UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Data Science Curriculum

Ilmar Uduste

Effect of Delays/Lag and Fighting it in Self-driving Neural Networks

Master's Thesis (15 ECTS)

Supervisor: Ardi Tampuu, PhD

Tartu 2022

Effect of Delays/Lag and Fighting it in Self-driving Neural Networks

Abstract:
Autonomous driving is a field of interest for academia and industry alike, with hopes of fully replacing humans in the driver seat with artificial intelligence. Recent advances in the domain have been made in the use of end-to-end self-driving models as opposed to modular approaches.

However, problem with building these end-to-end pipelines is that delays (lag) are commonly not taken into account. This work investigates the effect of lag in self-driving nets using Donkey Car, an autonomous car platform, and finds that the car drives worse when there is more lag in the pipeline. A novel method called *frameshift* is proposed to fight against the lag present and models using *frameshift* are proven to have significant real-life (closed-loop) performance gains over the baseline self-driving net, even in no-lag conditions and when comparing models analytically (open-loop). *Frameshift* is shown to be an effective tool in fighting against lag, although performance in very lag-heavy environments is inconsistent, as the car makes too few decisions per second to have consistent behaviour.

The findings presented show the need to test self-driving cars in real-life (closed-loop) as opposed to just analytically (open-loop) and also open up the field of end-to-end self-driving nets to the concept of using *frameshift* as a potential tool to fight against lag, although this novel idea requires further testing in different conditions and real-world data, to come to a final conclusion.

Keywords:
Autonomous vehicles, end-to-end self-driving, neural networks, lag, delays, model evaluation, frameshift

CERCS: P176 Artificial intelligence

Viivituste mõju ja vastumeetod selle vähendamiseks isesõitvates tehisnärvivõrkudes Lühikokkuvõte:

Isesõitvate autode valdkond on huviks nii teadlastele kui ka erasektorile ning muuhulgas loodetakse tulevikus asendada inimautojuht sama või isegi etema sõiduoskusega tehisintellektiga. Hiljutiste arengutega on valdkonnas leidnud kasutust isesõitvate modulaarsete torude asemel hoopis täielikult tehisnärvivõrkudel põhinevad torud.

Laadsete isesõitvate täielikult tehisnärvivõrkudel põhinevate torude ehitamisel aga esineb tõrkeid, nimelt ei võeta arvesse auto sõitmisel esinevaid viivitusi. Käesolev töö uurib viivituste mõju isesõitvatele tehisnärvivõrkudele, kasutades selleks puldiga juhitavat isesõitvat autoplatvormi Donkey Car. Töö käigus leiti, et isesõitev auto sõidab sedavõrd halvemini, mida rohkem on viivitusi torus. Antud viivituste vastumeetmeks pakuti välja uus meetod *kaadrinihe*, mille tõhusust ja positiivset toimet üle tavaliste mudelite omakorda tõestati nii analüütiliselt (avatud ahela hindamine) ning ka isesõitva autoga rajal (suletud ahela hindamine). Kaadrinihkega närvivõrgud sõitsid võrreldavatest närvivõrkudest etemini nii viivitustega kui ka viivitusteta keskkonnas, kuigi jõudlus viivituste-tihedas keskkonnas oli ebaühtlane, kuna auto ei jõua ühtlaseks sõiduks vajalikus koguses otsuseid teha.

Töös ilmnes vajadus testida isesõitvaid autosid ka päriselus (suletud testimine) ning mitte ainult analüütiliselt (avatud testimine) ning samuti, et kaadrinihke meetod omab piisavalt potentsiaali, et olla tulevikus uurimisaluseks isesõitvate autode valdkonnas, kuna laadseid meetodeid viivituste vastu võitlemiseks pole varasemalt rakendatud. Samas tõdeti, et äsja väljapakutud meetod nõuab edasist testimist ning uurimist enne lõpliku järelduse tegemist.

Võtmesõnad:
Autonoomsed sõidukid, otsast lõpuni isejuhtimine, tehisnärvivõrgud, viivitused, mudeli hindamine, kaadrinihe

CERCS: P176 Tehisintellekt

Contents

1 Introduction **6**

2 Background **7**
2.1 Machine Learning 7
2.1.1 Artificial Neural Networks 8
2.1.2 Convolutional Neural Networks 10
2.2 Autonomous Vehicles 12
2.2.1 Modular Self-driving 15
2.2.2 End-to-end Self-driving 16
2.2.3 Delay or Lag in Self-driving Vehicles 17
2.3 Donkey Car 17

3 Methodology **20**
3.1 Hardware Setup 21
3.1.1 Hardware Issues 22
3.1.2 Drivable Track 23
3.2 Data Preparation 25
3.2.1 Gathering Drive Tubs 25
3.2.2 Preprocessing Drive Tubs 26
3.3 Frameshift 28
3.3.1 Creating Frameshifted Tubs 28
3.4 Model Training 28
3.5 Evaluation 30

4 Experiments and Results **33**
4.1 Baseline Model Selection 33
4.1.1 Image Augmentation and Transformation 33
4.1.2 Training Frameshifted Models 35
4.2 Open-loop Performance of Frameshifted Models 35
4.2.1 Steering Angle Distribution 35
4.2.2 Error Metrics 37
4.2.3 Timegraph 40
4.3 Closed-loop Performance of Frameshifted Models 42
4.3.1 Performance at different Delays 42
4.3.2 Lag Inconsistency and Driving Score 44
4.3.3 Lag Delta 45
4.4 Explaining Open-loop Evaluations Through Closed-loop Performance . 47

5 Conclusion **48**

6 Future Work **50**

References **53**

Appendix **54**
I. Experimental Results of Frameshifted Models on the Track 54
II. Confusion Matrix of Frameshifted Model Predictions 55
III. Licence 56

1 Introduction

Autonomous driving is a field of study that is of great interest to both industry and academia, and therefore has seen a surge of interest and developments over the past few decades. These developments have been fueled by recent advances in sensing and computing technology, together with the potentially transformative impact on automotive transportation and the perceived societal benefit. Autonomous vehicles have the potential to dramatically reduce the contribution of driver error and negligence as the cause of vehicle collisions [Paden et al., 2016]. In the last few years, end-to-end approaches, where a single neural network has replaced the entire driving pipeline, have been gaining traction as viable alternatives to modular methods in autonomous driving [Tampuu et al., 2020]. However, there are still many issues to contend with before end-to-end approaches can be declared 'usable' in the real world.

One of the aforementioned problems is that delays (lag) are commonly not considered when building driving pipelines. Imitation learning based end-to-end models learn to mimic human driving by learning a function that maps observation o_t at time t to the action a human would take at that same time point, a_t. However, when deployed the actions take time to calculate, and the appropriate control command for time t is found at $t + dt$. Moreover, these signals then also need to be passed to the motor and steering wheels, which might incur further delays. This means that the signal being sent to the car might not be relevant by the time it is deployed. Being late can become an issue since even if the model can perfectly predict what a human would do in a particular situation, the answer is found too late and the vehicle might not drive correctly (e.g. stay on track).

Among the works surveyed by Tampuu et al., no authors discuss this issue nor report taking any measures to fight the delays. Preliminary experiments regarding lag in self-driving nets have been undertaken in a student project by training a neural network to drive a Donkey Car (an open-source, pre-assembled, self-driving toy car) [Donkey, 2022], while also having some built-in lag [H. Kurniawan, 2021]. Initial results seem to indicate that models trained on zero lag (0 ms lag) perform badly in conditions where lag exists and the higher the lag, the worse the performance is.

This thesis aims to investigate the effect that lag in predicting the actions of a self-driving car might have on the performance on the self-driving car. In particular, a new method to combat the lag called *frameshift* is proposed. In essence, frameshift consists of shifting the labels of the data of a supervised learning task with respect to the input along the time axis and here it is used to shift the steering angle of the self-driving car in relation to the frames from the camera of the car. Self-driving nets are trained using imitation learning and frameshift and the performance of this novel method is evaluated. To the best of the author's knowledge, this is the first time anyone has published using this type of method to combat lag in self-driving nets.

All actions noted in this thesis were taken by the author, unless explicitly stated otherwise.

2 Background

This chapter describes the prerequisite knowledge required to understand the field of autonomous vehicles and the content of this thesis, more specifically the fundamental principles of machine learning, artificial neural networks, convolutional neural networks, modular self-driving, end-to-end self-driving and also the Donkey Car that is used in this thesis as the self-driving car.

2.1 Machine Learning

Machine learning (ML) refers to the study and use of algorithms that allow computer programs to automatically improve through experience or data [Mitchell, 1997]. These algorithms initially have no inherent know-how on how to solve a specific task, instead, they observe sets of training data points that are given to them and the algorithms discover useful regularities that help solve the task [Bishop, 2007]. *Training* machine learning *models* refers to iterating over training samples to discover and fine-tune the set of patterns and rules that allow the model to solve the task with maximal performance [Tampuu, 2020]. Having trained a model, new data not present in the training data can be introduced to the model with hopes of the model solving the task in also this new situation.

There are many categories of machine learning, but this thesis specifically deals with only the *supervised learning* part of machine learning. Supervised learning refers to training the ML algorithms with data containing not only input data, but also the desired outputs. Through this training data, the supervised learning algorithms learn a function that is used to predict the output associated to a particular input as accurately as possible. This accuracy metric can be defined in many different ways and is often called the *loss function*. Ideally, a well trained supervised learning algorithm would also correctly determine the output for inputs that were not originally present in the training data (i.e. generalize to new situations), but this might not always be the case due to limitations in the model or stochasticity in the desired outputs. The aim of the algorithm is to, rather simply, minimize the sum error (sum loss) across all training input-output pairs.

Supervised learning tasks include, among others, classification, which is a task of identifying the correct value among a set of categories that an observation belongs to, and regression, referring to statistical processes meant for estimating the relationships between inputs and continous-valued outputs.

In classification tasks, the output is a categorical value and therefore belongs to a limited set of values. Examples of classification include predicting whether the animal in the picture is a dog or a cat, or marking a given email to be spam or non-spam. So the question that can be answered by using classification algorithms is '*which class does this input map out to?*'.

In regression tasks, the output is a real numbered value, meaning the output has an infinite amount of possible values. The simplest type of regression models is linear regression, which is a method for finding the line that most closely fits the training data according to some mathematical criterion. Examples of use cases for regression include predicting the taxi fare based on the distance driven, or predicting the weight of a person based on their age, height and other personal data. Regression usually answers questions such as '*how many?*', '*when?*' or '*where?*' and the core of the problem of this thesis, predicting steering angles based on camera frames, is also a regression problem where we answer the question '*how much and in which direction would a human steer the car in this situation?*'.

The data meant for training and using these machine learning models is usually split into two or three categories: training, validation and test data, with validation data being optional (but highly recommended for training models). Naturally, the models are trained on the training data, then possibly validated (to evaluate and pick the best-performing models) on the validation data while training, then later on tested on the test data. This train-val-test split is done to test if the model could generalize to situations it hasn't seen while training. If 100% of the data is used for training, then the model might show good performance on training data, but it is unknown how well it generalizes to new data. Seeing as the goal of a model is often to predict answers based on new and previously unseen data, the ability to generalize is a rather important one, and so this train-val-test split is very often used.

2.1.1 Artificial Neural Networks

Artificial neural networks or ANNs refer to sets of algorithms or computing systems vaguely inspired by biological neural networks that can be optimized to perform tasks without being programmed to use any task-specific rules. Similarly to a human's nervous system, information and data is processed by interconnected neurons that combine their inputs through the usage of weight parameters that define the neurons' connection strengths with each other [Stanford University, 2022]. The structure of an ANN can vary from network to network, but one of the simpler architectures is a fully-connected neural network. An example of this consists of fully-connected neurons in four layers (Figure 1). Each layer performs the same basic (linear) operation, but differ in placement and usage.

The input layer is the layer through which the feature values (inputs) are inserted into the network, by setting neuron activations equal to the values of input features. In the case of an input of a 32x32 RGB image, the image is separated into three channels and each pixel-color value (a total of 32x32x3 = 3072 values) is inserted into a separate neuron of the input layer.

The hidden layer(s) is responsible for summing up the incoming weighted signals from the neurons in the previous layer and processing the summed signal with a thresh-

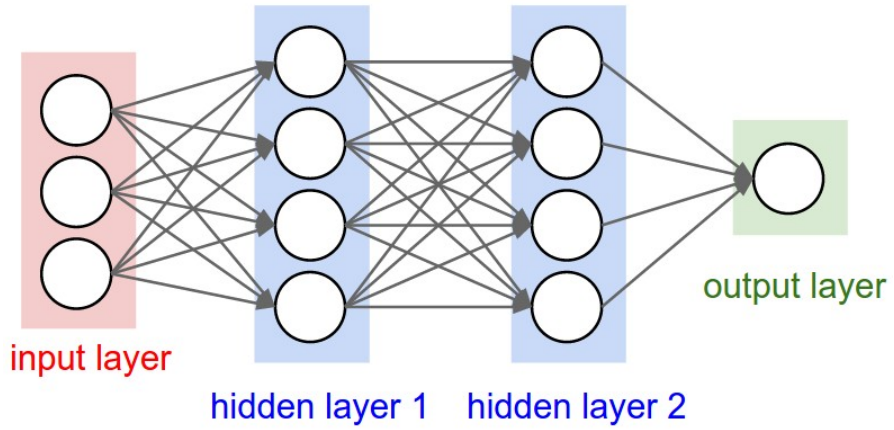

Figure 1. A 4-layer neural network with three inputs, two hidden layers of 4 neurons each and one output node/neuron. Notice that there are connections (synapses) between neurons across layers, but not within a layer [Stanford University, 2022]. Each neuron of layer $n + 1$ receives connections (inputs) from all of the neurons in layer n, giving this netowork type the name "fully-connected". Image from https://cs231n.github.io/neural-networks-1/.

olding function (i.e. an activation function). In a fully-connected neural network (like the one in Figure 1), each neuron is connected with all the neurons in the previous layer.

The output layer maps the summed values of the previous layer to the required outputs. In the case of classification tasks, this is usually done using *softmax* (a normalization algorithm) to output a vector of probabilities p_n (the sum of this vector is 1) representing how likely the input is to belong to a certain class of output. In regression tasks, the output layer often simply sums together the weighted inputs from the previous layer and outputs this single number without further processing, as the expected output generally is a real-numbered output (e.g. price when predicting taxi fare).

Neural networks, like humans, learn through experience and data. This means that all the weights (or edges in Figure 1) and also biases are learned in the training process. The output of each hidden layer neuron can be expressed like so:

$$h = \sum_{i}^{N} x_i w_i + b$$

where h designates the output of the neuron, N the number of incoming edges, x_i the output of the previous layer's neuron at index i, w_i the weight of the connection between the i-th neuron in the previous layer and the current neuron and b signifies the bias value in the current neuron.

The parameters or weights and biases of the ANN are adjusted in the training process,

which itself consists of two subsequently applied algorithms: forward pass and backwards propagation (also known as *backprop*) [Rumelhart et al., 1986]. The forward pass consists of feeding the input data through the ANN and the ANN predicting an output which can then be compared to the desired output and the size of loss (e.g. Mean Squared Error in regression or Cross Entropy loss in classification) is calculated.

During backpropagation, the gradients of the error are calculated with respect to all the parameters of the ANN and after the fact, the parameters of the net are updated according to the gradients. To find these gradients, the derivatives in the last layer are calculated, then all gradients are found moving layer-by-layer back towards the input layer (this move gives the name to the backprop algorithm). This method is used since the number of trainable parameters in modern networks, such as object-detection nets using convolutional neural networks (e.g. ResNet [He et al., 2015] and AlexNet [Krizhevsky et al., 2012]) can reach up to tens or hundreds of millions, and this backpropagation minimizes the number of calculations needed to update these parameters.

Deep learning refers to the study and usage of such ANNs that have multiple hidden layers.

2.1.2 Convolutional Neural Networks

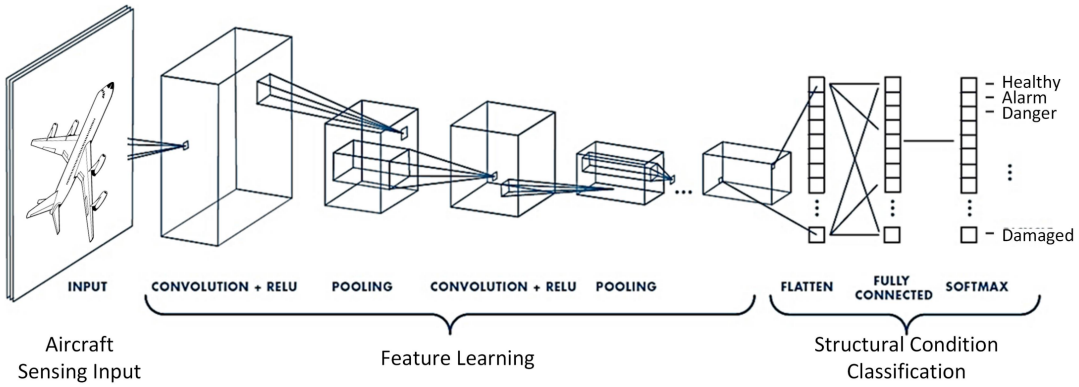

Figure 2. A typical Convolutional Neural Network architecture in aircraft structural health monitoring; the typical configuration, inputs and outputs are illustrated showing the system operation principle. [Tabian et al., 2019]

Regular artificial neural networks with just fully connected layers, however, tend to struggle with the computational complexity required to compute image data [O'Shea and Nash, 2015]. *Convolutional Neural Networks* (CNNs), with the use of kernels, pooling and fully-connected layers, can drastically reduce the dimensionality of images, but still extract valuable information out of them. This is why CNNs are the preferred tool for image recognition [Yamashita, 2018, Tabian et al., 2019, O'Shea and Nash, 2015].

Convolutional Neural Networks are analogous to fully-connected ANNs in that they are comprised of neurons that self-optimise through learning. But CNNs particularly are used in the field of pattern recognition within images. They allow to encode image-specific features into the architecture of the net, making the network more suited for image-focused tasks, whilst reducing the number of parameters required to set up the model [O'Shea and Nash, 2015].

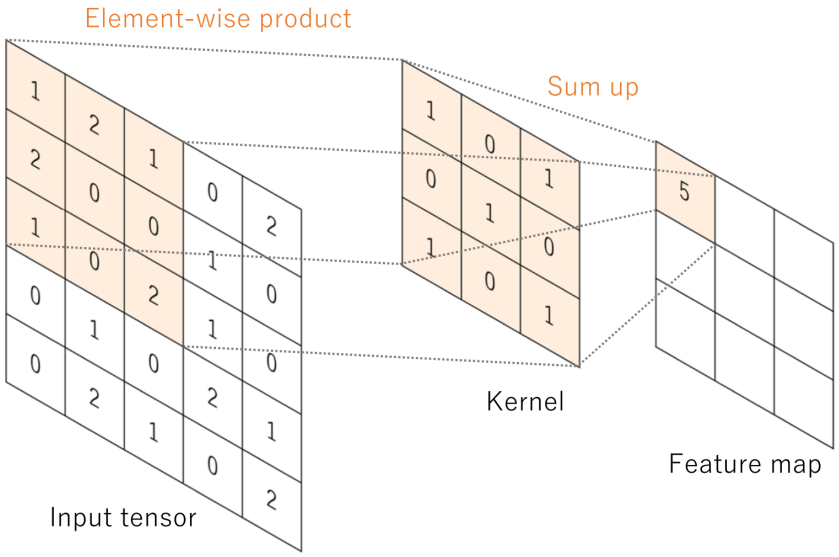

Figure 3. An example of a convolution operation with a kernel size of 3 × 3, no padding, and a stride of 1. A kernel is applied to different locations across the input tensor, and an element-wise product between each element of the kernel and the current region of input tensor is calculated at each location and summed to obtain the output value in the corresponding position of the output tensor. Collection of these values forms a spatial image called a feature map. [Yamashita, 2018]

There are many variations of CNNs, but it usually has three types of layers: convolutions, pooling and fully-connected layers (Figure 2). The convolution layer is made of multiple convolution kernels (Figure 3) that learn feature representations of the inputs and generate feature maps (Figure 4). A feature map is the result of first convolving the input with a learned kernel, and next, by applying an element-wise non-linear activation function on the convolved results [Tabian et al., 2019].

The convolutional layers consist of matrices of learnable weights called *filters* (or kernels) that are applied to the input (e.g. an image, output of other convolutional layers) to recognize patterns, make sense of them, and extract features (e.g. lines, shape, textures and objects) out of the input [Yamashita, 2018]. Each filter in the convolutional layer extracts a different feature from the input (Figure 4).

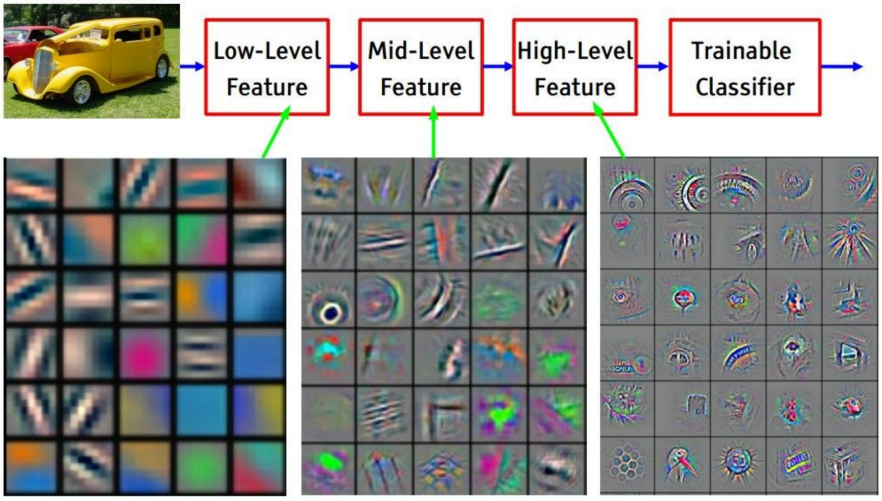

Figure 4. Feature visualization of a convolutional net trained on ImageNet. [Stanford University, 2022]

CNNs also differ from ANNs by often using features such as *padding*, *stride* or *pooling*. Padding is the simple process of padding (or adding pixels) the border of the input image, and is an effective method to give further control over the dimensionality of the output volumes [O'Shea and Nash, 2015].

The distance between two successive locations the kernel is applied at is called a stride, which also defines the convolutional operation. A common choice of stride is 1; however, a stride larger than 1 is sometimes used in order to achieve downsampling of feature maps. An alternative technique to perform downsampling is a pooling operation, as described below [Yamashita, 2018].

The pooling layer aims to gradually reduce the dimensionality of the representation, and thus, further reduce the number of parameters and the computational complexity of the model. It functions by iterating over each activation map in the input and scaling its dimensionality using the "MAX" function. In the case of pooling kernels with a dimensionality of 2x2 applied with a stride of 2, the activation map is scaled down to 25% of the original size (only the maximal value of each 2x2 area is kept) - whilst maintaining the depth (number of feature maps) volume at its original size [O'Shea and Nash, 2015].

2.2 Autonomous Vehicles

Autonomous vehicles (also known as self-driving cars and driverless cars) have been studied and developed by many universities, research centers, car companies, and companies

of other industries around the world since the middle of the 1980s [Badue et al., 2021]. These developments have been fueled by recent advances in sensing and computing technology together with the potential transformative impact on automotive transportation and the perceived societal benefit. Autonomous vehicles have the potential to dramatically reduce the contribution of driver error and negligence as the cause of vehicle collisions. They will also provide a means of personal mobility to people who are unable to drive due to physical or visual disability [Paden et al., 2016].

The level of autonomy of self-driving cars is classified using a system developed by the Society of Automotive Engineers (SAE) that is based on the amount of human driver intervention and attentiveness required by them [SAE, 2018]. The detailed descriptions of these levels of autonomy are as follows:

- **Level 0 - No Driving Automation.** Most cars are at level 0, since they have no autonomous driving capability, although there may be systems in place to help the driver, such as emergency braking (since it's technically not driving, rather the stopping of driving).
- **Level 1 - Driver Assistance.** This is the lowest level of automation and usually includes systems that are meant for driver assistance, e.g. steering or cruise control. Adaptive cruise control, where the drivable vehicle keeps a safe distance from the next one, is classified as Level 1 autonomy, as the driver still controls steering and braking.
- **Level 2 - Partial Driving Automation.** Automated driver assistance systems (or ADAS) are in place and the vehicle can perform steering and acceleration. This still falls short of autonomous driving, as the human driver can take control at any time. Note that most commercially sold 'autonomous vehicles' are at Level 2 autonomy.
- **Level 3 - Conditional Driving Automation.** This level of autonomy now features environment detection, which is a subtle difference compared to level 2, but it is quite a large jump technologically. The car can also make more informed decisions, but human override is still often required. Audi A8s are classified as Level 3 in the EU.
- **Level 4 - High Driving Automation.** Human interaction isn't required here in most cases, as Level 4 autonomy allows the vehicle to intervene if there is a system failure or something else goes wrong. These vehicles can operate in self-driving mode, but they are usually limited by legislation to only be allowed to drive in a certain geographic area (also known as *geofencing*). These areas are usually urban environments where the maximum speed is up to 50 km/h (e.g. cities, towns etc.). Most level 4 vehicles are currently geared towards ridesharing and a few notable examples are already on the 'robotaxi' market.

- **Level 5 - Full Driving Automation.** The self-driving vehicle can do everything by itself that a human driver can. This level of autonomy requires no human intervention, so the vehicle most likely has no steering wheel nor pedals for a human to drive. These kinds of fully autonomous cars are undergoing testing around the world, but no models are available to the general public as of the time of writing [SAE, 2018, Synopsis, 2022].

The architecture of the autonomy system of self-driving cars usually consists of two main parts: the perception system, and the decision-making system [Paden et al., 2016]. The perception system is generally divided into many subsystems responsible for tasks such as autonomous car localization, static obstacles mapping, road mapping, moving obstacles detection and tracking, traffic signalization detection and recognition, among others. The decision-making system is commonly partitioned as well into many subsystems responsible for tasks such as route planning, path planning, behavior selection, motion planning, obstacle avoidance and control, though this partitioning is very dependant on the particular implementation [Badue et al., 2021].

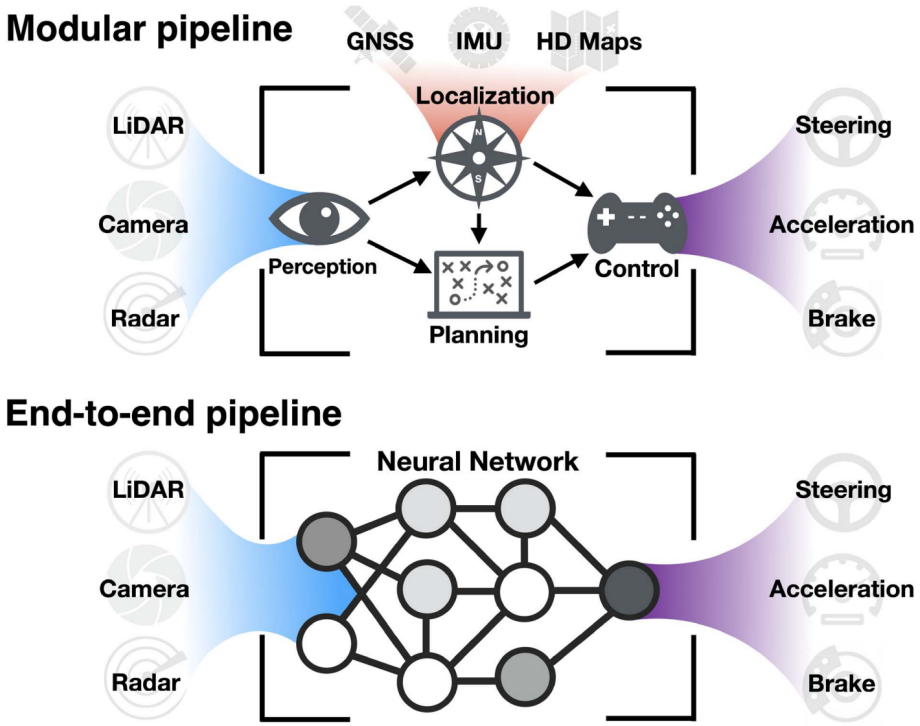

Figure 5. Modular and end-to-end pipelines. The modular pipeline for autonomous driving consists of many interconnected modules, while end-to-end approach treats the entire pipeline as one learnable machine learning task. [Tampuu, 2020]

On a broad level, the research on autonomous driving can be divided into two main approaches (Figure 5): 1) modular self-driving and 2) end-to-end self-driving [Tampuu et al., 2020]. The next chapters detail the main differences and use-cases for these two self-driving paradigms.

2.2.1 Modular Self-driving

Modular self-driving, also known as the mediated perception approach, is widely considered to be the conventional approach to autonomous vehicles and is the primary method of doing so [Tampuu et al., 2020]. This is due to the traditional divide-and-conquer engineering principle, where the AI drivers rely on modules with identifiable responsibilities (Figure 5), such as route planning, maneuver control, traffic sign recognition etc. [Xiao et al., 2019]. The atomicity of the architecture helps explain the decisions of the AI driver in terms of its modules, which is a clear advantage over end-to-end self-driving nets.

The decision making system of a typical self-driving car is hierarchically decomposed into four components (Figure 6). At the highest level, a route is planned through the road network. This is followed by a behavioral layer, which decides on a local driving task that progresses the car towards the destination and abides by the rules of the road. A motion planning module then selects a continuous path through the environment to accomplish a local navigational task. A control system then reactively corrects errors in the execution of the planned mo-

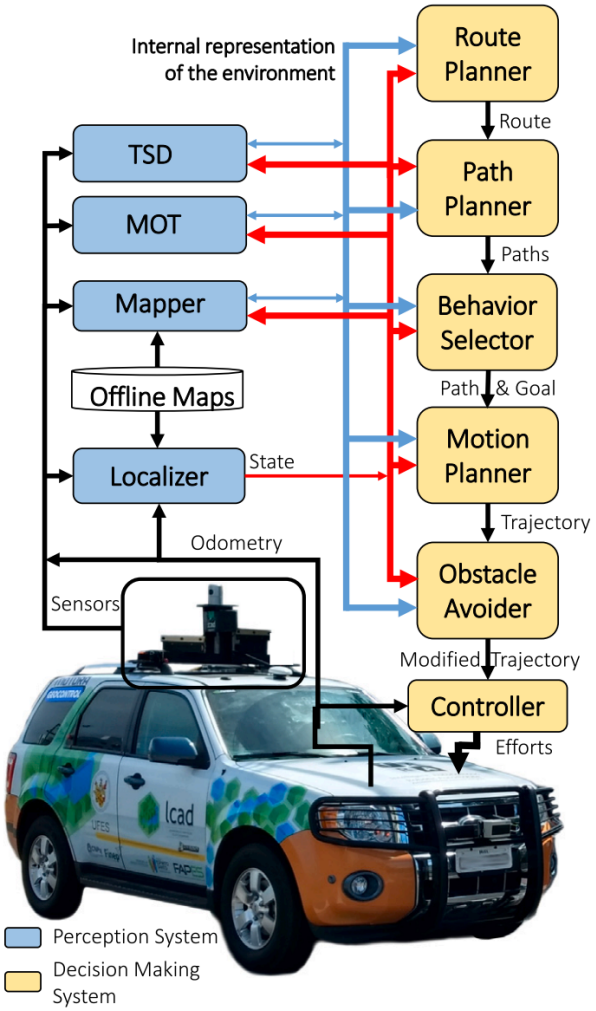

Figure 6. Overview of the typical architecture of the automation system of self-driving cars. TSD denotes Traffic Signalization Detection and MOT denotes Moving Objects Tracking. [Badue et al., 2021]

tion [Paden et al., 2016].

Currently, most of the commercially available cars with self-driving capability (e.g. cars by Waymo or more established car companies with new electric and autonomous vehicle offerings) are of the modular self-driving type [Badue et al., 2021]. These modular pipelines do offer high levels of autonomy: mostly up to Level 2 autonomy, sometimes up to Level 3 and rarely up to Level 4 (geared towards ridesharing or taxi cars) [Synopsis, 2022], but building and maintaining such pipelines are costly and despite many man-years of work, such approaches are still far from complete autonomy [Tampuu et al., 2020].

2.2.2 End-to-end Self-driving

In the last few years, end-to-end approaches, where the entire driving pipeline has been replaced by a single neural network, have been gaining traction as viable alternatives to modular approaches in autonomous driving [Tampuu et al., 2020]. However, there are still many issues to contend with before end-to-end approaches can be declared 'usable' in the real world.

End-to-end driving, also referred to as direct perception, generates ego-motion, or the continuous operation of steering wheel and pedals, directly from sensory inputs (Figure 5). Three main approaches to end-to-end driving currently dominate the field: *direct supervised deep learning*, *neuroevolution* and also *reinforcement learning* [Yurtsever et al., 2019]. This thesis focuses only on direct supervised deep learning or imitation learning, where the end-to-end pipeline learns from the actions of an *expert driver* (the author).

When compared to modular pipelines, end-to-end pipelines have the distinct advantage of the entire pipeline of transforming sensory inputs to driving commands being a single learning task. The pipeline does not have to rely on inputs and outputs from many different systems to make a final decision about driving, so optimizing a vehicle for autonomous driving is much simpler in the end-to-end paradigm, as engineers have to cover all possible situations and scenarios themselves in the modular approach [Tampuu et al., 2020].

The prospect of using deep learning as a substitute for the whole self-driving pipeline seems promising, however it has not yet been sucessfully deployed in urban environments (except for limited demonstrations). The biggest shortcomings of end-to-end driving in general are the lack of hard coded safety measures and interpretability [Yurtsever et al., 2019]. Supervised learning in self-driving also begs the question: should self-driving nets learn to drive like humans, when approximately 1.3 million people die from road traffic injuries each year? [World Health Organization, 2021]

Nevertheless, the field of end-to-end driving is rapidly growing and the interest by many industries to automate driving and also research end-to-end pipelines bodes hope for the roads to be rather safe in the future.

2.2.3 Delay or Lag in Self-driving Vehicles

One of the problems regarding autonomous vehicles is that latency or lag is not always taken into account when building driving pipelines. The actions that a self-driving vehicle should take will take time to calculate in real-time and then to pass this command to the motor and steering wheels. This can become an issue since even if the model can perfectly predict what a human would do in a particular situation, the answer is found too late and the vehicle might not stay on track.

In end-to-end driving, correcting for this lag is not common, as evidenced by no work surveyed discussing the topic [Tampuu et al., 2020]. A student project at the University of Tartu in which the self-driving neural networks had built-in lag suggests that models trained without correcting for lag perform badly in conditions where lag exists and the higher the lag is, the worse the performance is [H. Kurniawan, 2021]. However, further testing is required to substantiate these claims and this study aims to do so.

2.3 Donkey Car

The end-to-end self-driving models in this thesis are built and tested using a Donkey Car (Figure 7).

The self-driving models are quite expensive to test on real cars and therefore a self-driving car platform called Donkey has been built by a community of self-driving car enthusiasts. The Donkey Car itself is a remote-controlled and camera-equipped car with self-driving capabilities and its hardware design is open-source, making it easy for one to build their own Donkey Car.

The work of this thesis is based on the underlying assumption that the Donkey Car is controlled via Raspberry Pi. The important part for this thesis is that one can connect to the Donkey Car via *ssh*, drive tracks and gather data, develop self-driving models based on that data and then test said models on the same car.

The training data for self-driving models is ideally gathered by driving the Donkey Car through some kind of track without any crashes or mistakes. The data of the expert behaviour is recorded onto the disk of the car and extracted onto a computer to do preprocessing, validation and modelling via imitation learning, which is the dominant paradigm used in end-to-end driving [Tampuu et al., 2020].

The data itself usually comes in the form of 3 features recorded simultaneously:

1. **Frames from the camera of the car that is being driven.** The number of frames is usually between 10 to 30 per second and in our particular case, it's 20Hz. The default resolution of the Donkey Car camera is 320x240 pixels and, since it is a wide-angle camera, the field of vision is 160°.

2. **Throttle:** a float between 0 and 1 corresponding to a ratio of battery voltage that is fed into the car's engine. Unfortunately, this is not a very reliable feature,

Figure 7. The Donkey Car along with the Logitech F710 Controller used to drive it.

as it correlates only somewhat with the car's speed or acceleration due to being dependent on the voltage of the battery (i.e. how charged the battery is) and other factors (e.g. heat build-up in engines). The speed of the car can vary wildly even at the same throttle.

3. **The steering angle:** a float between -1 and 1 reflecting the voltage given to the engine controlling steering, with -1 mapping to a full left turn and +1 a full right turn. The battery level is unlikely to influence the turning radius, but might alter the speed of achieving the desired wheel angle. The effective turning radius is, however, also dependent on the car physical setup. At maximal turn value, the radius can vary from 90 cm to 70 cm depending on the physical calibration. In this work, we always use the same car, i.e same physical setup, excluding wear and tear. The actual steering radius is also influenced by the speed of the car. For simplicity's sake, this project will assume that self-driving cars will be trained and tested with the same conditions i.e. with the same speed, ergo the speed of the car should not have an effect on the experiments.

Driving data is recorded into folders called *tubs*, where the data is organised into images (320x240 RGB frames from the camera at 20Hz) and catalogs (in a JSON format) that contains metadata that goes along with a particular frame: timestamp, behaviour

label, user steering angle, user mode and user throttle. A 20-minute ‘tub’ or drive takes up approximately 260MB and more than 95% of that size is from the captured images.

These driving tubs are then used as training data in training a self-driving model via the integrated Donkey Python module. The performance and prediction speed of the self-driving net depend on the selected model type, the amount, quality and resolution of the training data and also time of day, as lighting conditions can change the input data (images) of these nets quite a lot. Building a functional self-driving net can take multiple iterations of collecting training data, cleaning the data, training networks and then testing the networks.

3 Methodology

The main research question that this thesis investigates is "Does adding frameshift to self-driving neural nets (i.e. correcting for the lag) improve performance in laggy situations?". For a more complete understanding of the effects, secondary questions are also raised, such as "In what range of lag amount can models with differing frameshift amounts function?" or "How does adding negative frameshift (predicting the past) affect the behaviour of the car?".

The main steps to answer these and similar research questions are as follows:

1. **Find out the time delta** that the end-to-end pipeline in the Donkey Car needs in order to send a predicted steering angle to the car from the moment it captures an image. This will further on be referred to as *signal-to-steering time* (STS).

2. **Capture recordings** of successful and clean drives on a particular track (talked about in detail further on).

3. **Create frameshifted recordings** corresponding to an amount of positive or negative lag.

4. **Train end-to-end self-driving models** using the exemplary recordings as training data. Use image augmentation/transformation techniques, lag amount, lag direction and model type to differentiate between these models.

5. **Assess the performance** of the aforementioned self-driving models in order to evaluate the ranges of lag that lag-corrected and non-lag-corrected models can drive with at human-like speeds.

In this project, it is intended to use only two of the aforementioned recorded features for self-driving neural networks: the frames from the camera as input and the steering angle of the car corresponding to each frame (Figure 8) as output. During data collection, the car will be driven by the author of the project and the controller is a Logitech F710 Wireless Gamepad, emulating the experience one might have driving RC cars or cars in a video game. Throttle is set to a fixed value during data collection (and later during model testing), to reduce the mental load during data collection - it is much easier to only control steering, allowing one to drive faster and cleaner. It is also known that simpler networks struggle to learn throttle control and we would risk losing time getting models to drive at all if one attempts to create models that output both steering and throttle values.

For a baseline self-driving linear neural network, about 5,000 - 20,000 frames (or $\tilde{8}$ to $\tilde{3}0$ minutes at 10Hz) of clean driving should suffice, although performance gains have been noted when training a model on up to 100,000 frames, with negligible performance gains over that number [R. Abumalikov, 2022]. It should be noted that not all of these

Figure 8. A 320x240 frame taken from a video (made via the *makemovie* tool in Donkey) of the driving of the Donkey Car. The green line indicates the steering angle applied to the car by the operator and the blue line indicates the prediction of the steering angle by the self-driving neural network.

frames have to come from the same drive, as they can be combined from different recordings that are captured during different times of the day (leading to potentially different lighting conditions and therefore, more variation in the data) and also multiplied via data augmentation (explained in-detail in further chapters).

If it is proven that incorporating frameshift into laggy self-driving neural nets can aid performance in the Donkey Car setup, then it opens the door to test these kinds of methods on real data and real cars. Fortunately, research in the field of autonomous vehicles is quite popular and many joint efforts between universities and companies in the private sector have popped up to aid in the revolution of self-driving cars [University of Tartu, 2021], therefore doing similar experiments to the current project on a real car certainly lies in the realm of possibility.

3.1 Hardware Setup

As mentioned previously, the car used in this work to gather expert driving data and then test end-to-end self-driving models is the Donkey Car. This subsection explains

the specific hardware and software setup in use to facilitate building self-driving models along with frameshift.

Before each drive session, there were some mandatory items on the checklist. Firstly, the Donkey Car is powered by a lithium polymer (Li-Po) battery that had to be charged up for each drive. The battery was deemed usable if the battery voltage was in the range of 7.6V - 8V. Secondly, once the Donkey Car had been turned on, a connection was established via *ssh* between the car and a computer which had a specific python environment set up with the needed modules. Thirdly, the turning radius and straight steering direction were calibrated in the mobile app for Donkey to make sure that the car's turning radius in either direction was equal and that the car could drive straight without veering left or right. Finally, one needed to check whether the driving configuration matched the desired options by reading/updating the config file, which included details such as camera resolution, frame rate, steering controller layout etc.

To simulate a laggy environment, a *time.sleep(lag)* function call, where *lag* is the desired amount of lag in seconds, was added in the class *DriveMode* of the file *manage.py* that is used to start the driving mode of the Donkey Car. This *time.sleep()* function is called every time the car starts a new frame, so in a 20Hz configuration, the *time.sleep()* could be run for 20 times a second. But if a frame lasts for more than it should have (a frame in a 20Hz configuration should last for just 50ms), then the following frames are skipped until the initial frame is processed (i.e. the steering angle is predicted passed through to the car's actuators).

3.1.1 Hardware Issues

The Donkey Car, although an invaluable tool for prototyping self-driving models, is not without its faults. During this work, multiple hardware issues propped up.

One of the main issues was how the car seemingly couldn't reset its steering angle to zero after turns well enough at one point. The car could be calibrated to drive straight correctly, but after turning left and then driving, the car's actuators hadn't reset the steering well enough, so the car still veered a bit to the left. This issue made it nigh impossible to drive clean laps, as it sometimes was able to drive straight, but sometimes not, and so it was very unpredictable. This issue was solved by using another Donkey Car, as new data collection was needed. The problem could theoretically have been solved by troubleshooting the steering actuators, but dealing with hardware issues was not in the scope of this work.

The other problem was how any configuration with a framerate over 30Hz simply didn't work. At 40Hz, the car was unable to respond to any commands by the driver. This was hypothesized to be the result of the data-capture pipeline being too time-intensive to fit into the short timeframe of 25ms for 40hz (although there may have been other factors such as possibly the capture rate of the camera) and so a framerate of 20Hz was chosen due to its round frametime of 50ms.

3.1.2 Drivable Track

Figure 9. The track on which the Donkey Car and its frameshifted models are trained and tested (the track is driven counterclockwise).

The track on which the entire project takes place is seen in Figure 9. Its shape is that of a rectangular one, as it is simply built from planks of wood, but it incorporates elements that are vital to testing the self-driving models' capabilities, such as:

- **Long straight corridors** (on the left and bottom) which test the model's ability to stay on-track and adjust its course if it's slightly steering towards a wall.
- **A narrow "snake corridor"** (on the right) that requires the car to be able to quickly turn left, then right, then left again (or just go straight through it at the right angle). This tests the model's ability to make quick and accurate adjustments to the steering angle.
- **A U-turn** (at the top) which, at high speeds, is only doable if the car is well positioned before the turn and the car steers at the maximum angle possible, testing

the model's ability to fully commit to a turn and turn on the maximum steering angle.

The car was ultimately driven on the track in a counter-clockwise direction as that was felt to be easier than driving the car in a clockwise direction, ergo, it would be easier to drive cleanly at higher speeds, therefore gathering suitable driving data would also be faster. This in turn causes the driver to have to turn left more than right (Figure 10) and it also seems that since the training tubs were collected at quite a high speed, then the steering is very discrete, meaning that the steering is either "completely left", "straight" or "completely right" with little to no in-between, leading to a possibly rough ride (steering wise).

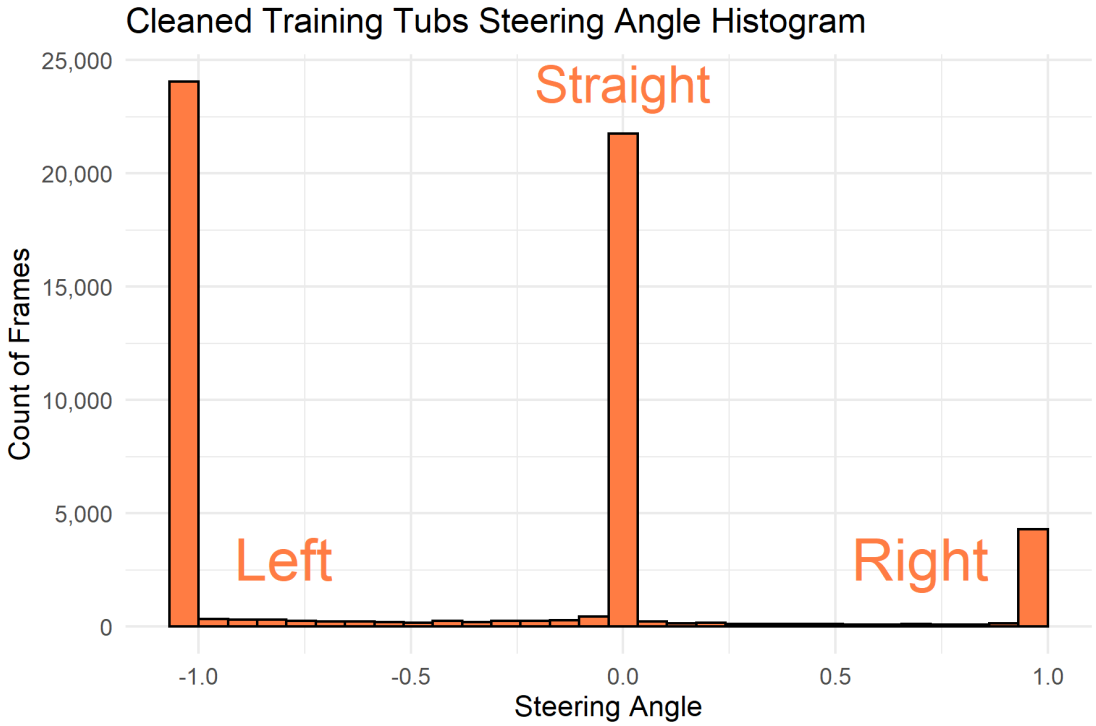

Figure 10. A histogram of the steering angles in the (cleaned) training tubs. A steering angle of -1 means "completely left" and 1 means "completely right". As evident from the histogram and the layout of the track, left turns are more prevalent than right turns.

There was also the possibility to use obstacles on the track (e.g. the toys in the middle area in Figure 9 would be placed on track), but it was decided that using obstacles in the driving would serve no purpose, as the goal of using obstacles is usually to test the generalization ability of a model by placing the obstacles in other locations during testing

time. This work does not aim to test the generalization ability and only studies the effect of delays, therefore placing obstacles differently during testing would have no point.

3.2 Data Preparation

This chapter details the process of gathering and also pre-processing data for training these self-driving models and their frameshifted variants.

3.2.1 Gathering Drive Tubs

Initially, multiple tubs of 10Hz data (with both clockwise and counter-clockwise driving) were recorded, but it was decided that new data needed to be gathered, since **a)** frameshift can only be done in 100ms increments on 10Hz data and this drastically reduces the chances to investigate frameshift's effect on driving and **b)** the data had both counter-clockwise and clockwise driving, leading to multiple situations where self-driving models trained on the data would try to turn around in the middle of the track and go the other way.

To solve these issues, the final training and test data used in this thesis was collected by driving the Donkey Car at a refresh rate of 20Hz on the aforementioned track (Figure 9) counter-clockwise for 6 sessions of approximately 30 minutes each. The test data is used for evaluating the performance of the self-driving net in an open-loop manner (explained in further chapters), meaning without giving the model control over the car.

The cleaning of the tubs (explained in further chapters) allowed to also measure the speed of the car by marking down the indices of frames whenever the car completed a clean lap (drove a lap around a fixed point on the track without making any mistakes). These indices could then be used to calculate the number of frames it took to complete a lap and, since the framerate of the camera is also known (20Hz), then it is possible to calculate the time of a lap in seconds (with a granularity of 0.05s).

Despite only being able to define throttle in terms of battery output, turning the throttle up incrementally during a drive as time went by allowed to achieve a similar car speed in different training tubs (mean of 8.33s, SD of 0.41s, Figure 11). Incidentally, the lap times in test tubs had the same mean of 8.33, but the distribution is much wider, with some of the fastest laps clocking in at around 7s. The lap times of the test tubs being more evenly spread out than the train tub laps also allows for testing whether or not the performance of a self-driving net is significantly different at speeds the net was not trained to drive at.

One should bear in mind that the speed at which the car drove on the track more resembles a race scenario and less a leisurely drive in the city. The choice of a high speed was chosen to make the effect of lag on the self-driving car more pronounced, as the distance travelled between two sequential camera frames increases with car speed, and so the reactions of the car have to be faster in order to drive cleanly. Gathering the driving

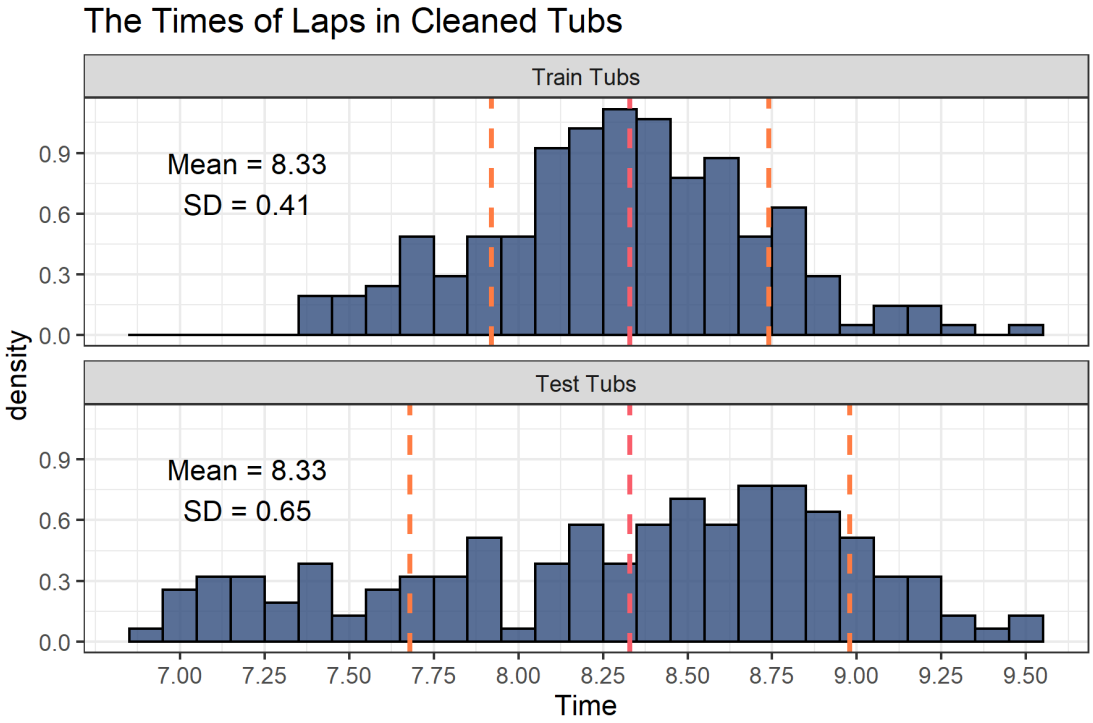

Figure 11. A histogram of the times of clean laps across (cleaned) training and test tubs. The vertical dashed lines indicate the mean lap times (red) and the distances of one standard deviation from the mean (orange).

data for clean high-speed laps is quite complicated, however, and might not even be possible for every researcher. The author of this work, however, has extensive experience with game controllers, so pushing the car to its physical limits on the track was possible.

3.2.2 Preprocessing Drive Tubs

After gathering the necessary driving data, the data is usually "cleaned" by manually removing any mistakes (e.g. situations where the operator drives the car into an obstacle) or undesirable behaviour from the footage. This can easily be done using Donkey Car's pre-packaged Manager and GUI (Figure 12). The Donkey Car's built-in software allows for both image augmentation and image transformation.

Image augmentation is a technique where input data, in this case images, is changed (augmented) to create variation in the data. The purpose is two-fold. Firstly, it can help extend the data amount when one does not have a lot of data. Secondly, it can create a model that is more resilient to variations in the data at inference time. In our case, we want to handle various lighting conditions. Multiplication and Blur features are currently

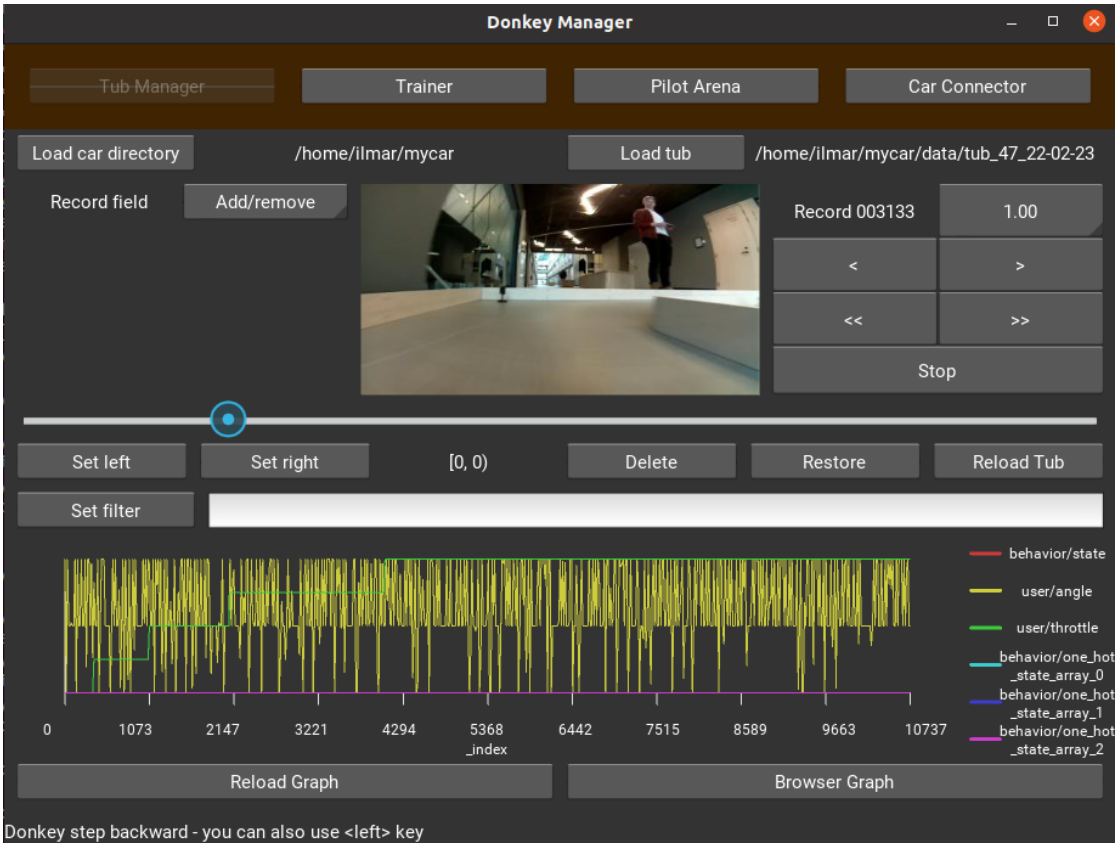

Figure 12. The Donkey Manager GUI which is used for cleaning 'tubs' (data from one drive) and training a self-driving neural network.

supported which generate brightness modifications and apply a Gaussian blur. These can be used individually or together. Augmentations are only applied during training; they are not applied when driving on autopilot [Donkey, 2022].

With **image transformation**, there is a possibility to crop and mask the images. The techniques are quite similar, they both 'erase' pixels on the image. This is done to remove pixels that are not important and that may add unwanted detail that can make the model perform poorly under conditions where that unwanted detail is different. Cropping can erase pixels on the top, bottom, left and/or right of the image. Trapezoidal masking is a little more flexible in that it can mask pixels using a trapezoidal mask that can account for perspective in the image. Generally it is practical to use either cropping or trapezoidal masking but not both. Transformations must be applied in the same way in training and when driving on autopilot [Donkey, 2022].

Driving mistakes can be removed from the data via the GUI, but also during the drive by pressing the Y-button on the gamepad: this deletes the last 100 frames of the current

drive. When one has used this feature, then the footage may, at first glance, look buggy and erratic, but in reality, there isn't a problem. However, one must take special care while frame shifting when frames have been deleted/spliced, since frame B that follows frame A in the footage might not be the original next frame, the car might be in another place/state entirely, so simply shifting frame B's steering angle to frame A's steering angle would be incorrect. Cleaning the tub of any mistakes and marking down the times of the laps usually takes up as much time as the drive itself, depending on how well the expert drove and how often they used the Y-button when mistakes were made.

Data from multiple drives/tubs can also be stitched together to increase the total number of frames that the self-driving neural network is trained on. This is highly helpful, as collecting the recommended 100,000 frames at 10Hz would require 2.5 hours of uninterrupted and clean driving, which would be impossibly difficult for the driver, but it is also practically impossible due to limitations on the car battery.

3.3 Frameshift

3.3.1 Creating Frameshifted Tubs

To frameshift entire tubs of data, helper scripts were created and uploaded to a public GitHub repository [Uduste, 2022a]. Classes *frame_shifter.py* and *data_loader.py* were designed to use the tub-loading features of Donkey to manipulate its metadata. The final script *frameshift.py* handles these classes to complete the steps detailed in Figure 13.

1. **Make a new folder** for the soon-to-be frameshifted tub.
2. **Copy all images** from original tub to frameshifted tub folder.
3. **Copy catalog manifest files** to frameshifted tub folder.
4. **Edit manifest file** to shift indexes for deleted (cleaned) frames and save manifest file in frameshifted tub folder.
5. **Edit each catalog file** to shift every steering angle (both deleted and non-deleted) the necessary number of frames and save catalog files in frameshifted tub folder.

Figure 13. The steps taken by the *frameshift.py* script to create frameshifted tubs.

3.4 Model Training

The baseline and frameshifted self-driving nets were trained using a Google Colab notebook in which the *donkey* module and *donkeycar* git repo were downloaded and set up. *TensorFlow* was also downgraded to version 2.2.0, as later versions seemingly

have issues with *donkeycar*. A Google Colab Pro subscription was also used to train these models even faster and to decrease the number of *out-of-memory* issues, as the Pro variant of Google Colab has more memory on its GPU.

Note that the models trained were automatically outputted as *.tflite* models, a framework which is designed for running machine learning applications on mobile, embedded and edge devices [TensorFlow, 2022]. This architecture was chosen as *.tflite* models were significantly faster than *.h5* models and the deployment of both model formats is equally simple in donkeycar.

Figure 14. A simplified diagram of how frameshift works in the case of frameshift = 1. Note that the green line indicates the steering angle of the Donkey Car and the steering angle in the original frame 2 has now been shifted over to frameshifted frame 1, same goes for the original frame 3 and frameshifted frame 2 etc.

In this work, a new method for data preprocessing called *frameshift* is introduced. Frameshift is simply shifting input frames in a certain time direction in relation to output labels (Figure 14). In this particular case, frameshift can be expressed by the original data at frame f_i being matched with the steering angle a_i and the frameshifted data at frame f_i being matched with the steering angle a_{i+fs}, where i is the index of frames and fs is the number of frames to be shifted. Frameshift can be either positive, meaning frames will be matched with labels from fs frames in the future and the car takes "preemptive actions", or negative, meaning that frames will be matched with labels from fs frames in the past and the car's behaviour is likely even more belated than with no frameshift.

3.5 Evaluation

There are primarily two ways to assess whether a self-driving net truly has the potential to autonomously drive: *open-loop testing* and *closed-loop testing* (Figure 15).

The easier of the methods is open-loop evaluation where the recorded behaviour of the human expert driver is compared to that of the self-driving net, without actually giving the model control over the car. One way of evaluating network performance in an open-loop manner is by looking at how far off the self-driving net's predicted steering angles are from the expert's steering angles for a particular recording. This comparison can either be done visually via the Donkey Car's makemovie tool (Figure 8) or by computationally comparing the difference of the steering angles on a timegraph. Looking at the video of predicted and expert-caused steering angles can give qualitative insight into the biases the model has and help troubleshooting, while the analytical approach can give good quantitative info.

The main metric for comparing predicted and expert-generated steering angles is the mean absolute error (MAE) of the steering angle, although this metric is not without its issues. As the model is not actually in control of the car, this metric does not directly measure the driving ability of the model, nevertheless, it has been shown that MAE and actual driving ability correlate somewhat ($r = 0.6$), better than mean squared error (MSE) with actual driving ability [Codevilla et al., 2018].

Despite its shortcomings, many models in the end-to-end self-driving nets paradigm are evaluated only in an open-loop manner [Hecker et al., 2018, Zeng et al., 2019, Hecker et al., 2019], as it is computationally easy, cheap and time-saving compared to the alternative.

Closed-loop evaluation, however, directly evaluates the performance of a driving model in a simulated or realistic driving scenario by giving the self-driving net control over the vehicle (Figure 15). This approach is obviously more costly and time-consuming than open-loop evaluation, but it is a better indicator as to whether a self-driving net can actually autonomously drive.

There are many metrics used in closed-loop testing (such as the average distance between infractions [Bewley et al., 2018] or fraction of distance travelled towards the goal in a successful manner [Codevilla et al., 2018]), however, the ones in use in this thesis are infraction and intervention count. Infractions are whenever the self-driving net causes a mistake and the car collides with something (usually the wall of the track), but still keeps driving the lap. Interventions are counted whenever the car comes to a halt and cannot drive anymore (e.g. due to a collision) and therefore requires human intervention to go on. These situations are counted per lap: each lap was either driven cleanly, had an infraction or had an intervention and so these clean, infraction or intervention labels are mutually exclusive. The counts of these situations are then averaged over multiple laps to get an estimate of how well/cleanly the self-driving net drives on the track.

A novel metric called *driving score* combining these infraction and intervention

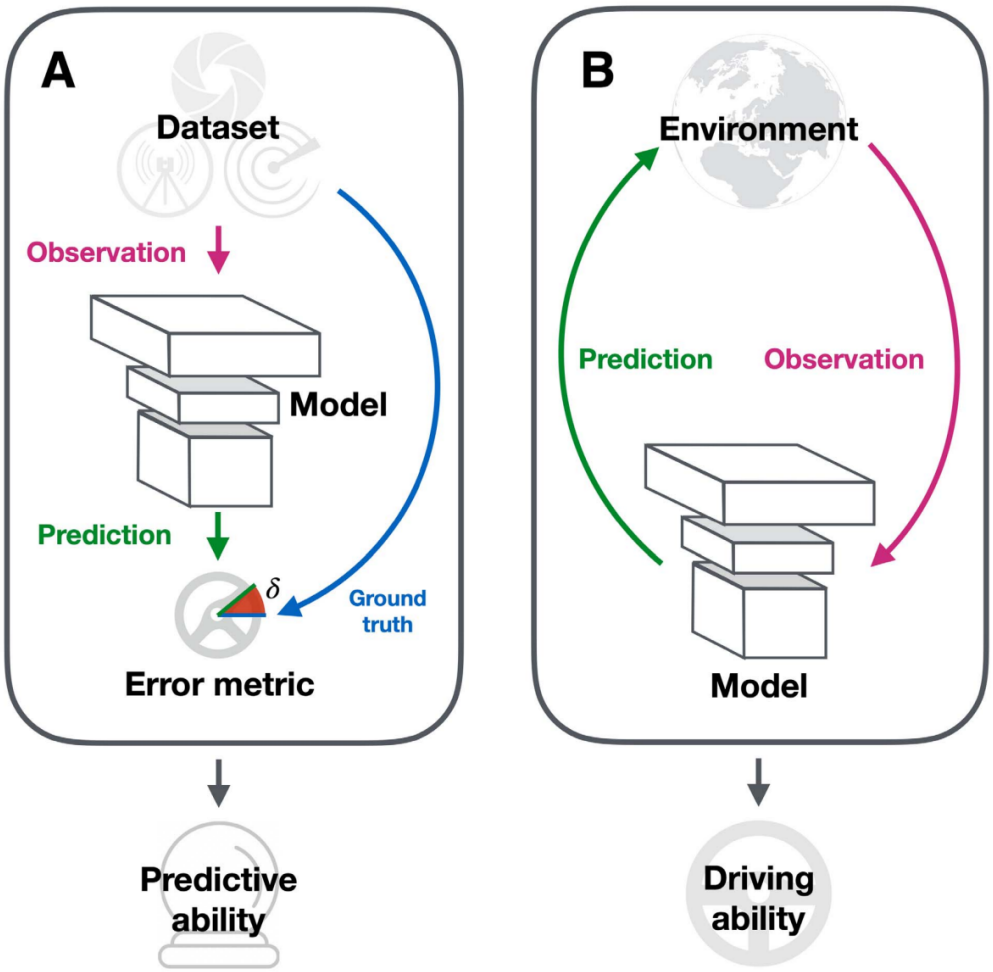

Figure 15. Open and closed-loop evaluation. **A:** open-loop evaluation uses a part of the original data set to evaluate the model. Similarity between predicted and ground-truth values is measured. No actual driving is done. **B:** closed-loop evaluation deploys the model in an environment. The model's predictions are used as driving actions. The resulting driving behavior is observed and quantified. The figure and caption are from [Tampuu et al., 2020].

counts is used in this thesis. The formula is as follows:

$$DS = 10 - 2.5 \times (IFL \times 1 + ITL \times 4)$$

where DS is the driving score, IFL is infractions per lap ($IFL \in [0, 1]$) and ITL ($ITL \in [0, 1]$) is interventions per lap. This formula scales the values of the driving score to be from 0 to 10 and heavily penalizes interventions while also slightly penalizing infractions. A score of 10 refers to perfect driving (without any interventions nor infractions in the measurements) and a score of 0 means the car couldn't drive a single lap without intervention.

Considering and using the aforementioned aspects of evaluation, some conditions can be set in order to test and conclude hypotheses. Firstly, the self-driving models trained with different parameters and lag are consistent in their behaviour i.e. the models don't have much variance in their behaviour and faults. Secondly, there should be clear and explainable differences between models trained with certain parameters and certain amounts of lag. If these two conditions are fulfilled, then it is possible to accept or reject hypotheses regarding the effect of lag on driving performance.

4 Experiments and Results

This section details the experiments and their results. The results in *.csv* form, plots and notebooks used in this section are stored in the publicly available project GitHub repository [Uduste, 2022a].

4.1 Baseline Model Selection

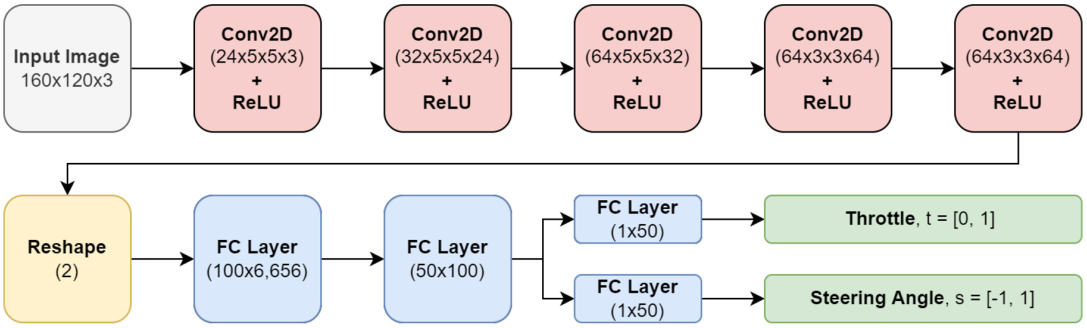

Figure 16. The architecture of the Keras Linear neural network which is the default self-driving neural network for DonkeyCar. This network architecture was chosen as the basis for the baseline and the frameshifted models of the experiment. Note that a constant throttle was used instead of the throttle prediction in the testing of frameshifted models, although it was still trained as a feature.

The default model for these self-driving cars in Donkey Car is a linear Keras model that uses one neuron to output a continuous value via the Keras Dense layer with linear activation (Figure 16). The official documentation reports that the benefits of this model type include being able to steer smoothly and performing well in limited compute environments (which the Raspberry Pi powered Donkey Car inevitably is), while its main drawback is it not learning to throttle well [Donkey, 2022]. Since, in this particular experiment, the throttle is constant anyway, then this is not a problem.

The baseline model that all frame-shifted models are compared with are models that don't have any frame shifting (0ms). Different combinations of image augmentation/-transformations were also tested for frame-shifted performance.

4.1.1 Image Augmentation and Transformation

To get an idea of how much the Donkey Car images should be cropped, some images were cropped and visualised manually in *eda1.ipynb*. It seemed that, for a resolution of 320x240 pixels, a suitable crop would be 100 pixels from the top (Figure 17), as that did

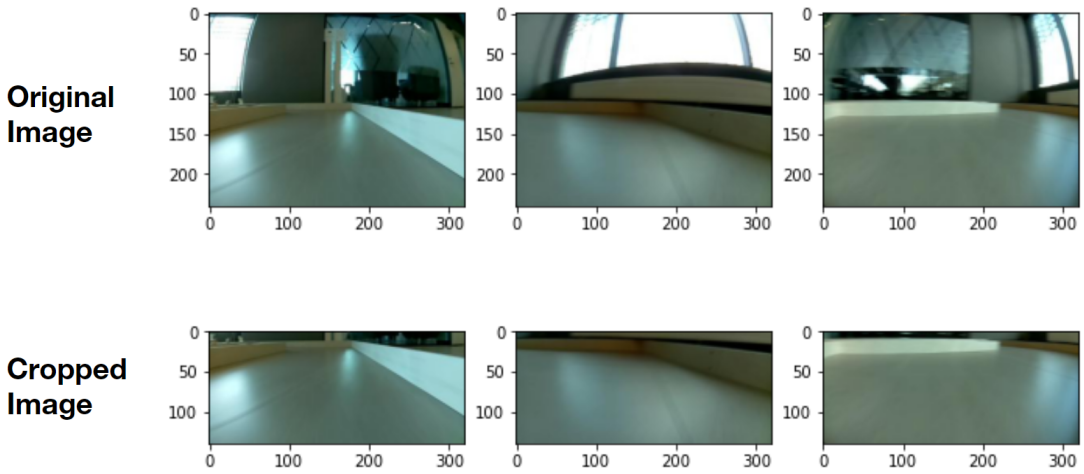

Figure 17. Original 320x240 RGB images from the Donkey Car compared to their cropped variants (100 pixels from the top of each image).

away with most of the seemingly useless information from the image, such as elements off the track, windows, people etc, while retaining the information concerning the track (e.g. track walls, track floor). When testing out the lower-resolution 160x120 images, a suitable crop would then proportionally be 50 pixels.

The cropped variants of models were also trained and evaluated, but it seemed that the feature simply didn't work on the hardware configuration present, as the car constantly drove into walls when driving with the cropped models. The point of a cropped model is to discard useless information from the image and therefore reduce model training, prediction time and increase the ability of the model to generalize, but since 1) the low-resolution-full-image model already had very good baseline performance with very fast prediction times (average of 25ms) and 2) the track always stays the same, meaning that the walls and ceiling in the background are equally valid as features for deciding when to turn as the walls of the track. If the track changed, then the upper portion of the image would possibly lead the model to false signals, but we always drive on the same track, so the background is always the same. Because of these facts, pursuing cropped models was deemed unnecessary.

Brightness augmentation was deemed unnecessary after testing, as the baseline low-resolution model did not seem to have any issues driving the track and the testing was done in similar lighting conditions throughout the work, negating the need for different brightness in the training data. Using brightness augmentation also significantly added to the training time of each model without adding any visible benefits, so no image augmentation nor transformation was used at all in the baseline model.

4.1.2 Training Frameshifted Models

Finally, a model was built with the default parameters (160x120 resolution and no image augmentation nor transformation) for every frameshifting value possible, resulting in 7 self-driving models with the following frameshifts: -100ms (negative frame shifting, predicting the past), -50ms, 0ms (no frame shifting), 50ms, 100ms, 150ms, 200ms. Frames can be shifted by a minimum of 50ms at a time, this is due to the configured 20Hz framerate of the training data, making for a 50ms frametime. The models are named FS -2, FS -1, FS 0, FS 1, FS 2, FS 3 and FS 4 respectively.

The number of frames in the training set totalled at 46,620, while the validation set had 11,655 frames. Each model took about 12 ($\pm$ 3) epochs of training time, depending on when the validation loss hadn't gone down for many epochs. The validation loss was usually around 0.06 and training took around 15 minutes for each model.

4.2 Open-loop Performance of Frameshifted Models

Before deploying these frameshifted models on the track and evaluating their performance in a closed-loop fashion, it is sensible to first do a sanity-check and rate the performance of these models in an open-loop way. This allows to understand if the models are capable to imitate human at all and discover major issues, if any. Also, a drop in prediction accuracy would reveal to what extent predicting future commands is harder than predicting non-shifted labels.

Each model was used to predict the steering angle based on the frames in the cleaned test tubs (which consisted of 45,459 frames) and then this prediction was compared to the ground truth. Note again, that open-loop performance might not be indicative of real-world performance, especially in the context of frameshifted models, as there is no lag in open-loop and yet lag is the very thing which frameshifted models fight against. It was attempted to avoid such issues by setting the ground truth for frameshifted models to be the steering angle that corresponds to the frame that the frameshifted model is predicting in the future or the past. Accurately simulating lag, however, is not feasible in the scope of this work and so the following open-loop results might only somewhat correlate with real driving performance, as even with minimal lag, there is weak correlation between open-loop and closed-loop performance [Codevilla et al., 2018].

4.2.1 Steering Angle Distribution

Although the training data had very discrete and extreme steering angles, with little to no values between 'completely left', 'straight' and 'completely right' (Figure 10), the self-driving models predict steering angles in a much more uniform and continuous fashion (Figure 18). These histograms show many peculiarities in the frameshifted models.

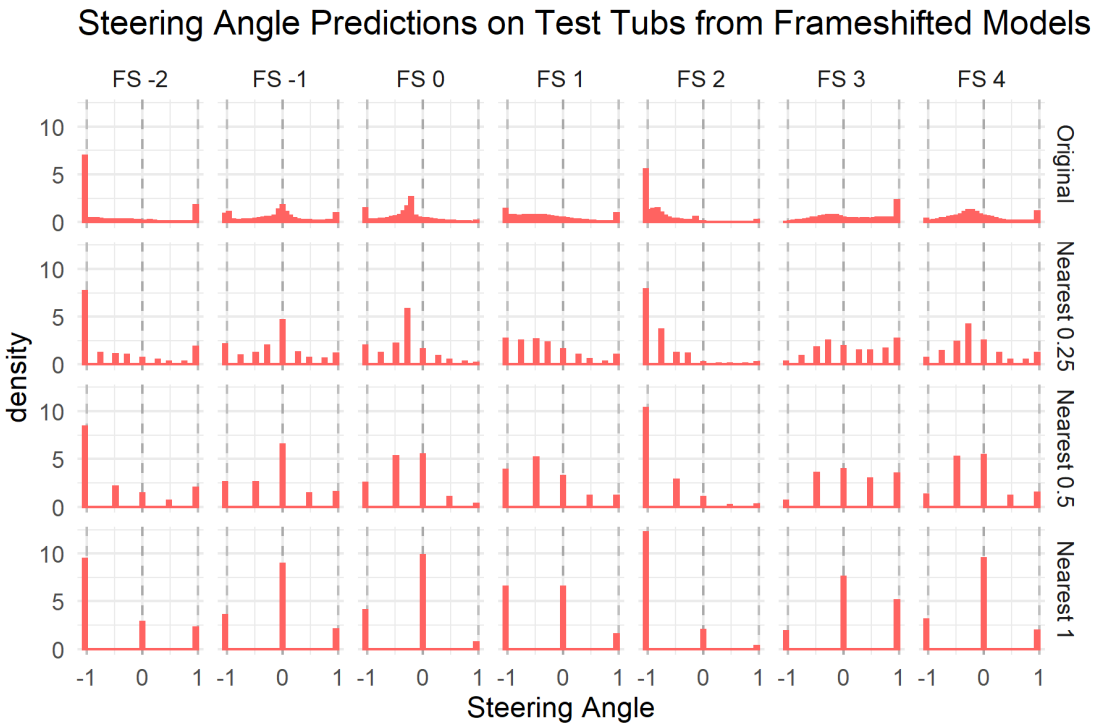

Figure 18. Histogram of steering angle predictions on test tubs from frameshifted models (FS). The number behind FS designates the number of frames shifted. Different rows also show different rounding values, the first being the original value, the second being the steering angle value rounded to the nearest 0.25 etc.

Firstly, even the standard non-frameshifted model (FS 0) seems to drive straight more than left, while the ground truth in test data includes more left angles than center angles. This also holds true for FS -1, FS 3 and FS 4. FS 3 in particular exhibits unusual behaviour, as it predicts turning right quite a bit more often than left, even though the track is heavily left-turn slated (Figure 9 and 10).

Secondly, none of the models come close to emulating the discrete nature of the expert driver's steering angles. Only when rounding the predicted steering angles to the nearest 0.25, 0.5 or 1 do the histograms start to look like the training data. Purely by looking at the shape of the "Nearest 1" row histograms, one could say either FS -2 or FS 1 perform the best, since they most resemble the training data (FS 2 doesn't, as it has almost no right turns). One should keep in mind, though, that this comparison is done against human recordings and it is not possible to accurately predict whether these models would actually perform better than the human driver or not, this comparison is merely a gauge of how human-like the models' behaviour would be.

The uniformness of the steering angle distributions is worrying, however, since driving the track at high speeds requires full left/right turns sometimes and it could be that not steering all the way may cause these models to crash into the walls of the track. It may be that a (-)0.5 steering angle can also facilitate a full turn and clear these corners, this would lead a badly under-predicting model to succeed if it gets the direction correct at the right moment. Nevertheless, for a more complicated case, e.g. real-world driving, it is a serious problem if networks always under-predict in terms of magnitude and never turn sharp.

4.2.2 Error Metrics

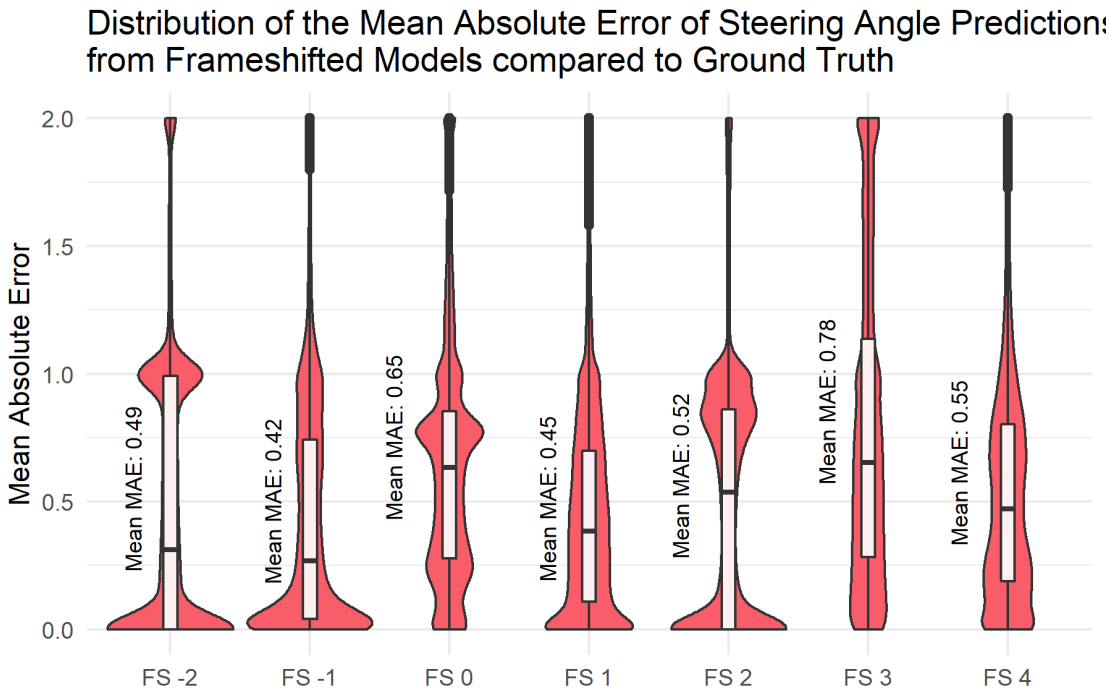

Figure 19. The distribution (violin and boxplot) along with the means of the mean absolute error (MAE) of steering angle predictions from frameshifted models compared to the ground truth (expert driver angles).

Looking at the mean absolute error between predictions and the ground truth further reveals the intricacies of frameshifted models (Figure 19). By looking at the medians and means of the mean absolute errors across the test tubs, it might seem by the mean MAE of 0.49 that FS -2 might be one of the better frameshifted models, but one has to

take into account the fact that open-loop evaluation is based on evaluating each frame separately. This means each frame is treated as being separate and independent from other frames, while in reality, it's the polar opposite. The mistakes that a model makes on the track are cumulative and any kind of situation where the model has a MAE of 1 means a situation where it should turn in one direction, but it isn't doing so and this might cause the car to crash and end the drive.

In evaluating model performance based on MAE, a more indicative measure of performance than the mean are the upper quartiles (75%, the upper bound of the boxes in Figure 19), since that indicates the minimum error that happens 25% of the time. If the minimum error is large enough, it just might happen that every 4th frame (25% of the frames) might cause the car to drive incorrectly.

According to the previous mentality of evaluating MAE, the FS -1, FS 0, FS 1, FS 2 and FS 4 models seem promising. A case could also be made for FS -2, but the upper error size quartile being at roughly 1 is worrying. What is peculiar is that predicting the past (FS -1) or predicting the future (FS 1) have a smaller mean absolute error than a regular non-frameshifted model. In fact, all frameshifted models besides FS 3 have a smaller median MAE than the non-frameshifted model, but the upper quartiles for the frameshifted models besides FS -1 and FS 1 are higher than the regular.

One interesting find is that the error metric for FS 3 is far higher (worse) than FS 4. There doesn't seem to be an apparent reason why a 200ms frameshift would be better than a 150ms frameshift, so this might just again show how open-loop evaluation is not the best method of measuring self-driving performance.

A rule-of-thumb among Donkey Car enthusiasts seems to suggest that a MAE of over 0.5 indicates a quite poor real-world performance, but this heuristic is tested in the following chapter.

While the mean absolute error graphs indicate the average error a model makes, there is no indication as to what kinds of mistakes are made. This is why the predicted steering angles are also visualized in a confusion-matrix-like fashion in histogram form (Figure 20) and in a confusion matrix form (Appendix II). The ground truth steering angle is rounded to -1, 0 and 1 and this is possible due to the very discrete nature of the expert's driving.x

The confusion matrices show how the self-driving models behave in situations where they should be turning a specific direction or driving straight. For example, it seems that turning right is not a problem for any frameshifted model, as the models mostly or only predict turning right when they should turn right. FS 0, on the other hand, for some reason cannot handle these right turns and it predicts turning left more often than right in cases where it should be turning right. It is weird that this discrepancy is present, especially considering that the FS 0 model was the model that performed well on the track in baseline model selection and it had no issues turning right there, but as previously stated, the model is not in control of the car in open-loop evaluation.

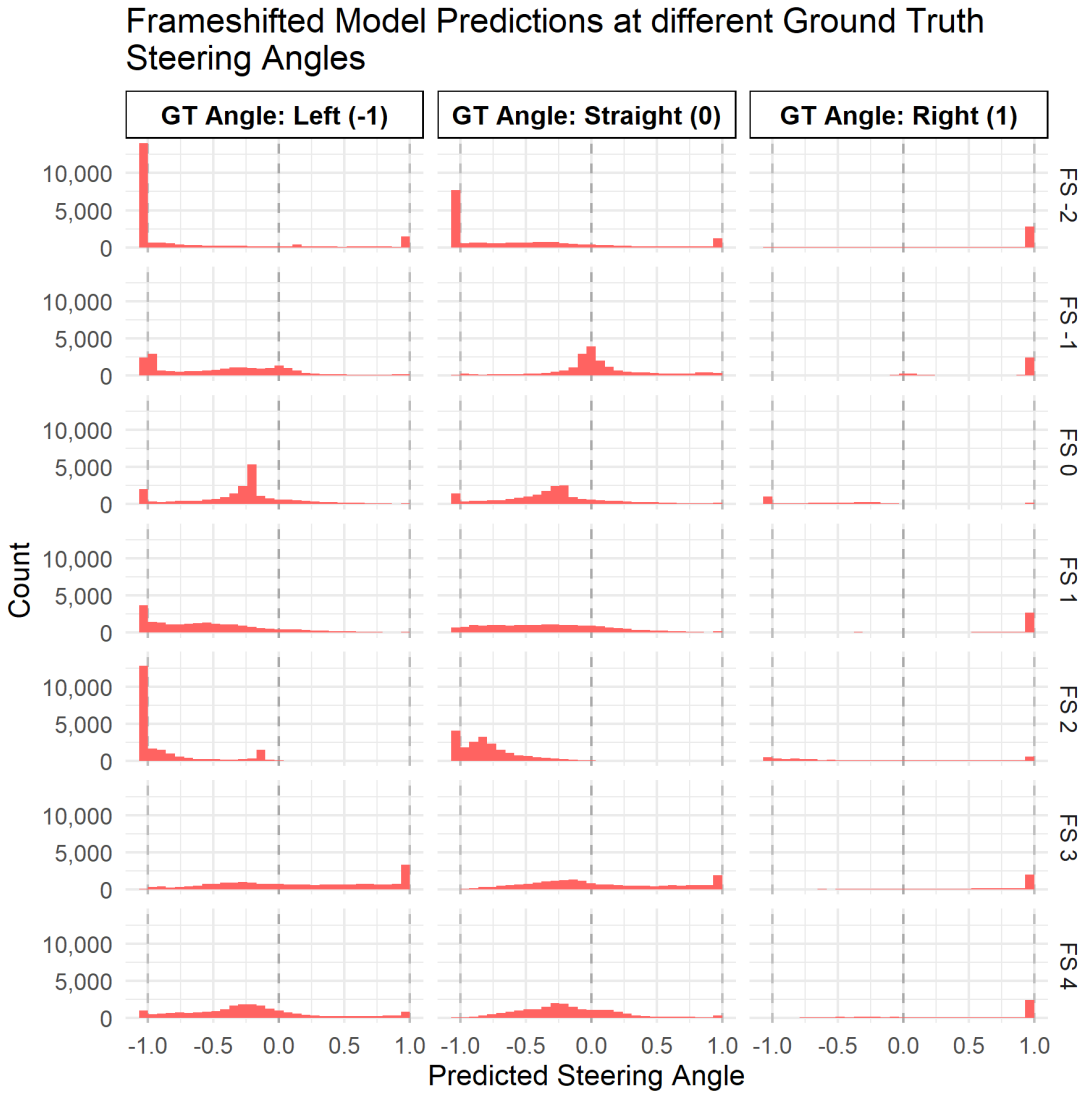

Figure 20. Histograms of predicted steering angles according to groups of the ground truth steering angle. The left column indicates predicted steering angles when the ground truth is -1 (left), the middle column for ground truth steering angle 0 (straight) and the right column for ground truth steering angle 1 (right).

Only FS -1 and FS 1 models somewhat resemble the discrete nature of the training data. They still exhibit problematic behaviour, though, as FS -1 drives straight too much when it should turn left and FS 1 turns left too much when it should be driving straight. It is possible, however, that these problems are all smoothed out during closed-loop testing.

FS 3 and FS 4 models seem like they barely predict the directions correctly, as both models barely turn left (in fact, FS 3 predicts turning right the most when it should be turning left). This is highly concerning, as the track is driven counter-clockwise and therefore consists of mostly left turns. FS -2 and FS 2, on the other hand, seem to have somewhat the opposite problem: they turn left too much. FS -2 seems like it barely drives straight when it should (although it very accurately predicts when the car should be turning left) and FS 2 exhibits the same issues. All in all, these graphs indicate that the models perform badly, at least in an open-loop setting. In real-world testing, the results might be different.

4.2.3 Timegraph

By plotting the predictions of each model against the ground truth (where frames have been shifted according to frameshift) on a time-scale, it is possible to visualize the behaviours of different frameshifts and this is done in Figure 21. The visually rectangular steering of the author serves as stark contrast to the continuously valued steering of frameshifted models.

The models that most closely follow the ground truth are FS 1 and FS -1, with FS 1 predicting turns slightly before the ground truth and FS -1. The models both also perform well in places where sharp turns are needed, as they almost turn fully to the left or to the right in these situations. FS -2 exhibits similar behaviour, but it doesn't seem to do these sharp turns that clearly, as the model oscillates between steering angles quite a lot.

The non-frameshifted model seems like it doesn't follow the ground truth line at all, but perhaps it can manage to drive the track correctly, even if it seemingly doesn't keep the steering angle at extreme enough angles for an extended amount of times to pass certain corners.

The FS 2 model barely turns right and this could be a problem in certain corners. Based on this timegraph, it wouldn't likely even finish a single lap, since it would fail a right turn (by seemingly not turning right when it should). FS 3 has a similar problem by constantly turning in the wrong direction (hence the high mean MAE) and FS 4 doesn't seem to fare much better.

In summary, open-loop evaluation indicates that FS -1 and FS 1 models might work well on the track, while FS -2, FS 2, FS 3 and FS 4 have poor predictive performance. The non-frameshifted model, for some reason, evaluated poorly in the open-loop tests, but it is highly unlikely to perform accordingly bad in real life, as this is the baseline model type that was chosen based on its reported relatively mistake-free driving at high-speeds.

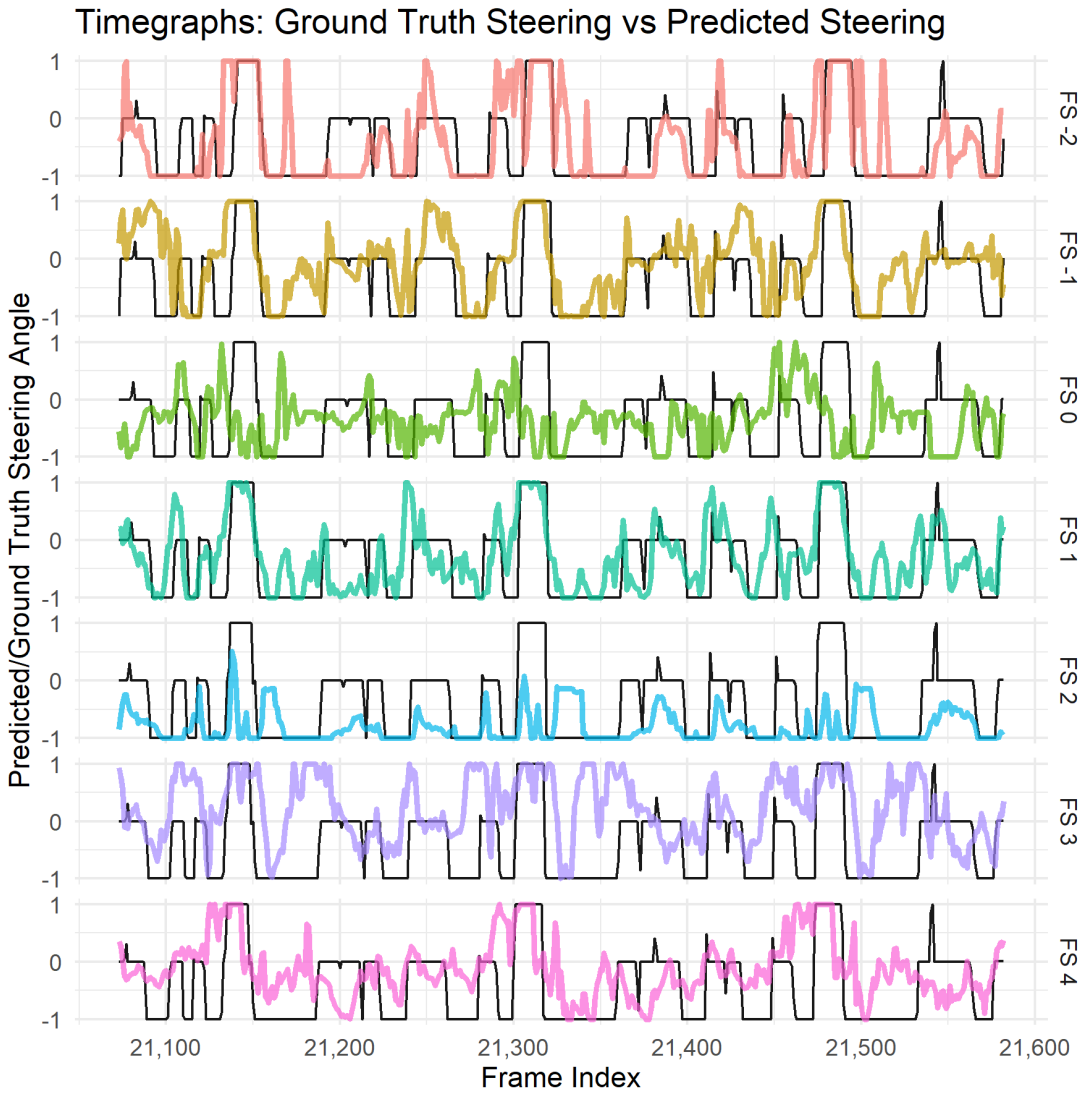

Figure 21. Timegraphs of 3 sequential clean laps. The black lines on each row represent the ground truth (expert driver angles) and colored lines represent model predictions (-1 is a left turn and 1 is a right turn). Each row is a separate frameshifted model.

4.3 Closed-loop Performance of Frameshifted Models

The one baseline and six frameshifted models were all tested on the track at different artificial delays and the number of infractions and interventions per 25 laps were marked down. An example video of the baseline model driving the track can be seen on YouTube [Uduste, 2022b].

During the testing, it quickly became clear that not every model can be driven at the same speed, e.g. models with a higher frameshift tried to turn corners too early at slow speeds, so they had to be driven at higher speeds (likely because the turn radius of a car has negative correlation with speed and the decision making frequency in relation to space is lower at high speeds.). Negative frameshifts, however, started to turn corners with a significant delay compared to the baseline models, and therefore had to be driven at slower speeds to complete clean laps. To give each model a fair chance, the testing included finding the fastest speed at which a model could comfortably drive. Fortunately, this fastest speed was visually easy to estimate from the steering behaviour of the models. When testing these cars, the acceleration to the top speed had to be handled by a human or done in a straight corridor, as the models had only been trained on frames at a constant speed and they possessed no examples in the training data of how to steer at speeds lower than their top speed.

For every model and artificial delay combination, the maximal speed at which the car was able to drive clean laps and the time the Donkey Car took to infer predictions and send the signal to the wheels for each frame was marked down. The results of the experiments are available in text form in Appendix I and in *.csv* form in the aforementioned GitHub repository.

4.3.1 Performance at different Delays

The first key experimental result is confirming what one would expect: the more lag (artificial delay in this case) there is, the "worse" the car drives. "Worse" in this case meaning less able to drive the car at the trained speed. This is to be expected as decisions arrive too late and there are fewer decisions per second. When looking at the performance of FS 0 across different artificial delays (Figure 22), it seems clear that the more lag there is, the slower the self-driving model is able to drive clean laps on the track. This rule holds true for all the frameshifted models as well. Intuitively, the lower speed counteracts the effect of lag to some degree - when advancing slowly, the model has more time to initiate a turn before being too late (too close to the wall).

It should be noted, however, that there might not be much of a difference between situations of 0 artificial delay and 25ms artificial delay (as is the case with the FS 0 model), as the frametime in 20Hz configurations is 50ms and the average prediction time of each frame for these self-driving nets is 25ms, then with a 25ms artificial delay for each frame, barely any frames need to be skipped at all. This similarity in results,

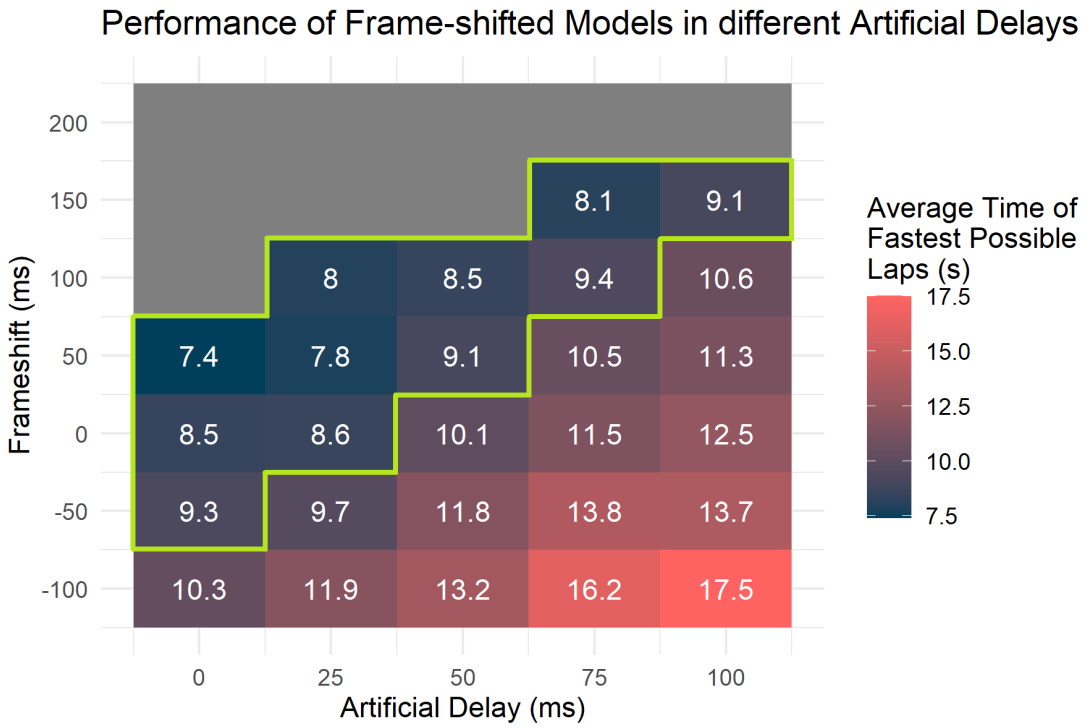

Figure 22. The connection between Frameshift, Artifical Delay and the average time of fastest laps. The number in the tiles designates the lap time and the gray areas indicate combinations where the car was tested on that artificial delay and that frameshift, but it failed to drive any clean laps, so measuring the time of a lap was not possible. The tiles highlighted with the green border are combinations that could drive at a human-like speed, defined as less than the mean lap time of the training data plus two standard deviations (faster than 9.4s).

however, would most likely not be present if the framerate of the camera and/or the prediction time of each frame were higher (since that would lead to frames being skipped i.e. there are fewer decisions made per second).

A second key finding here is that the regular non-frameshifted model is actually beaten in terms of speed by frameshifted models in both no artifical delay and artificial delay included situations (corresponding to higher and lower compute environements). At 0 artificial delay, the FS 1 model is able to drive quite a bit faster than the non-frameshifted model (7.4s vs 8.5s) and this is thanks to its preemptive sense to start turning corners earlier. The FS 1 model shows performance similar to that of a skilled expert driver with its speed and rather clean driving and this is also present at an artificial delay of 25ms. FS 2 and FS 3 models are also able to drive clean laps faster than the

baseline model, although this is in artificial delays of 25ms, 50ms and 75ms, as in zero artificial delay environments they started to turn corners too early and a suitable speed for them to successfully turn corners and yet not crash into walls could not be found.

In fact, there are quite many frameshifted models that are able to drive at human-like speeds at artificial delays where the baseline model could not (highlighted in green in Figure 22). This shows how frameshifted models could be used to circumvent possible performance constraints when driving in limited-compute environments. Just the fact that even at 0 artificial delay, the slightly frameshifted FS 1 model performs better than the baseline model is justification enough to further investigate this approach in further (and possibly larger) experiments.

A peculiarity in FS 3 and FS 4 models came about during every test with them and it was that they lacked the ability to make minor corrections in their steering. If the car was slightly steering off-course to the left, then a regular self-driving net would make a slight correction and turn to the right for a split-second. But these FS 3 and FS 4 weren't able to make these micro-corrections regardless of the amount of delay and if they started veering off to a wall, they just crashed into it. Adjusting for micro-corrections at time $t + FS$ in the future based on frame at time t is, intuitively, a rather difficult task which the FS 3 and FS 4 models struggle with. This is also one of the reasons why these models had relatively poor driving score (discussed further in the next subsection) or didn't drive any clean laps at all.

4.3.2 Lag Inconsistency and Driving Score

The performance of frameshifted models at high artificial delays (>50ms) is impressive, however, it is quite inconsistent. The frameshifted models require higher speeds (since they pre-emptively turn corners), but the higher speed causes the distance travelled between frames to be higher and the high artificial delay makes the car skip the steering angle calculation for critical frames sometimes. These *frameskips* may lead the car to not turn at necessary moments and that will cause the car to not only hit the wall (defined as an infraction), but crash and be unable to continue without external help (defined as an intervention).

This is why the driving score (marked in Appendix I) for frameshifted models at high artificial delays is usually poor, as the driving score metric heavily penalizes interventions which are more common at higher speeds (found across all experiments) and assigns a smaller negative score for infractions (more common at lower speeds). It is also the reason why it is hard to quantitatively gauge driving performance across different speeds, as it is difficult to tell the whole story of how a car performs at different speeds at different parts of the track with just one metric.

One may consider more qualitatively investigating driving performance in the future, but in this work, the best indicator of whether frameshift model contributes positively to the driving performance of a model is its ability to drive at human-like (or even faster)

speeds in conditions that the baseline model can't, such as laggy conditions. This is precisely what was shown with the previous chapter.

It should be mentioned, however, that negative frameshifted models (FS -1 and FS -2) were very clean in their driving, in some cases pulling off 25 clean laps in a row without any infractions nor interventions, albeit at slower speeds than the baseline model. A hypothesis to explain this peculiarity would be that at slower speeds and at a negative frameshift, the self-driving net isn't so extreme in their actions (as the expert driver was in the training data) and could drive more smoothly by having somewhat delayed reactions. Because of the slow speeds, there are also more decisions made per distance unit travelled. Perhaps adding negative frameshift to the autonomous network would help if one were to aim for the smoothest self-driving experience possible, but this remains to be seen in future testing.

4.3.3 Lag Delta

In this work, the time it takes for the whole self-driving pipeline to process a frame, predict the suitable steering angle and then pass this steering angle to the steering actuators is called *signal-to-steering time* (STS). This STS metric also takes artificial delays into account, so it is suitable as a measure of how long the entire pipeline takes for each frame.

One would ideally want to build self-driving models that are frameshifted specifically according to the amount of STS and this difference in time can be defined as a lag delta $\Delta l = STS - FS$, where Δl is the lag delta, STS is signal-to-steering time and FS is frameshift. Theoretically, a lag delta of 0 would mean that self-driving net would predict a suitable steering angle for a frame f_t at time t, then it would pass along the steering angle that was in the training data for f_{t+STS}, meaning the steering angle prediction would match the particular time the prediction signal would reach the steering actuators.

With this lag delta metric, it is possible to see the connection between the average time of fastest possible laps, STS and the frameshift applied to combat STS (Figure 23). It turns out that lag delta has a negative correlation with the time a model can drive a clean lap at the track, meaning that lower lag deltas lead to higher timed (faster) laps and higher lag deltas lead to slower laps.

Models with a negative lag delta (frameshift was bigger than STS) were often able to drive faster than the average lap time in the training set (8.33s) and the most impressive frameshifted model that was capable of clean laps even faster than the fastest lap in the training set had a lag delta of -26.3ms (zero artificial delay, FS 1). It would seem as if self-driving nets should aim to have a slightly below 0 lag delta in order to maximise the capability of driving at human-like speeds. However, one should bear in mind that many of the data points present in Figure 23 with a lag delta below 0 represent situations where there is high artificial delay and a high frameshift, leading to lag inconsistency (as talked about in the previous chapter) and fewer decisions made per time unit. This

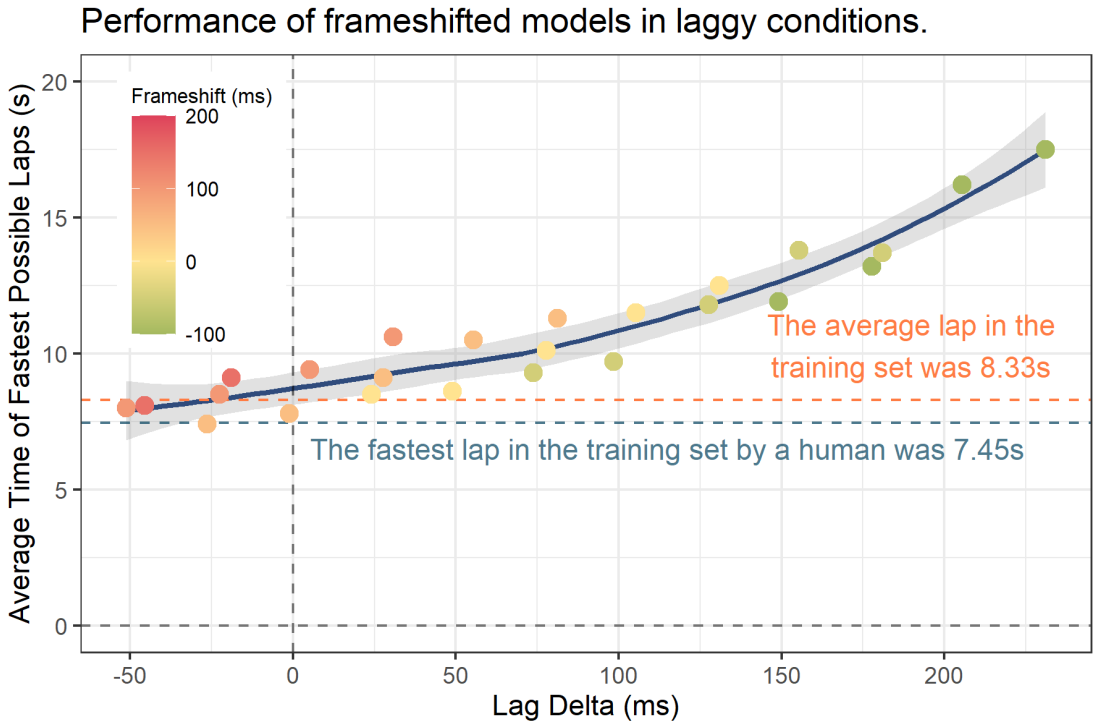

Figure 23. The connection between Lag Delta, the average time of fastest laps and frameshift. The lag delta on the x-axis is calculated by the formula $\Delta l = STS - FS$, where Δl is the lag delta, STS is signal-to-steering time and FS is frameshift.

claim about a negative lag delta having a positive impact on driving performance should definitely be investigated further before taken as a heuristic for self-driving models.

Models with a lag delta over 75ms could not drive cleanly at human-like speeds (lap times under 9.4s) and models with a lag delta over 150ms (negative frameshift) were only able to drive at speeds such that the car almost didn't cross the force threshold required to make the car move in certain situations (e.g. during extreme turns). This fuels the belief that environments where the signal-to-steering time is already over 100ms require frameshifting in order to maximize driving likeness to that of the training data (the driving of an expert). In this work, however, the training data was gathered at a very high speed, which might mean that this frameshift is practical only in extreme situations where the distance travelled by the car between frames is too great for the self-driving net to handle. Therefore, frameshifted models in every-day conditions (e.g. city traffic for regular cars) should also be tested to see whether this would affect driving performance.

4.4 Explaining Open-loop Evaluations Through Closed-loop Performance

Some of the peculiarities discovered in the open-loop evaluation are explainable through the closed-loop performance of the self-driving nets.

Firstly, it was hypothesized that the uniformness of the steering angle distributions (Figure 18) compared to the expert's driving might cause the models to not handle the car in steeper corners, but this proved not to be the case. In fact, the self-driving nets excelled in handling and sometimes performed even better in these turns than the expert driver in the training data. This might be explainable by the visuals in the timegraph (Figure 21), as the models seemed to still predict the necessary boundary steering values in the necessary places.

Secondly, from the MAE distribution (Figure 19), it seemed that FS 4 would perform better on the track than FS 3, as its mean and upper quartile were lower than those of FS 3, and even similar to those of FS 1, a model which performed very well. But FS 4 was unfortunately unable to drive a single clean lap, as it started to turn corners too early at any speed and artificial delay setup.

The differences in the shapes of error distributions of models also did not always manifest in some kind of completely different behaviour. For example, FS 2 was not drastically different from FS 1, as their error distributions would suggest. Also, FS -1 and FS 1 had very similar MAE distributions, yet performed quite differently: FS 1 was able to drive quite a bit faster than the baseline model and FS -1 was not. In general, MAE and MSE were not indicative of performance on the track and were, in fact, quite deceptive, as was already written in previous literature [Tampuu et al., 2020, Codevilla et al., 2018].

Finally, the timegraph (Figure 21) seemed to be the most indicative evaluation tool of the behaviour of self-driving nets on the track, although it is still not without its issues. The intricacies of FS 1 and FS -1 were quite understandable from the timegraph, but FS 0 and FS 2 had quite nonsensical predictions on the timegraph that definitely did not represent their real-world performance. There is also the issue of the car not actually being in control of itself during open-loop evaluation, so it doesn't actually get into situations which it should by its own predictions. This all means that ultimately, the open-loop evaluation only tells a small part of the story and self-driving nets should definitely be tested in a closed-loop manner to actually test their performance.

5 Conclusion

In this work, end-to-end self-driving networks were trained based on imitation learning methods and the performance of these networks in laggy conditions was evaluated. A novel method called frameshift was proposed to fight against this lag and improve the performance of the autonomous car. The essence of a frameshift is to shift the input data in relation to the output data along the time axis. Here, frameshift was used by shifting the input for the self-driving net, camera frames, in relation to the output, the steering angle of the car.

The training data of expert driving behaviour was gathered with the goal of having a high speed of the car, the driving more resembling racing than driving in the city, so that the effect of lag on the performance of the self-driving net would be more pronounced. Seven self-driving models with different frameshift times in 50ms increments (due to the 20Hz framerate of the car's camera) were trained based on this training data: two negatively frameshifted models (-100ms, -50ms), a baseline non-frameshifted model and four positively frameshifted models (50ms, 100ms, 150ms, 200ms).

The performance of these frameshifted models was compared to the baseline model in both an open-loop (analytical) and closed-loop (testing the car on the track) fashion and lag was simulated by adding an artificial delay into the driving module of the car.

In the open-loop evaluation, the slightly frameshifted models had more favorable error metrics than the baseline model and their steering angle distribution also more resembled the behaviour of the expert driver than the baseline model. The discrepancy between the predictions of frameshifted models and the ground truth was still large, though, and some frameshifted models were predicted to have terrible performance on the track due to not predicting left turns (even though the track is driven counter-clockwise, leading to a majority of left turns) or not predicting sharp enough turns. The open-loop behaviour of these models was not completely interpretable without testing them in real-life conditions, however.

The performance of different self-driving models on the track showed that the more lag there is, the worse the car drives, but the effect of this lag can be mitigated using frameshift with great success. Even in environments of zero artificial delay, frameshifted models were shown to have better performance than non-frameshifted equivalents, although there was a limit as to how much frameshift could be added to a model before it could not cleanly drive in any conditions (e.g. the 200ms frameshifted model did not pass any lap in any condition). It was also clear that adding lag to the driving pipeline causes the car to make fewer decisions per time unit and this hinders the car's ability to make micro-adjustments in the steering, leading the car to make mistakes quite often.

The need for closed-loop testing of self-driving models cannot be overstated, as the discrepancies between open-loop and closed-loop performance were significant. For example, open-loop evaluation showed that FS 4 would be a better self-driving model than FS 3, but this was proved to be false in real-world testing. The model is not in

control of the car in open-loop testing and therefore, open-loop testing is not a valid indicator of how a car performs in real life. This makes the usage of toy cars such as Donkey Car a valuable tool in building and validating end-to-end self-driving models, as novel techniques in the field can only be developed if they are tested in real driving environments (and not only in an analytical open-loop fashion, which doesn't account for factors such as speed and delays).

The findings presented open up the field of end-to-end self-driving nets to the concept of using frameshift as a potential tool to fight against lag, although this novel idea requires further testing in different conditions, e.g. using the Autonomous Driving Lab's Lexus and real-world data, to come to a final conclusion.

6 Future Work

The development of end-to-end self-driving models has been quite rapid in the last years and the findings of this thesis hopefully add another subject of investigation in the domain. Based on the experience gained in the work, some directions can be given in respect to the next possible facets of lag and frameshift to investigate.

Firstly, the frameshifted models are tested in this work in laboratory conditions, albeit in a closed-loop manner, and the performance gains are notable, but this does not necessarily mean one might see frameshifted models beating non-frameshifted equivalents in real-life conditions (e.g. real-life cars driving on highways). Frameshift should also be tested in every-day conditions to see whether frameshift even makes a difference or is required to beat lag when there are other factors to account for, such as city traffic or speed limits.

Secondly, the camera hardware used in this work facilitated driving at 20Hz (making for a frametime of 50ms), but the effect of frameshift could also be investigated at a higher framerate. The smallest unit of framerate possible presented in the thesis was 50ms, but if one were to use a 40Hz camera, this could be 25ms. This would allow the model to be fine-tuned more precisely (frameshift-wise) and could lead to larger performance gains.

Thirdly, the negatively frameshifted models in this work were not able to drive at speeds of the training data, but they were able to drive very cleanly at slower speeds, a feat that the non-frameshifted baseline model couldn't boast. The effect of negative frameshift should further be treated as a subject of interest, as it could possibly lead to smoother driving when the driving in the training data is not smooth enough.

Lastly, the baseline model used to predict steering angles in this thesis was a single-frame model, but frameshift in multi-frame models was not tested. It is possible that the performance gains of frameshifted models are also present when the underlying model is something else than a single-frame model, but this remains to be seen and should definitely be a topic of investigation in the future.

References

[Badue et al., 2021] Badue, C., Guidolini, R., Carneiro, R. V., Azevedo, P., Cardoso, V. B., Forechi, A., Jesus, L., Berriel, R., Paixão, T. M., Mutz, F., de Paula Veronese, L., Oliveira-Santos, T., and De Souza, A. F. (2021). Self-driving cars: A survey. *Expert Systems with Applications*, 165:113816.

[Bewley et al., 2018] Bewley, A., Rigley, J., Liu, Y., Hawke, J., Shen, R., Lam, V.-D., and Kendall, A. (2018). Learning to drive from simulation without real world labels.

[Bishop, 2007] Bishop, C. M. (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition.

[Codevilla et al., 2018] Codevilla, F., López, A. M., Koltun, V., and Dosovitskiy, A. (2018). On offline evaluation of vision-based driving models.

[Donkey, 2022] Donkey (2022). Donkey car documentation and explanation. `https://docs.donkeycar.com/`.

[H. Kurniawan, 2021] H. Kurniawan, N. Muhammad, A. T. (2021). Implementing network lag for keras linear with donkey car. *Medium*. https://handykurniawan.medium.com/implementing-network-lag-for-keras-linear-with-donkeycar-43fd726c2afe.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[Hecker et al., 2019] Hecker, S., Dai, D., and Gool, L. V. (2019). Learning accurate, comfortable and human-like driving. *CoRR*, abs/1903.10995.

[Hecker et al., 2018] Hecker, S., Dai, D., and Van Gool, L. (2018). End-to-end learning of driving models with surround-view cameras and route planners. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill series in computer science. McGraw-Hill, 1 edition.

[O'Shea and Nash, 2015] O'Shea, K. and Nash, R. (2015). An introduction to convolutional neural networks. *CoRR*, abs/1511.08458.

[Paden et al., 2016] Paden, B., Čáp, M., Yong, S. Z., Yershov, D., and Frazzoli, E. (2016). A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55.

[R. Abumalikov, 2022] R. Abumalikov, A. A. (2022). Self-driving donkey car. `https://github.com/rabdumalikov/self-driving-donkey-car`.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

[SAE, 2018] SAE (2018). Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles.

[Stanford University, 2022] Stanford University (2022). University course cs231n: Deep learning for computer vision.

[Synopsis, 2022] Synopsis (2022). The 6 levels of vehicle autonomy explained.

[Tabian et al., 2019] Tabian, I., Fu, H., and Sharif Khodaei, Z. (2019). A convolutional neural network for impact detection and characterization of complex composite structures. *Sensors*, 19(22).

[Tampuu, 2020] Tampuu, A. (2020). *Neural networks for analyzing biological data*. PhD thesis, Tartu University.

[Tampuu et al., 2020] Tampuu, A., Matiisen, T., Semikin, M., Fishman, D., and Muhammad, N. (2020). A survey of end-to-end driving: Architectures and training methods. *IEEE Transactions on Neural Networks and Learning Systems*, 33(4):1364–1384.

[TensorFlow, 2022] TensorFlow (2022). Tensorflow lite.

[Uduste, 2022a] Uduste, I. (2022a). Donkey car implemented lag. `https://github.com/ilmaruduste/donkey-car-implemented-lag`.

[Uduste, 2022b] Uduste, I. (2022b). Self-driving donkey car speed test. https://www.youtube.com/watch?v=7EHxEE7igr0.

[University of Tartu, 2021] University of Tartu (2021). University of tartu and bolt expand collaboration to develop it solutions for self-driving vehicles. `https://cs.ut.ee/en/content/university-tartu-and-bolt-expand-collaboration-develop-it-solutions-self-driving-v`

[World Health Organization, 2021] World Health Organization (2021). Road traffic injuries.

[Xiao et al., 2019] Xiao, Y., Codevilla, F., Gurram, A., Urfalioglu, O., and López, A. M. (2019). Multimodal end-to-end autonomous driving. *CoRR*, abs/1906.03199.

[Yamashita, 2018] Yamashita, N. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*.

[Yurtsever et al., 2019] Yurtsever, E., Lambert, J., Carballo, A., and Takeda, K. (2019). A survey of autonomous driving: Common practices and emerging technologies. *CoRR*, abs/1906.05113.

[Zeng et al., 2019] Zeng, W., Luo, W., Suo, S., Sadat, A., Yang, B., Casas, S., and Urtasun, R. (2019). End-to-end interpretable neural motion planner. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Appendix

I. Experimental Results of Frameshifted Models on the Track

Frameshift	Artificial Delay	Signal-To-Steering Time	Lag Delta	Infractions per Lap	Interventions per Lap	Driving Score	Average Time of Fastest Laps
-100ms		23.5ms	123.5ms	0	0	10	10.3s
-50ms		23.9ms	73.9ms	0.04	0.04	9.50	9.3s
0ms		24.1ms	24.1ms	0.2	0.12	8.3	8.5s
50 ms	0ms	23.7ms	-26.3ms	0.12	0.2	7.7	7.4s
100 ms		-	-	-	-	-	-
150 ms		-	-	-	-	-	-
200 ms		-	-	-	-	-	-
-100ms		49.1ms	149.1ms	0.16	0.20	7.6	11.9s
-50ms		48.6ms	98.5ms	0.04	0.08	9.1	9.7s
0ms		49.0ms	49.0ms	0.12	0.12	8.5	8.6s
50 ms	25ms	48.9ms	-1.1ms	0	0.04	7.7	7.8s
100 ms		48.8ms	-51.2ms	0.04	0.56	4.3	8.0s
150 ms		-	-	-	-	-	-
200 ms		-	-	-	-	-	-
-100ms		77.7ms	177.7ms	0.12	0.20	7.7	13.2s
-50ms		77.8ms	127.8ms	0	0	10	11.8s
0ms		77.8ms	77.8ms	0.04	0.24	7.5	10.1s
50 ms	50ms	77.7ms	27.7ms	0.08	0.68	3.0	9.1s
100 ms		77.4ms	-22.6ms	0.04	0.56	4.3	8.5s
150 ms		-	-	-	-	-	-
200 ms		-	-	-	-	-	-
-100ms		105.4ms	205.4ms	0.2	0.04	9.1	16.2s
-50ms		105.3ms	155.3ms	0.36	0.04	8.7	13.8s
0ms		105.2ms	105.2ms	0.24	0.44	5.0	11.5s
50 ms	75ms	105.3ms	55.3ms	0.16	0.8	1.6	10.5s
100 ms		105.1ms	5.1ms	0.08	0.6	3.8	9.4s
150 ms		104.4ms	-45.6ms	0.12	0.68	2.9	8.1s
200 ms		-	-	-	-	-	-
-100ms		131.1ms	231.1ms	0.28	0.08	8.5	17.5s
-50ms		131.0ms	181.0ms	0.12	0.04	9.3	13.7s
0ms		131.0ms	131.0ms	0.16	0.16	8.0	12.5s
50 ms	100ms	131.2ms	81.2ms	0.04	0.2	7.9	11.3s
100 ms		130.9ms	30.9ms	0.6	0.04	8.1	10.6s
150 ms		131.2ms	-18.9ms	0.08	0.4	5.8	9.1s
200 ms		-	-	-	-	-	-

II. Confusion Matrix of Frameshifted Model Predictions

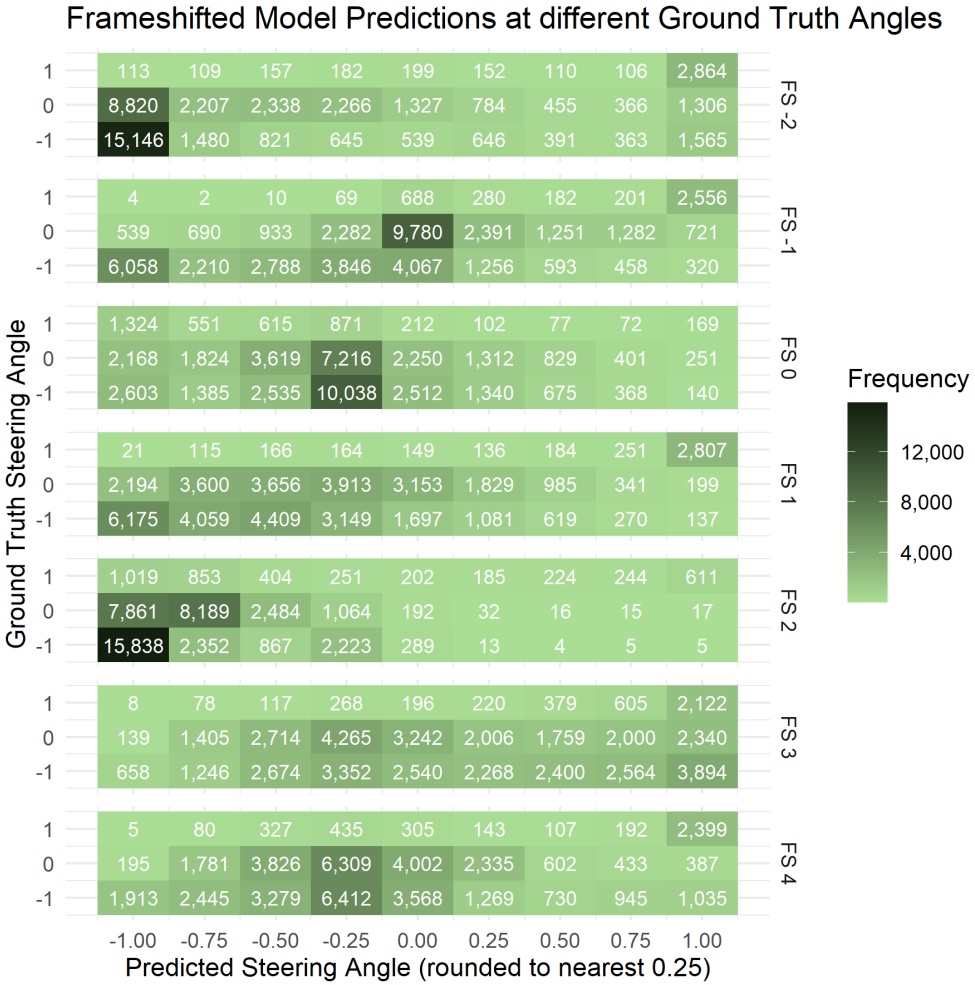

Figure 24. A confusion matrix of predicted steering angles versus the ground truth steering angle. The ground truth steering angles are either left (-1), straight (0) or right (1) and the predicted steering angles are rounded to the nearest 0.25.

III. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Ilmar Uduste**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Effect of Delays/Lag and Fighting it in Self-driving Neural Networks,

   supervised by Ardi Tampuu.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Ilmar Uduste
17/05/2022