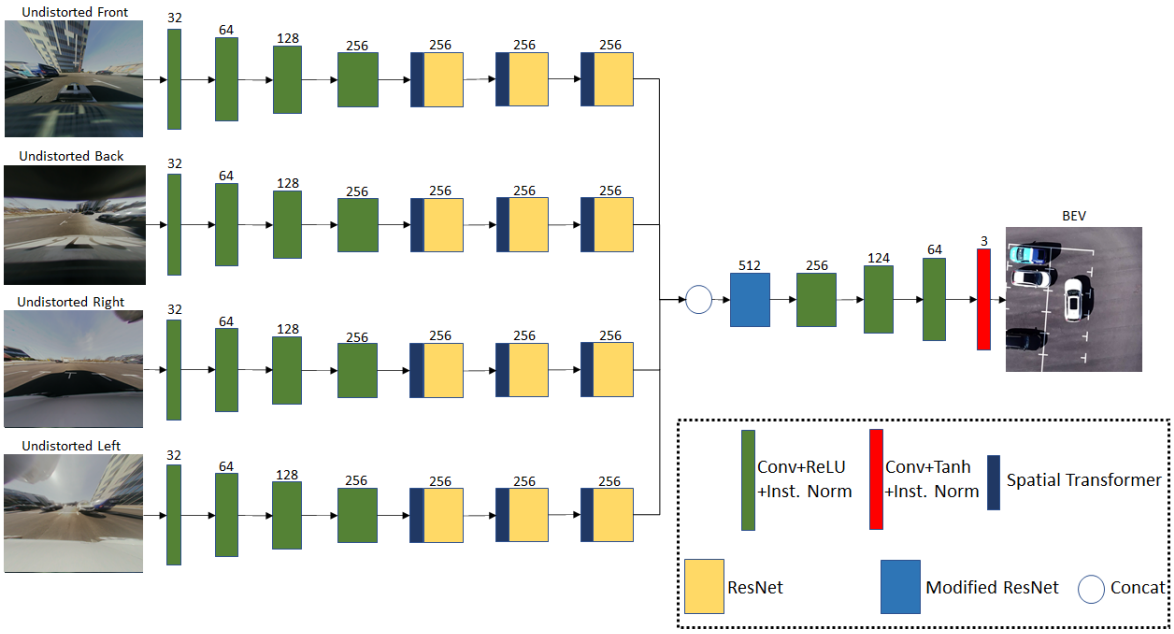

Figure 3.15: STN3-Bottleneck: Architecture of the generator network. The number of filters is represented for each block.

For the third variation, Fig. 3.15, we followed the approach in [1]. This meant placing a ST module before each ResNet block, in the bottleneck of the network, to perform incremental transformations to the ground

plane, Fig. 3.16. We will refer to this variation as STN3-Bottleneck.

Figure 3.16: Visualisation of 3 incremental transformations applied to the Undistorted frontal view.

3.2.2.6 Implementation Details

The details relative to the implementation of the models are the following:

- Data: The dataset that was used to train the model was Dataset 1. As input/label images we used the 4 Undistorted images, and as target images we used the Drone images.
- Pre-processing: For the Drone ground truth images, a black rectangle was placed over the ego car, as it was not in our interest to learn to map it.
- Iterations: Each model was trained for 250000 iterations with a batch size of 1, which was enough for the results to converge. Usually these models should be trained for more iterations but overfitting was already being observed before 250000 iterations.
- Learning Rate: The Learning Rate is initially set at 0.0001, instead of the default 0.0002 in Pix2pixHD, to stabilise the training of the ST modules. After 125000 iterations a linear scheduler decayed the learning rate to 0, at 250000 iterations.
- Image Size: The input image size was set at 512x512.
- Data Augmentation: No data augmentation was performed. While it may seem that Random Horizontal Flip could be easily applied, it would involve switching the right and left views when flipping, which might temper with the ST homography estimations.
- Localisation Network: Two different localisation network architectures were used, since the number of input channels to the network vary depending of where the ST module is located on the generator. For details regarding the localisation network architecture and implementation, please refer to appendix A.
- Batch size: The batch size was kept at 1 for all the models.

3.3 Dataset Study

As mentioned in section 3.1.3, three different datasets were created. The intent of this, was to evaluate how well the models could learn to generate a BEV given different sets of data. Each dataset was created in order

to present a different degree of difficulty. Dataset 3 presents the scenarios that should be easiest for the model to learn, while Dataset 1 presents the scenarios that should be the most challenging, as the one presented in Fig. 3.17.

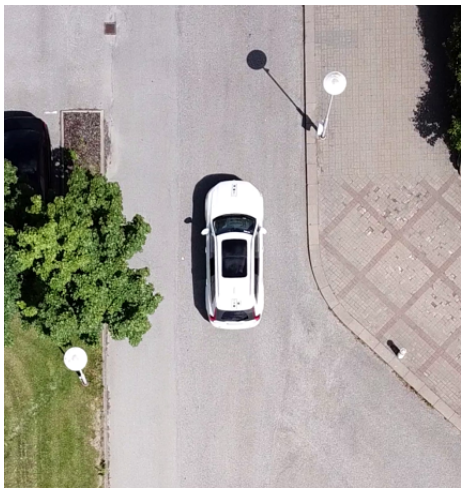

Figure 3.17: Drone image of a challenging scene. Tree tops should not be present in a "perfect" BEV, however they are represented in the drone ground truth images.

In order to evaluate how the different datasets affect the models' learning of the BEV, the same model, Pix2pixHD, with the same hyperparameters, using the same training configuration as in approach 1 (section 3.2.1), was trained for 250000 iterations for each of the three different datasets.

3.4 Evaluation of the Generated Bird's Eye View

Evaluation of generated images from GANs is a difficult problem [41]. With many works relying in qualitative perceptual studies and quantitative measures. In perceptual studies, participants are typically shown results from different methods, including the proposed method with source image, and asked to select the best translated image to target domain. In the other hand, quantitative measures involve the calculation of specific numerical scores used to summarise the quality of generated images.

Due to time constraints we did not perform a qualitative perceptual study, and instead relied on two of the most popular quantitative measures, the Frechet-Inception Distance (FID) and the Kernel-Inception Distance (KID).

To run these measures an evaluation dataset was defined. This dataset contains 159 images, all taken from the same isolated route that was not seen by the models during training.

3.4.1 Frechet-Inception Distance (FID)

The Frechet Inception Distance (FID) [42] score was proposed as an improvement over the Inception Score [43]. As with the Inception Score, the FID score uses the Inception v3 classification model [44] to capture features from an input image. The Inception v3 model is then run for two collections, one containing the real images, and another containing the generated images. The features of each collection are then summarised

as a multi-variate Gaussian. Then the distance between these two multi-variate Gaussian distributions is calculated using the Frechet distance.

A lower FID score indicates that generated images more closely match the statistical properties of real images.

3.4.2 Kernel-Inception Distance (KID)

The recently proposed Kernel-Inception Distance (KID) [45], computes the squared Maximum Mean Discrepancy between the feature representations of real and generated images. The feature representations are again extracted from the Inception v3 model. In contrast to the FID score, KID has an unbiased estimator, which makes it more reliable, especially when there are fewer test images than the dimensionality of the inception features. As it is with FID, a lower KID score indicates that generated images more closely match the statistical properties of real images.

Chapter 4

Results and Discussion

In this chapter, we will go through the quantitative results from each of the studies and approaches described in the previous chapter. We will also compare and discuss the qualitative properties of the images produced by the different models. Lastly, results from additional experiments are also presented.

4.1 Dataset

The results from the study described in section 3.3, where the same model was trained with different datasets, are presented in the following table.

	FID (lower is better)	**KID** (lower is better)
Dataset 1	1.587837524	24.40750003
Dataset 2	1.8459935	30.11656404
Dataset 3	1.964971619	32.08196163

Table 4.1: FID and KID score for the different datasets trained on the Pix2pixHD model.

From table 4.1, we note that the model trained on Dataset 1, which should have been the most challenging dataset, performed the best. While the model trained on Dataset 3, which should have been the least challenging dataset, performed the worst. The explanation that we find for these results is that all models overfitted the data. With Dataset 3, which has the least amount of images thus overfitting the most and performing worst. Dataset 2 and Dataset 1 followed with better results as these contained more data and overfitted less.

Unfortunately, the only conclusion that we can take from this study, is that the datasets didn't contain enough images to train these models without causing overfitting.

4.2 Approach 1: Single-Input Model

When evaluating the results from the first approach described in section 3.2.1, it is possible to take several findings. From table 4.2, it is noted that Pix2pixHD obtained the best score by a large margin, outperforming all the other models in both the ground plane details and on the generation of other cars, being able to remove most of the distortions from the other cars and mapping then to correct position, and represent well defined road details, as seen in Fig. 4.1 (we must also note that Pix2pixHD was trained with a larger size image). Pix2pix performed the worst, generating very blurry images. CycleGAN and AttentionGAN obtained similar scores, with the latter obtaining a marginally better score.

	FID (lower is better)	**KID** (lower is better)
Pix2pix	4.264246216	78.70054245
Pix2pixHD	1.425966034	22.57144451
CycleGAN	3.478949585	64.3399477
AttentionGAN	3.465979309	64.1622901

Table 4.2: FID and KID score for the different single-input models trained on Dataset 1.

Analysing the sample images from the different models in Fig. 4.1, the metrics scores seem to correspond to the observed quality from the generated images. Although CycleGAN has a lower score than AttentionGAN, it captured the ground plane details better than AttentionGAN, however it does not properly generate the other cars, most visibly in scenes (A) and (B). It is also observable that although Pix2pixHD generally generates more defined road markings, CycleGAN and AttentionGAN road markings are more accurate in relation to the ground truth and also more accurately orientated. This is particularly observable on the top-left corner of scene (C) and on the right side of scene (B). The reason for this could be because the Pix2pixHD rely on paired data, which due to constrains explained in section 3.1, does not always align with the ground truth.

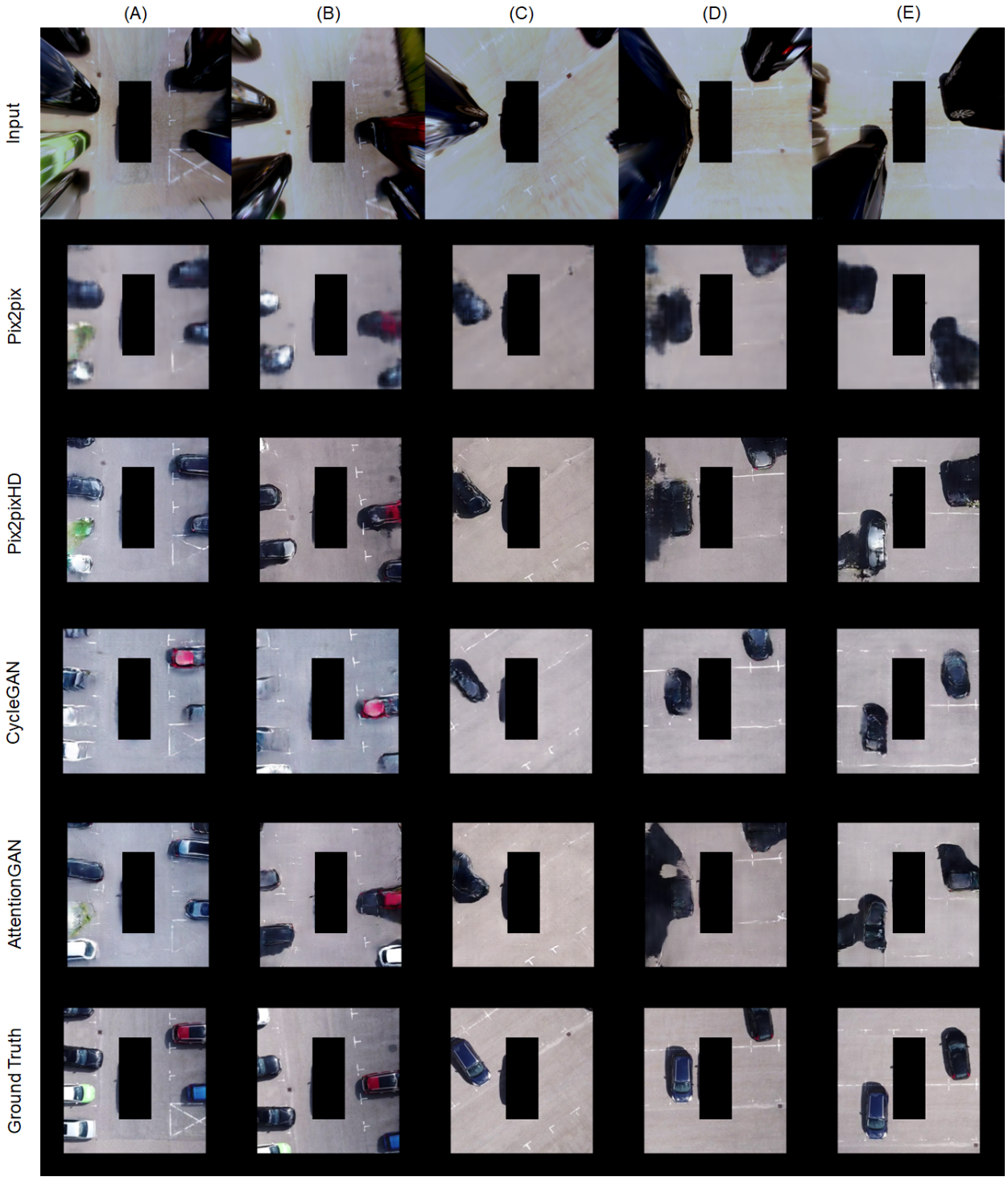

Figure 4.1: Samples from 5 different scenes, (A) to (E), generated by the four different models (Pix2pix, Pix2pixHD, CycleGAN, AttentionGAN), and the respective input and ground truth.

4.3 Approach 2: Multi-Input Model

From the model variations described in section 3.2.2, all were able to generate a BEV from the 4 Undistorted images. However, the overall quality of the generated images remained relatively low. We believe that this is mostly due to model overfitting, which is caused by the reduced amount of training data and the lack of data augmentation. Another reason could be the size of the input images, that is 4 times smaller than the images that are used to generate the Classic BEV. Nevertheless, it is still possible to compare the different model variations from the results, and consequently, take away conclusions.

Figure 4.2: Samples from 5 different scenes, (A) to (E), generated by the three different model variations (STN3-Bottleneck, STN1-Bottleneck, STN1-Encoder), and the respective ground truth.

From the generated images, it is possible to assess two main characteristics of the generated images: how well the ground plane and road markings details are represented, and how the distortions from the other cars are corrected.

When observing the ground plane details in Fig. 4.2, the STN1-Encoder variation was the one that was able to generate them with more similarity to the ground truth, this is particularly observable at scene (C), where most of the road markings at the bottom and top-left corner of the image are generated, while for the

other variations it is harder to make out those details. Also, in scene (A), on the left side of the image the road details are again more accurately portrayed on the variation STN1-Encoder. Comparing STN1-Bottleneck and STN3-Bottleneck, it is possible to observe that the latter is able to reproduce details better, as seen in scene (A) and (B), where the small dark drain on the left side of the image is faintly generated with STN3-Bottleneck, while it is not generated at all with STN1-Bottleneck. We argue that this difference is due to the incremental transformation performed on STN3-Bottleneck.

When looking at how the other cars are generated, all variations generate similar results, although it is noticeable that the STN1-Encoder model variation generated cars are slightly more blurry and incorrectly shaped, compared with the other variations.

In Fig. 4.3, the initialisation homography and a representation of the learnt homographies from each of the model variations is presented. Most noticeable in the right view, it is observable that the ST modules are generally able to correct the initial homography to a transformation that will align the images more closely with the ground truth. Although, a failure case is present in the front view of the STN1-Encoder model, where the ST model not only didn't correct the initial transformation, but also completely failed to learn a reasonable transformation.

	FID (lower is better)	**KID** (lower is better)
STN3-Bottleneck	2.10153244	36.19625866
STN1-Bottleneck	2.234797058	37.92501092
STN1-Encoder	2.124474487	36.25893891

Table 4.3: FID and KID score from our multi-input model variations.

From the scores in table 4.3, STN3-Bottleneck performed the best and STN1-Bottleneck performed the worst. We interpret that the better score for STN3-Bottleneck is due to a combination of good ground plane details, and better representation of other cars. While, the STN1-Bottleneck model was penalised for the lack of ground plane details and the STN1-Encoder model was favoured for its better representability of the ground plane details.

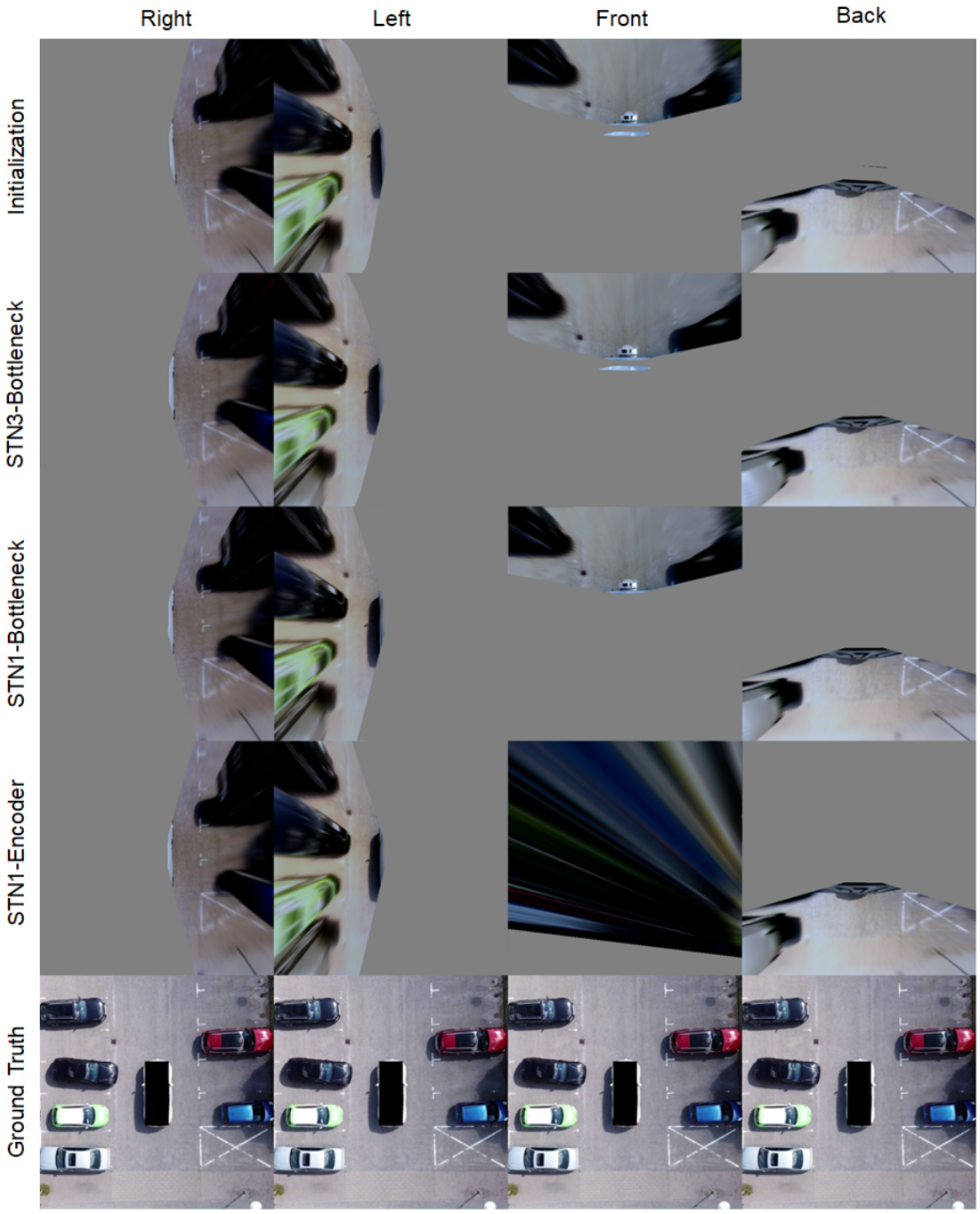

Figure 4.3: Visualisation of the initialisation homographies, the learned homographies from each of the model variations (STN3-Bottleneck, STN1-Bottleneck, STN1-Encoder), and the respective ground truth. Each homography was applied to the undistorted images in order to gain a more intuitive understanding of the transformation. All images are taken from the same scene.

4.4 Additional Experiments

Although not described in the previous chapter, other experiments worth noting were also run. In particular experiments using the images directly from the fisheye cameras, without any pre-processing. These experiments

were run using the same state-of-the-art single input GAN models as in section 3.2.1. And several variations of mappings were tried:

- Concatenated Fisheye Images to Classic BEV.
- Stitched side-by-side Fisheye Images to Classic BEV.
- Concatenated Fisheye Images to Drone image.
- Stitched side-by-side Fisheye Images to Drone image.
- Single frontal Fisheye Images to Classic BEV frontal view.
- Single frontal Fisheye Images to Drone frontal view.

Unfortunately all these experiments generated very unsatisfactory results and in many cases the GAN models entered in mode collapse [46]. As we discuss in section 3.2.1 current state-of-the-art GAN models are optimised for tasks where images from both domains are aligned, therefore they perform poorly when large perspective transformations are required, as seen here.

Another experiment using the same STN3-Bottleneck with the same homography initialisation from section 3.2.2 was also run. Here, the only difference is that instead of trying to learn the mapping from the Undistorted images to the Drone image, the model learns the mapping to the Classic BEV.

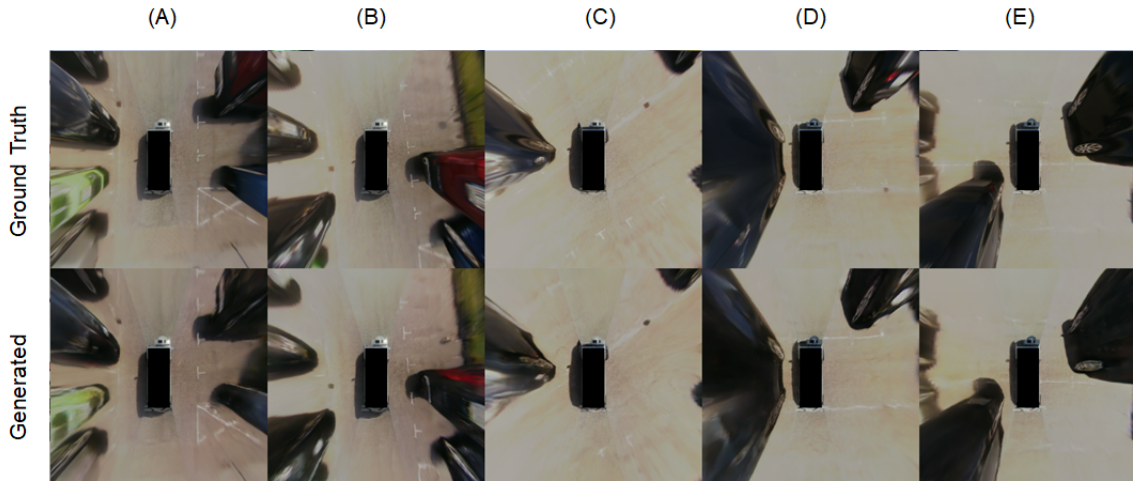

Figure 4.4: Samples from 5 different scenes, (A) to (E), generated by STN3-Bottleneck model variation, using the Classic BEV images as ground truth.

As seen in Fig. 4.4, this experiment proved to be quite successful, with our model being able to map the undistorted images to the Classic BEV very accurately, achieving an equivalent quality. Only with some colour discrepancies, visible on the red car in scene (A) and (B). If the model was fed the full resolution images, even better results should be expected. This demonstrates that the STN3-Bottleneck model has the capability to correctly learn the mapping from multiple images to a single image.

Chapter 5

Conclusion and Further Work

Generating a Bird's Eye View (BEV) with current state-of-the-art Generative Adversarial Networks (GANs), given only fisheye images was not achieved. Nonetheless, it is shown that without greatly increasing the methods' complexity, similar approaches yield promising results. In particular, Spatial Transformer (ST) modules demonstrate a strong potential in aiding a GAN to learn a correct mapping to the BEV.

Of the two proposed methods, both the single-input and the multi-input approaches presented viable options and produced similar results. The single-input approach is simpler in its implementation, but its potential is limited by the information contained a BEV generated through traditional methods. On the other hand, the multi-input models using ST modules are harder to train, but their possibilities are still very much open for future experimentation.

By training paired and unpaired GAN models, it is observed, that although paired models result in overall higher quality and better metric scores, unpaired models demonstrate higher accuracy at generating ground plane details. Our multi-input model was only implemented in a paired configuration, consequently, we believe that an unpaired/unsupervised implementation should also be explored.

Our multi-input models showed that is possible to generate a BEV, with reasonable quality, from 4 undistorted images, by resorting to ST modules that ensure spatial consistency. Of the model variations, the STN1-Encoder was able to generate better ground plane details, although for this model the ST module was more unstable while training. On the other hand, the STN3-Bottleneck better corrected the other cars' distortions and was easier to train. Therefore, it is recommended that a model combining both variations is explored. Such model would contain ST modules both in the encoder and in the bottleneck of the generator, and follow a U-Net style architecture [36] with skip connections. Also, only simple CNN architectures were used for the localization networks in the ST modules, which proved to be sufficient. Nonetheless, we believe that other architectures are to be explored in order to stabilize the training process.

This study also stresses how crucial a large quantity of data is to avoid model overfitting. While the recorded 6276 samples proved to be enough to conduct comparative studies, a dataset containing at least 15000 samples is recommended to train a robust model, capable of inferring in parking lot scenarios. Furthermore, data augmentation should also be better explored, including augmentation of the ST modules, by using random homography initialisation.

References

[1] Tom Bruls et al. "The Right (Angled) Perspective: Improving the Understanding of Road Scenes Using Boosted Inverse Perspective Mapping". In: *arXiv:1812.00913 [cs]* (May 2, 2019). arXiv: 1812.00913. URL: http://arxiv.org/abs/1812.00913 (visited on 06/30/2020).

[2] Nicolai Müller et al. *The road to 2020 and beyond: What's driving the global automotive industry?* Aug. 2013. URL: https://www.mckinsey.com/~/media/mckinsey/dotcom/client_service/Automotive%20and%20Assembly/PDFs/McK_The_road_to_2020_and_beyond.ashx.

[3] Hanspeter Mallot et al. "Inverse Perspective Mapping Simplifies Optical Flow Computation and Obstacle Detection". In: *Biological cybernetics* 64 (Feb. 1991), pp. 177–85. DOI: 10.1007/BF00201978.

[4] Davy Neven et al. "Towards end-to-end lane detection: an instance segmentation approach". In: *2018 IEEE intelligent vehicles symposium (IV)*. IEEE. 2018, pp. 286–291.

[5] Bonolo Mathibela, Paul Newman, and Ingmar Posner. "Reading the road: road marking classification and interpretation". In: *IEEE Transactions on Intelligent Transportation Systems* 16.4 (2015), pp. 2072–2081.

[6] Pietro Cerri and Paolo Grisleri. "Free space detection on highways using time correlation between stabilized sub-pixel precision IPM images". In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE. 2005, pp. 2223–2228.

[7] Alex Zyner, Stewart Worrall, and Eduardo Nebot. "Naturalistic driver intention and path prediction using recurrent neural networks". In: *IEEE Transactions on Intelligent Transportation Systems* 21.4 (2019), pp. 1584–1594.

[8] Nico Engel et al. "Deep Object Tracking on Dynamic Occupancy Grid Maps Using RNNs". In: *arXiv:1805.08986 [cs]* (May 23, 2018). arXiv: 1805.08986. URL: http://arxiv.org/abs/1805.08986 (visited on 09/11/2020).

[9] Ian J. Goodfellow et al. "Generative Adversarial Networks". In: *arXiv:1406.2661 [cs, stat]* (June 10, 2014). Reporter: arXiv:1406.2661 [cs, stat]. arXiv: 1406.2661. URL: http://arxiv.org/abs/1406.2661 (visited on 02/25/2020).

[10] Kapje Sung et al. "Development of Image Synthesis Algorithm with Multi-Camera". In: *2012 IEEE 75th Vehicular Technology Conference (VTC Spring)*. 2012 IEEE 75th Vehicular Technology Conference (VTC Spring). ISSN: 1550-2252. May 2012, pp. 1–5. DOI: 10.1109/VETECS.2012.6240323.

[11] Buyue Zhang et al. "A Surround View Camera Solution for Embedded Systems". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops. ISSN: 2160-7516. June 2014, pp. 676–681. DOI: 10.1109/CVPRW.2014.103.

[12] Yucheng Liu and Buyue Zhang. "Photometric alignment for surround view camera system". In: *2014 IEEE International Conference on Image Processing (ICIP)*. 2014 IEEE International Conference on Image Processing (ICIP). ISSN: 2381-8549. Oct. 2014, pp. 1827–1831. DOI: 10.1109/ICIP.2014.7025366.

[13] Miguel Oliveira, Vitor Santos, and Angel D. Sappa. "Multimodal inverse perspective mapping". In: *Information Fusion* 24 (July 2015), pp. 108–121. ISSN: 15662535. DOI: 10.1016/j.inffus.2014.09.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S1566253514001031 (visited on 09/07/2020).

[14] J. Jeong and A. Kim. "Adaptive Inverse Perspective Mapping for lane map generation with SLAM". In: *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. 2016, pp. 38–41.

[15] Ian J. Goodfellow et al. "Generative Adversarial Networks". In: *arXiv:1406.2661 [cs, stat]* (June 10, 2014). arXiv: 1406.2661. URL: http://arxiv.org/abs/1406.2661 (visited on 02/25/2020).

[16] Mehdi Mirza and Simon Osindero. "Conditional Generative Adversarial Nets". In: *arXiv:1411.1784 [cs, stat]* (Nov. 6, 2014). arXiv: 1411.1784. URL: http://arxiv.org/abs/1411.1784 (visited on 02/21/2020).

[17] Phillip Isola et al. "Image-to-Image Translation with Conditional Adversarial Networks". In: *arXiv:1611.07004 [cs]* (Nov. 26, 2018). arXiv: 1611.07004. URL: http://arxiv.org/abs/1611.07004 (visited on 02/21/2020).

[18] Jun-Yan Zhu et al. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks". In: *arXiv:1703.10593 [cs]* (Nov. 15, 2018). arXiv: 1703.10593. URL: http://arxiv.org/abs/1703.10593 (visited on 02/25/2020).

[19] Hao Tang et al. "AttentionGAN: Unpaired Image-to-Image Translation using Attention-Guided Generative Adversarial Networks". In: *arXiv:1911.11897 [cs, eess]* (Feb. 12, 2020). arXiv: 1911.11897. URL: http://arxiv.org/abs/1911.11897 (visited on 05/27/2020).

[20] Junho Kim et al. "U-GAT-IT: Unsupervised Generative Attentional Networks with Adaptive Layer-Instance Normalization for Image-to-Image Translation". In: *arXiv:1907.10830 [cs, eess]* (Apr. 8, 2020). arXiv: 1907.10830. URL: http://arxiv.org/abs/1907.10830 (visited on 04/16/2020).

[21] Xinge Zhu et al. "Generative Adversarial Frontal View to Bird View Synthesis". In: *2018 International Conference on 3D Vision (3DV)*. 2018 International Conference on 3D Vision (3DV). ISSN: 2378-3826. Sept. 2018, pp. 454–463. DOI: 10.1109/3DV.2018.00059.

[22] Max Jaderberg et al. "Spatial Transformer Networks". In: *arXiv:1506.02025 [cs]* (Feb. 4, 2016). arXiv: 1506.02025. URL: http://arxiv.org/abs/1506.02025 (visited on 06/30/2020).

[23] Xinchen Yan et al. "Perspective Transformer Nets: Learning Single-View 3D Object Reconstruction without 3D Supervision". In: *arXiv:1612.00814 [cs]* (Aug. 12, 2017). arXiv: 1612.00814. URL: http://arxiv.org/abs/1612.00814 (visited on 09/10/2020).

[24] Tinghui Zhou et al. "View Synthesis by Appearance Flow". In: *arXiv:1605.03557 [cs]* (Feb. 11, 2017). arXiv: 1605.03557. URL: http://arxiv.org/abs/1605.03557 (visited on 09/10/2020).

[25] Miaomiao Liu, Xuming He, and Mathieu Salzmann. "Geometry-aware Deep Network for Single-Image Novel View Synthesis". In: *arXiv:1804.06008 [cs]* (Apr. 16, 2018). arXiv: 1804.06008. URL: http://arxiv.org/abs/1804.06008 (visited on 06/30/2020).

[26] Berthold Horn. *Robot Vision*. Jan. 1986. ISBN: 978-0-262-08159-7.

[27] Bruce D Lucas, Takeo Kanade, et al. "An iterative image registration technique with an application to stereo vision". In: (1981).

[28] "Aperture Problem". In: *Encyclopedia of Neuroscience*. Ed. by Marc D. Binder, Nobutaka Hirokawa, and Uwe Windhorst. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 159–159. ISBN: 978-3-540-29678-2. DOI: 10.1007/978-3-540-29678-2_310. URL: https://doi.org/10.1007/978-3-540-29678-2_310.

[29] Christopher G Harris, Mike Stephens, et al. "A combined corner and edge detector." In: *Alvey vision conference*. Vol. 15. 50. Citeseer. 1988, pp. 10–5244.

[30] Jianbo Shi et al. "Good features to track". In: *1994 Proceedings of IEEE conference on computer vision and pattern recognition*. IEEE. 1994, pp. 593–600.

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[32] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. "Multilayer feedforward networks are universal approximators." In: *Neural networks* 2.5 (1989), pp. 359–366.

[33] Thalles Silva. *An intuitive introduction to Generative Adversarial Networks (GANs)*. freeCodeCamp.org. Jan. 7, 2018. URL: https://www.freecodecamp.org/news/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394/ (visited on 09/16/2020).

[34] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[35] Ting-Chun Wang et al. "High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs". In: *arXiv:1711.11585 [cs]* (Aug. 20, 2018). arXiv: 1711.11585. URL: http://arxiv.org/abs/1711.11585 (visited on 07/02/2020).

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

[37] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *arXiv:1512.03385 [cs]* (Dec. 10, 2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (visited on 04/17/2020).

[38] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *arXiv:1411.4038 [cs]* (Mar. 8, 2015). arXiv: 1411.4038. URL: http://arxiv.org/abs/1411.4038 (visited on 09/01/2020).

[39] Mingxing Tan and Quoc V Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *arXiv preprint arXiv:1905.11946* (2019).

[40] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[41] Ali Borji. "Pros and cons of gan evaluation measures". In: *Computer Vision and Image Understanding* 179 (2019), pp. 41–65.

[42] Martin Heusel et al. "Gans trained by a two time-scale update rule converge to a local nash equilibrium". In: *Advances in neural information processing systems*. 2017, pp. 6626–6637.

[43] Tim Salimans et al. "Improved techniques for training gans". In: *Advances in neural information processing systems*. 2016, pp. 2234–2242.

[44] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[45] Mikołaj Bińkowski et al. "Demystifying mmd gans". In: *arXiv preprint arXiv:1801.01401* (2018).

[46] Ian Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks". In: *arXiv:1701.00160 [cs]* (Apr. 3, 2017). arXiv: 1701.00160. URL: http://arxiv.org/abs/1701.00160 (visited on 02/25/2020).

Appendix A

Localisation Networks

Listing A.1: Pytorch implementation of the ST module localisation network. Used when the input has a 256 channel width, which is the case for STN1-Bottleneck and STN3-Bottleneck.

```
self.net = nn.Sequential(nn.Conv2d(256, 128, kernel_size=7),
                         nn.ReLU(True),
                         nn.Conv2d(128, 64, kernel_size=7),
                         nn.ReLU(True),
                         nn.Conv2d(64, 32, kernel_size=7),
                         nn.ReLU(True),
                         nn.Conv2d(32, 16, kernel_size=3),
                         nn.MaxPool2d(2, stride=2),
                         nn.ReLU(True),
                         nn.Conv2d(16, 10, kernel_size=3),
                         nn.MaxPool2d(2, stride=2),
                         nn.ReLU(True),
                         Flatten(),
                         nn.Linear(10**3, 256),
                         nn.ReLU(True),
                         nn.Linear(256, out_dim))
```

Listing A.2: Pytorch implementation of the ST module localisation network. Used when the input has a 3 channel width, which is the case for STN1-Encoder.

```
self.net = nn.Sequential(nn.Conv2d(3, 64, kernel_size=11, stride=4),
                         nn.ReLU(inplace=True),
                         nn.Conv2d(64, 192, kernel_size=5),
                         nn.ReLU(inplace=True),
                         nn.Conv2d(192, 384, kernel_size=3),
                         nn.MaxPool2d(kernel_size=3, stride=2),
                         nn.ReLU(inplace=True),
                         nn.Conv2d(384, 192, kernel_size=3),
                         nn.MaxPool2d(kernel_size=3, stride=2),
                         nn.ReLU(inplace=True),
                         nn.Conv2d(192, 64, kernel_size=3),
                         nn.MaxPool2d(2, stride=2),
                         nn.ReLU(inplace=True),
                         nn.Conv2d(64, 32, kernel_size=3),
                         nn.MaxPool2d(2, stride=2),
                         nn.ReLU(True),
                         nn.AdaptiveAvgPool2d((16, 16)),
                         Flatten(),
                         nn.Linear(16*16*32, 2048),
                         nn.ReLU(True),
                         nn.Linear(2048, 1024),
                         nn.ReLU(True),
                         nn.Linear(1024, out_dim))
```