ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

SAMPLE-EFFICIENT DEEP LEARNING METHODS FOR AUTONOMOUS SYSTEMS

M.Sc. THESIS

Yunus BİÇER

Department of Aeronautics and Astronautics Engineering

Aeronautics and Astronautics Engineering Programme

JUNE, 2019

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

SAMPLE-EFFICIENT DEEP LEARNING METHODS FOR AUTONOMOUS SYSTEMS

M.Sc. THESIS

Yunus BİÇER
(511161155)

Department of Aeronautics and Astronautics Engineering

Aeronautics and Astronautics Engineering Programme

Thesis Advisor: Asst. Prof. Dr. Nazım Kemal ÜRE

JUNE, 2019

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

OTONOM SİSTEMLER İÇİN VERİMLİ ÖRNEKLEMELİ DERİN ÖĞRENME YÖNTEMLERİ

YÜKSEK LİSANS TEZİ

Yunus BİÇER
(511161155)

Uçak ve Uzay Mühendisliği Anabilim Dalı

Uçak ve Uzay Mühendisliği Programı

Tez Danışmanı: Asst. Prof. Dr. Nazım Kemal ÜRE

HAZİRAN, 2019

Yunus BİÇER, a M.Sc. student of ITU Graduate School of Science Engineering and Technology 511161155 successfully defended the thesis entitled "SAMPLE-EFFICIENT DEEP LEARNING METHODS FOR AUTONOMOUS SYSTEMS", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Asst. Prof. Dr. Nazım Kemal ÜRE** Istanbul Technical University	.....
Jury Members :	**Asst. Prof. Dr. Emre KOYUNCU** Istanbul Technical University	.....
	Dr. Umut GENÇ University of Cambridge, Eatron Technologies	.....
		.....

Date of Submission : **3 May 2019**
Date of Defense : **13 June 2019**

To my family,

FOREWORD

First of all, I want to give special thanks to my supervisor Assist Prof. Dr. N. Kemal ÜRE for his guidance and support. Also, thanks to Ali Alizadeh for his help. Finally, I would like to say thank you to my family for their love and support.

June, 2019 Yunus BİÇER

TABLE OF CONTENTS

Page

ABBREVIATIONS

NN	**:** Neural Network
CNN	**:** Convolutional Neural Network
LSTM	**:** Long Short-Term Memory
FCN	**:** Fully Connected Neurons
ROI	**:** Region of Interest

SYMBOLS

π^*	: Expert Policy
π_i	: Trained Policy
$\hat{\pi}_{safe,0}$	: Classifier
c_i	: Class ID
$\mathbf{S}$	: State Space

LIST OF TABLES

Page

LIST OF FIGURES

Page

SAMPLE-EFFICIENT DEEP LEARNING METHODS FOR AUTONOMOUS SYSTEMS

SUMMARY

Machine learning technology has got popularity in autonomous systems in recent years because of the huge improvements at deep learning methods. Imitating a controller using deep learning methods to take actions in complex environments became one of the main research area for autonomous systems. One way to learn a controller is using imitation learning which uses expert demonstrations. An end-to-end deep learning method which takes raw demonstrations comes from an expert and outputs a control signal can be used as a controller. End-to-end imitation learning is a popular method for performing autonomous system tasks. The objective of this work is to develop a sample efficient end-to-end deep learning method for an autonomous system, where we attempt to increase the value of the information extracted from samples, through selective analysis obtained from each call to expert policy. The standard approach relies on collecting pairs of inputs and outputs from an expert policy and fitting a deep neural network to this data to learn the task. Although this approach had some successful demonstrations in the past, learning a good policy might require a lot of samples from the expert policy, which might be resource-consuming. In this work, we develop a novel framework based on the Safe Dataset Aggregation (SafeDAgger) approach, where the current learned policy is automatically segmented into different classes, and the algorithm identifies classes with the weak performance at each step. Once the weak classes are identified, the sampling algorithm focuses on calling the expert policy only on these classes, which improves the convergence rate. Firstly, DAgger algorithm implemented to the problem of estimation of the aircraft states during the landing in order to show performance of data aggregation methods in autonomous system tasks. After that, the proposed framework verified on autonomous driving tasks and the presented simulation results showed that the proposed approach can yield significantly better performance compared to the standard Safe DAgger algorithm while using the same amount of samples from the expert.

OTONOM SİSTEMLER İÇİN VERİMLİ ÖRNEKLEMELİ DERİN ÖĞRENME YÖNTEMLERİ

ÖZET

Son yıllarda, makine öğrenmesi teknolojisinin otonom sistemlerde kullanılması, bilgisayar donanımların güçlenmesi ve derin öğrenme yöntemlerindeki gelişmeler nedeniyle önem kazanmıştır. Derin öğrenmedeki gelişme ve araştırmalara paralel olarak, otonom sistem görevlerinde yöntem olarak derin öğrenme yöntemlerinin kullanılması ve araştırılması akademide artmıştır.

Otonom sistemler için derin öğrenmedeki ana araştırma alanlarından biri, standart kontrolcülerin başarısız olacağı karmaşık ortamlarda öngörülerde bulunabilecek ve kontrolörün yerini tutabilecek bir sinir ağı eğitmektir. Bir uzmanı veya ana kontrolcüyü taklit ederek bir sinir ağı eğitmek, günümüzde bir kontrolcünün görevlerini öğrenmenin ana yollarından biridir

Derin öğrenmenin otonom sistemlerde temel kullanım alanlarından biri, kendi kendini süren araba teknolojilerinde görülebilir. Özellikle, otonom sürüş algoritması oluşturmak için görüntü temelli yöntemlerin kullanılması, geniş bir araştırmacı kitlesinin ilgisini çekmiştir ve çeşitli öğrenme ve kontrol mimarilerinin gelişmesine yol açmıştır. Bu yöntemler kabaca klasik yöntemler ve uçtan uca derin öğrenme diye ikiye ayrılabilir. Klasik yöntemler, otonom sürüş sorununu üç aşamada ele alır; algılama, planlama ve kontrol. Algılama aşamasında, otoyol üzerindeki şerit işaretlerini tespit etmek için renk ayırma, kenar algılama vb. gibi öznitelik çıkarma ve görüntü işleme teknikleri uygulanır. Planlama aşamasında, algılama aşamasında tespit edilen şerit işaretlerine göre otomobilin gitmesi gereken yolun planlaması yapılır. Kontrol bölümünde ise, otomobil için direksiyon, hız vb. kontrol eylemlerini bir kontrol algoritması kullanarak belirlemek için planlama ve algılama aşamalarından gelen yol ve rota bilgisi kullanılır. Klasik yöntemlerin performansı büyük ölçüde algılama aşamasının performansına bağlıdır ve bu aşamada kullanılan sub-optimal yöntemler sebebiyle belirlenen otonom sürüş algoritması optimal olmayabilir. Klasik yöntemlerin sıralı yapısı sebebiyle algılama aşamasındaki bir hata büyüyerek son aşamaya kadar gider ve önlenemez hatalara yol açar.

Öte yandan, uçtan uca derin öğrenme yöntemleri, uzman bir sürücünün sürüş verisinden elde edilen örneklerden bir fonksiyon öğrenir. Öğrenilen fonksiyon, kontrol girişlerini doğrudan görüş verilerinden üreterek klasik kontrol sekansının üç katmanını tek bir adımda birleştirir. Şimdiye kadar, uçtan uca derin öğrenme yöntemleri ile otonom sürüş algoritması üretmede en popüler yaklaşım sinir ağları(NN) kullanmaktır. CNN, LSTM yapısındaki ağlarda diğer kullanılan yöntemler arasında sayılabilir.

Uçtan uca derin öğrenme yöntemleri, gösterilen veri ile sınırlıdır ve uzmandan toplanan veriler ile eğitilen yada öğrenilen fonksiyonun tahminlerinde hatalar düşük olsa da uzun vade de öğrenilen otonom sürüş algoritması zayıf performans gösterebilir. Bu performans kaybı, kısmen, öğrenilmiş otonom sürüş algoritmasının, eğitim

aşamasında kullanlan verilerin dağılımına benzer ya da ait olmayan durumları gözlemlemesinden kaynaklanmaktadır.

DAgger algoritması, bu konuyu hem uzmandan hem de eğitilen sinir ağından tekrar eğitimde kullanmak üzere yeni veri toplayarak çözer. DAgger'ın ana fikri, öğrenilen otonom görev algoritmasını geliştirmek için uzmandan aktif olarak daha fazla örnek almaktır. Bu durum için ilk olarak DAgger algoritmasının otonom sistemlerde uygulanabilirliği incelenmiştir. F-16 savaş uçağının otonom iniş sırasında baş istikameti açısını iniş pistine bakan kameradan gelen görüntüye bakarak tahmin etmek üzere DAgger algoritması yöntemleri kullanılarak bir sinir ağı eğitildi. Aynı zamanda karşılaştırma yapabilmek için standart uçtan uca derin öğrenme yöntemleri kullanarak başka bir referans sinir ağı eğitildi. Eğitilen sinir ağları aynı durumlar için test edildiğinde standart yöntemler ile eğitilen sinir ağı zayıf performans göstermiştir ve simülasyon sonucunda kaza kaçınılmaz olmuştur. Bunun yanında DAgger algoritması yöntemleri kullanılarak eğitilen sinir ağı test aşamasında daha iyi performans göstererek sorunsuz bir şekilde piste inişi gerçekleştirmiştir. DAgger daha iyi otonom görev performansı elde etmesine rağmen, uzmandan çok fazla örnek alması ile sonuçlanabilir ve bu da zaman ve kaynak kaybına yol açabilir.

Derin öğrenme yöntemlerini yüksek boyutlarda eğri uydurma gibi düşünürsek eğer, kullanılan datanın gözlemlenen uzay içindeki yeri önem arzeder. Derin öğrenme yöntemlerinin tahmin performansını arttırmak için, tahminlerin kötü olduğu alt-uzaydan daha fazla data toplanıp sinir ağı tekrar bu data kullanılarak eğitildiğinde performansta artış meydana gelecektir. DAgger'ın bir diğer versiyonu olan SafeDAgger algoritması hem uzmana yapılan çağrıyı azaltmak hem de yapılan çağırılardan elde edilen verinin kalitesini arttırmak için, öğrenilen otonom sürüş algoritmasının tahminlerinin güvensiz olduğu durumları tahmin ederek uzmana yalnızca bu gibi durumlarda çağrı yapar.

SafeDAgger algoritması her ne kadar iyi performans gösterse de veri toplama aşamasında sadece verilerin güvenli yada güvensiz olma durumlarına bakmaktadır. Güvensiz olarak seçilen durumların sınıflandırmasını yapmamaktadır. Bu durum sıralı karar verme süreci içeren otonom görevlerde, verimli örnek toplama bağlamında etkili bir yöntem değildir. Sıralı karar verme süreci gerektiren otonom görevlerde bir önceki karar bir sonraki gözlem uzayını etkiler. Verilen karar içinde bulunduğu anda küçük bir miktar hatalı olur ise sonrasındaki gözlem uzayına bakarak verilen kararların doğru olandan sapma ihtimali artar. Tüm bunların önüne geçmek için sorunlu gözlem uzayını başlangıçtan itibaren çözmek gerekir.

Bu çalışmada, Selective SafeDAgger adlı ve SafeDAgger algoritmasına kıyasla, örneklem verimli olan yeni bir veri toplama methodu önerilmektedir. Önerilen algoritma, öğrenilen otonom sürüş algoritması tarafından yürütülen sorunlu durumların güvenli ve güvensiz bölümlerini sınıflandırır. Sınıflandırma işleminden sonra belirlenen en sorunlu sınıftan örnek almaya odaklanmaktadır. Sorunlu sınıftan alınan örnekler ile sinir ağı yeniden eğitilerek tahmin performansının arttırılması amaçlanmıştır. Böylece problem kaynağından çözülerek sonraki gözlem uzayına etkisi azaltmak mümkündür. Özetlemek gerekirse, bu çalışma ile literatüre asıl katkımız, uzman sürüş verisinden en efektif örnekleri alan ve aynı zamanda uzmana yapılan çağrıları sınırlandırıp SafeDAgger yönteminden daha iyi performans elde etmeyi sağlayan bir taklit öğrenme yöntemi geliştirmektir.

Selective SafeDAgger algoritmasının performasını SafeDAgger algoritması ile karşılaştırmak amacıyla bir otonom araç simülasyonu oluşturulmuştur. Simülasyonun amacı araç önüne konulan bir kameradan gelen görüntülere bakarak aracın gitmesi gereken hızı ve direksiyon açısını tahmin etmektir. Bu amaçla iki algoritma da aynı ortamda derin öğrenme metodu kullanarak eğitilmiştir. Eğitilen sinir ağları test senaryoları ile karşılaştırılmıştır. Karşılaştırma sonucunda Selective SafeDAgger algoritmasının verimli örnekleme ve uzmana yapılan çağrı sayısı bakımından SafeDAgger algoritmasına üstünlüğü gösterilmiştir.

1. INTRODUCTION

In last decades, the machine learning technology gained importance in autonomous systems due to the improvement of the computational power in hardwares and deep learning which is one the methods for machine learning is got more advancement due to the parallel computation ability on GPUs. In parallel with developments in deep learning, research for deep learning in autonomous systems tasks also increased in academia.

One of the main research areas in deep learning for autonomous systems is learning a controller to make predictions in complex environments which standard controllers will fail in those environments. Imitation learning is one way to learn a controller with using expert demonstrations and use of deep learning in imitation learning became a state-of-the-art method, nowadays.

One of the main usages of deep learning can be seen self-driving car technologies. In particular, the use of vision-based methods to generate driving policies has been of interest to a wide body of researchers, resulting in a variety of different learning and control architectures. These methods can be roughly classified into classical and end-to-end methods. Classical methods approach the problem of autonomous driving in three stages; perception, path planning, and control [1]. In the perception stage, feature extraction and image processing techniques such as color enhancement, edge detection, etc. are applied to image data to detect lane markings. In path planning, reference and the current path of the car are determined based on the detected features in perception. In the control part, control actions for the car such as steering, speed, etc. are calculated from reference and the current path with an appropriate control algorithm. The performance of the classical methods heavily depends on the performance of the perception stage and this performance can be sub-optimal because of the manually defined features and rules in this stage [2]. Sequential structure of the classical methods might also lead to the non-robustness against errors, as an error in feature extraction can result in an inaccurate final decision.

On the other hand, end-to-end learning methods learn a function from the samples obtained from an expert driving policy. The learned function can generate the control inputs directly from the vision data, combining the three layers of the classical control sequence into a single step. By far, the most popular approach for representing the mapping from images to controls in end-to-end driving is using neural networks (NN). ALVINN by Pomerleau [3] is one of the initial works in this area, which uses a feedforward neural network that maps frames of the front-facing camera to steering input. Researchers from Nvidia utilized convolutional neural networks (CNN) [4] in order to automatize the feature extraction process and predict steering input. An FCN-LSTM architecture [5] is proposed to increase learning performance with scene segmentation. In [6], a visual attention model used to highlight some important regions of frames for better prediction. Although the steering input prediction in an end-to-end manner is a well-studied problem in the literature, the steering input alone is not sufficient for fully autonomous driving. In [7] a CNN-LSTM network is proposed to predict the speed and steering inputs synchronously.

Pure end-to-end learning policies are limited to the demonstrated performance, and although the training and validation loss on the data collected from the expert might be low, errors accumulated from the execution of the learned driving policy might lead to poor performance in the long run. This performance loss is partly due to the fact that learned driving policy is likely to observe states that do not belong to the distribution of the original expert demonstration data. DAgger [8] algorithm addresses this issue by iteratively collecting training data from both expert and trained policies. The main idea behind DAgger is to actively obtain more samples from the expert in order to improve the learned policy. Even though DAgger achieves better driving performance, it might end up obtaining a lot of samples from the expert, which can be time and resource consuming in many real-life scenarios. SafeDAgger [9] algorithm, an extension of DAgger, attempts to minimize the number of calls to the expert by predicting the unsafe trajectories of the learned driving policy and only calls the expert on such cases. Another extension of DAgger, EnsembleDAgger [10], predicts the variance of the decisions by using multiple models, also it takes the variance as additional safety criteria like SafeDAgger.

In this work, we propose a novel framework which is sample-efficient compared to the SafeDAgger algorithm (state-of-the-art data aggregation method), named Selective SafeDAgger. The proposed algorithm classifies the problematic observations executed by the learned policy to safe and multiple classes of unsafe segments. After the prediction, the model focuses on obtaining the expert policy samples primarily from the identified unsafe segment classes.

Our main contribution is an imitation learning algorithm that obtains the most promising samples from the expert policy, which enables outperforming the SafeDAgger method while limited to the same number of calls to the expert.

This paper is organized as follows. Definitions of the used terms and algorithms and details of the new proposed algorithm, Selective SafeDAgger, are provided in section II. In section III, experimental verification of algorithm and discussion about results are given. The conclusion is given in section IV.

2. METHODOLOGY

In this section, definitions of some fundamentals have given for ease of understanding for algorithms and detail of proposed algorithm is provided.

2.1 Deep Neural Networks

Neural networks can be defined as a set of algorithms that helps to devise patterns in dataset. Deep learning is composition of several neural networks or layers.

Neurons are the main elements of the layers. All computations occur in the neurons. It has two part, the first part scales the input data by multiplying with a weight and in the second part these weighted inputs are added to each other and the total collection of weighted inputs are passed to the activation function to decide whether signal should continue through the network or not.

Deep neural networks are constructed with using layers with a defined amount of neurons. At the first stage, data is given to input layer. After that at hidden layers, multiplications and summations are done. In the output layer, mapping from input to output is done as shown in Fig. 2.1.

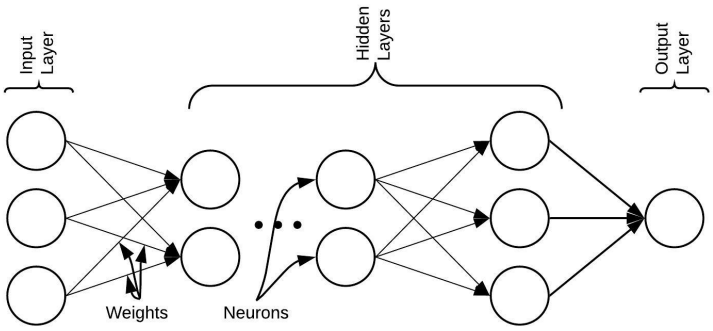


Figure 2.1 : Deep neural networks.

Deep learning, generally, used for mapping inputs to outputs by finding correlations. It can learn to approximate an unknown function between input and output. During the learning, neural network tries to find the right weights to predict output by using feed-forward, back-propagation and an optimization algorithm to minimize differences

between predicted output and ground truth. In this manner, we can say that deep learning tries to fit a curve between input and output in high dimension with using nonlinear tools.

2.2 Imitation Learning

Training a policy to make decisions using demonstrations can be definition of the imitation learning. It is a kind of supervised learning but with a difference which predictions are made sequentially.

There are 5 key elements of the imitation learning. First one is a demonstrator to imitate labeled data of this demonstrator. Demonstrator can be a driver, pilot or even a controller. Second one is an environment which labeled data comes from observations of this environment. Third one is a policy class that predicts the actions of the demonstrator. It is popular to use deep neural networks to perform this task. Fourth one a loss function to quantify differences between the action of our policy demonstrator. And the last one is a learning algorithm to minimize loss function problem. It is usually variant of gradient-descent optimization algorithm.

2.3 DAgger

It is well-known that the prediction performance for the imitation learning policies is limited with the collected dataset. In sequential decision-making problems, error accumulation of trained policy can lead to low performance in the long term. DAgger [8] handles this problem by aggregation of training data from both controller and trained policies. The basic idea behind DAgger is to collecting data from policy when it is in action. In parallel controller labels a recovery commands for new observation. Initial dataset and new observations are used to train the new policy as shown in algorithm 1.

$$\beta_i = \frac{N+1-i}{N} \tag{2.1}$$

Table 2.1 : Algorithm 1.

DAgger Algorithm

Initialize $D_0 \leftarrow \emptyset$
Train $\hat{\pi}_1$ over set D_0
for *i = 1 to N* **do**
    Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$
    Sample T-step trajectories using π_i
    Get dataset $D_i = \{(s, \pi^*(s))\}$ of visited states by π_i and ground truths given by π^*
    Aggregate datasets: $D_i \leftarrow D_i \cup D_{i-1}$
    Train $\hat{\pi}_{i+1}$ over set D_i
end
return best $\hat{\pi}_i$ on validation

Algorithm starts with collecting initial dataset D_0 and train initial policy $\hat{\pi}_1$ over D_0. A dummy policy(π_i) is defined as a weighted sum of initial policy and controller with a parameter of β_i which is defined in equation 2.1. At the beginning of the DAgger iterations $\hat{\pi}_1$ have poor performance and by weighting with $1 - \beta_i$ we can eliminate any chance of a crash. With using π_i in parallel with controller (π^*) for T-step time, a new dataset D_i is collected. This dataset has labels coming from the controller. A new policy ($\hat{\pi}_{i+1}$) is trained with using the union of the initial and new dataset. This iterative loop continues for N times.

2.4 SafeDAgger

The prediction performance of a neural network dramatically increases with use of DAgger algorithm but it comes with a cost. A large number of calls to expert policy needed for training of policy and calls for the expert policy can be costly, especially in case of human operator as a expert policy . Another disadvantages of using human operator is mislabeling problem. Human operator cannot feel the actual feedback of the their actions, since weighted sum of expert policy and trained policy applied to the system. As a result, human operator overreacts for some states and it cause the wrong labeling for that states.

SafeDAgger [9] which is an extension of DAgger, tries to minimize calls for the expert policy by looking a safety rule. By training a safety policy, algorithm tries to predict probability of deviation from the expert policy. If predicted probability becomes bigger

than a threshold, expert policy takes full control of the system. By using a safety strategy, number of calls to expert policy significantly decreases and since expert policy takes full control, mislabeling problem automatically solved.

Table 2.2 : Algorithm 2.

SafeDAgger Algorithm

Initialize $D_0 \leftarrow \emptyset$
Initialize $D_{safe} \leftarrow \emptyset$
Train $\hat{\pi}_1$ over set D_0
Train $\hat{\pi}_{safe,0}$ over set D_{safe}
for *i = 1 to N* **do**
    Collect D_i using safety strategy for $\hat{\pi}_1$ and $\hat{\pi}_{safe,0}$
    Aggregate datasets: $D_i \leftarrow D_i \cup D_{i-1}$
    Train $\hat{\pi}_{i+1}$ over set D_i
    Train $\hat{\pi}_{safe,i+1}$ over set D_{safe}
end
return best $\hat{\pi}_i$ on validation

The Algorithm which is shown in table 2.2, starts with collecting initial dataset D_0 and train initial policy $\hat{\pi}_1$ over D_0 as in DAgger. In addition to $\hat{\pi}_1$, a network $\hat{\pi}_{safe,0}$ is trained over D_0 to predict probability of deviation from the expert policy. With using safety rule for the control of the system for T-step time, a new dataset D_i is collected. This dataset has labels coming from the expert policy. New policies ($\hat{\pi}_{i+1}$) and $\hat{\pi}_{safe,i+1}$ are trained with using the union of the initial and new dataset. This iterative loop continues for N times.

2.5 Selective SafeDAgger Algorithm

Even though, DAgger and SafeDAgger solves the problem of imitation learning for some level by aggregating new dataset, this method still suffers from large number of calls to expert policy and aggregation of efficient training data. To enhance existing methods we have proposed a new dataset aggregation method which is called Selective SafeDAgger.

Main idea behind Selective SafeDAgger is to classify unsafe observations and collecting new dataset from most problematic class of observations. Unlike SafeDAgger which collects all unsafe dataset, Selective SafeDAgger tries to solve problem of unsafe observations from the roots. Some of the unsafe observations are the

result of the previous unsafe observations and if we solve the first unsafe observations we can get rid of following unsafe observations.

Table 2.3 : Algorithm 3.

Selective SafeDAgger

1 Collect D_0 using π^*
2 $\pi_0 = \arg\min_\pi l_{supervised}(\pi, \pi^*, D_0)$
3 **for** i = *1:N* **do**
4 $c_i \leftarrow$ Define unsafe classes over D_0
5 $D' \leftarrow [\,]$
6 **while** $k \leq T$ **do**
7 $\phi_k \leftarrow \phi(s)$
8 $c_{\phi_k} \leftarrow$ classifier output of $\pi_i(\phi_k)$
9 **if** $c_{\phi_k} \in c_i$ **then**
10 use $\pi^*(\phi_k)$
11 $D^{'} \leftarrow [\phi_k]$
12 $k = k + 1$
13 **else**
14 use $\pi_i(\phi_k)$
15 **end**
16 **end**
17 $D_i = D_{i-1} \cup D^{'}$
18 $\pi_{i+1} = \arg\min_\pi l_{supervised}(\pi, \pi^*, D_i)$
19 **end**
20 **return** best π_i over validation set

*Blue fonts distinguishes the difference between Selective SafeDAgger and SafeDAgger.

Algorithm 3 describes the proposed method in detail, which takes the expert policy π^* as an input and gives π_i as an output. The primary dataset D_0 is collected by using π^* which is then utilized in training a primary policy π_0 by a supervised learning method. Having the π_0 at hand, c_i, the unsafe classes of D_0 for the trained policy π_i are determined. An observation ϕ_k taken from environment $\phi(s)$ is evaluated by π_i to find its class c_{ϕ_k}. If c_{ϕ_k} is an element of c_i, π^* takes over the control of the car and ϕ_k is appended to $D^{'}$. Otherwise, π_i continues to command the car until it encounters an unsafe class. As depicted in lines 6-16, the algorithm continues to append data to $D^{'}$ for T number of iterations. The appended dataset $D^{'}$ is aggregated into D_{i-1} to create D_i and π_{i+1} is trained on D_i. This loop is repeated for N times as shown in lines 3-18. In the end, the algorithm returns the best π_i over the validation set.

3. EXPERIMENTS

To analyze and verify the proposed algorithms, two experiment are established. First one is heading angle prediction for an F-16 aircraft during landing by using Dagger algorithm. Second one is reference steering and speed prediction for a self-driving cars by using front facing camera images.

3.1 Visual Landing of an F-16 Aircraft

We developed a Convolutional Neural Network (CNN) that can learn from demonstrations and estimate the aircraft states. The main aim is to perform the autonomous landing for an aircraft model in various possible system failures. In this section, we first introduce the aircraft model and a sketch of the guidance and control loops that are used in both training and testing the agent.

3.1.1 Aircraft model & Airsim

In this work, a nonlinear high-fidelity model of the F-16 aircraft [11] is used. The model parameters and aerodynamic data tables are taken from [12]. Furthermore, the standard atmosphere model that is used and 6-DOF equations can be found in the literature [13], [14]. The observed state vector is as follows

$$x^T = [V_T, \gamma, \chi, \mu, \alpha, \beta, \phi, \theta, \psi, P, Q, R, p_n, p_e, h] \tag{3.1}$$

here, we assume the full-sate measurement to be used in the feedback law.

3.1.2 Automatic path planning and landing

Landing of any aircraft can be divided into three phases; approach, glide, and flare. During the approach, the aircraft maneuvers in order to align the nose and the velocity vector with the runway. From initial position x_0, y_0 the guidance logic calculates the glide initiation point x_g, y_g and forms a straight line which is to be followed. Figure 3.1 shows the planar geometry of approach. In the glide phase, the aircraft follows a prefixed flight path angle (generally -3^o) towards the runway. Flare is performed

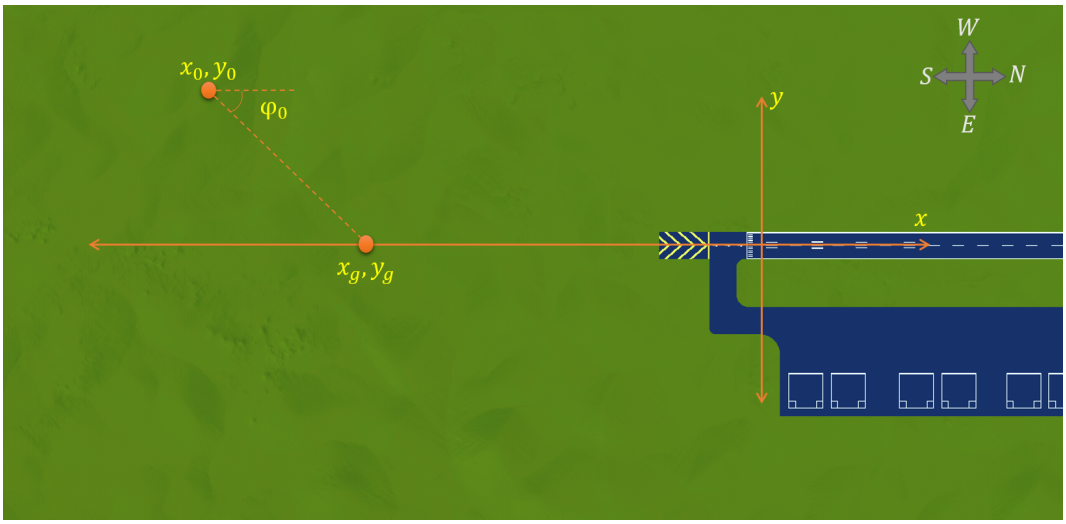


Figure 3.1 : Approach geometry.

40 ft above the runway to land without inflicting structural damage to the aircraft by pitching the nose up slightly to reduce the vertical speed.

Automatic path planning logic for landing has been taken from [15]. The algorithm calculates glide and flare initiation points given the initial position. A straight line between initial position x_0, y_0 and glide initiation point x_g, y_g is to tracked in approach. Likewise, the path to be followed in a glide is a straight line between x_g, y_g and the flare initiation point. To capture the decrease in vertical speed, flare path is an appropriate exponential between the flare initiation point and a desired point on the runway.

3.1.3 Initial dataset(D_0) collection

An autolanding aircraft simulation has been created with a low-level controller taken from [16] and a guidance logic taken from [15]. Guidance logic outputs reference altitude and heading angle to the low-level controller, which uses reference inputs to calculate control surface deflections. The low-level controller is designed using dynamic inversion and includes a separate lateral and longitudinal controller.

Visualization of the aircraft simulation has been achieved with Airsim which an Unreal game engine based plugin that works in the unreal editor and provides photo-realistic simulation environment. Airsim also has built-in API for communicating with Python scripts. A custom Python script receives state information of aircraft from Simulink over UDP network and sends to the Unreal environment with using Airsim API's. At the same time script captures screenshots of simulation at desired time steps and

labels the captured screenshot with the state information. Hence a pool of training images with desired size and amount from any phase of flight can be collected with this structure. Block diagram of the dataset collection framework is shown in Figure 3.2.

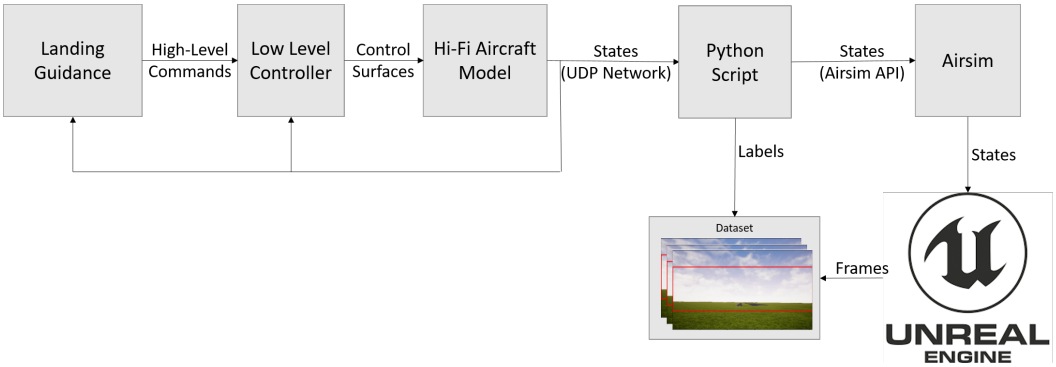


Figure 3.2 : Block diagram of data collection framework.

3.1.4 Data preprocessing

The recorded frames of the flight simulator were labeled with corresponding measured heading angle from the dataset. The total simulation time for 10 flights was 277.24 seconds and 27724 images were collected with the step size of 0.01 seconds. Dataset was split into three sub-dataset with the ratio of 0.7, 0.2 and 0.1 as training, validation, and test sets respectively. The resolution of images was down-sampled to $512 \times 288 \times 3$ (RGB) in order to reduce computational cost. Horizontal flip of images just changes the sign of labels. This doubles the dataset size, so we end up with 44350 labeled data. To prevent neural network model looking on irrelevant features in the environment a Region of Interest (ROI) was defined in images with the size of 512×160 as shown in Fig. 3.3. Each image was normalized to the range [0,1] by dividing to 255. In addition, in order to eliminate the time-series correlation of the dataset, we shuffled the labeled data. This prevents the network from forgetting the past experiences.

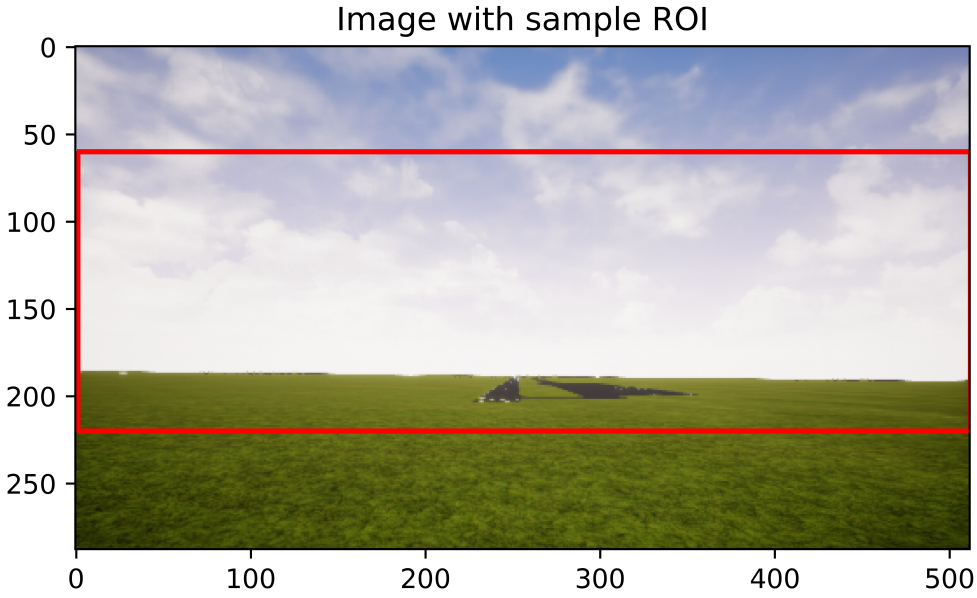


Figure 3.3 : Extracting region of interest from frames.

3.1.5 CNN architecture

Designed CNN architecture is mainly based on NVIDIA's [4] network which was designed to predict steering angles for cars from frame inputs. The network takes frames which have the size of $512 \times 160 \times 3$ as input and it comprises 5 convolutional and 5 densely connected layers as shown in Fig. 3.4. The first 3 layers of convolutional layers have kernel size of 5 with the stride of 2 and following convolutional layers has kernel size of 3 with the stride of 1. The number of filters in the convolutional layers is 24,36,48,64 and 64, respectively. Additionally, after first 3 convolutional layers average poling with a pool size of 3 is added. The network is flattened after last convolutional layer to prepare inputs of the dense layers. Densely connected layers have 200,100,50,10 and 1 neurons, respectively. For all layers except in the last layer, Rectified Linear Unit (ReLU) was used as the activation layer. In total network has 1,281,571 parameters to train.

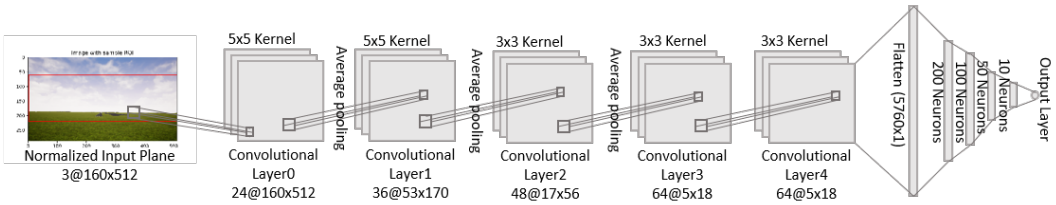


Figure 3.4 : CNN architecture.

3.1.6 Training initial policy $\hat{\pi}_1$

Nesterov Adam Optimizer (Nadam) was used as an optimizer for the training of the network with the initial learning rate of 10^{-5} and moment of 0.99. The learning rate was scaled with 0.5 when validation loss stuck in the plateau for 5 epoch. Labels of the images were heading angle in radian and values were relatively small for regression type networks. For that reason, a low value was chosen as initial learning rate and training was continued for 200 epoch with the batch size of 32 in order to decrease training and validation losses which are defined in equation 3.2 as mean square error for the single batch. At the end of the training best result obtained at 188^{th} epoch which loss values are given in table 3.1. Some of the prediction can be seen in Fig. 3.5

$$MSE = \frac{1}{n}\sum_{n=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.2}$$

Table 3.1 : Loss values for the three sub-dataset.

Training Loss	Validation Loss	Test Loss
9.11e-07	9.73e-07	1.69e-06

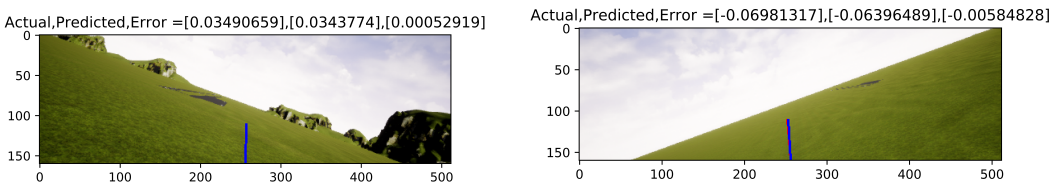


Figure 3.5 : Test set predictions (rad).

3.1.7 Visualization

Visualizing intermediate activations is a useful technique for analysis of CNN filters and it can give the first idea about the learned features of initial policy $\hat{\pi}_1$. Since higher layers activations contain less information about input features, only activations of the first and second convolution layer filters plotted and they can be seen in Fig. 3.6.

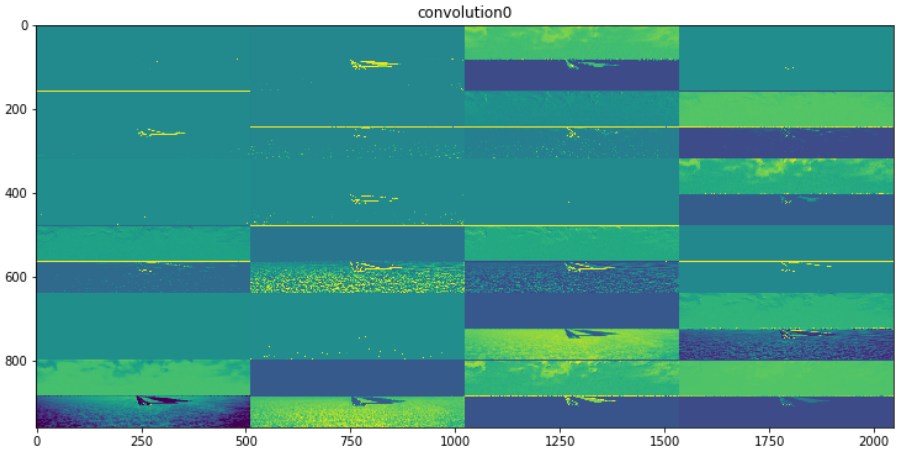
It can be seen that in the figure, some of the runways features extracted in filters. It may be said that Convolution0 filters behave like runway detector and convolution1 filters find the orientation of the runway.

3.1.8 Evaluation

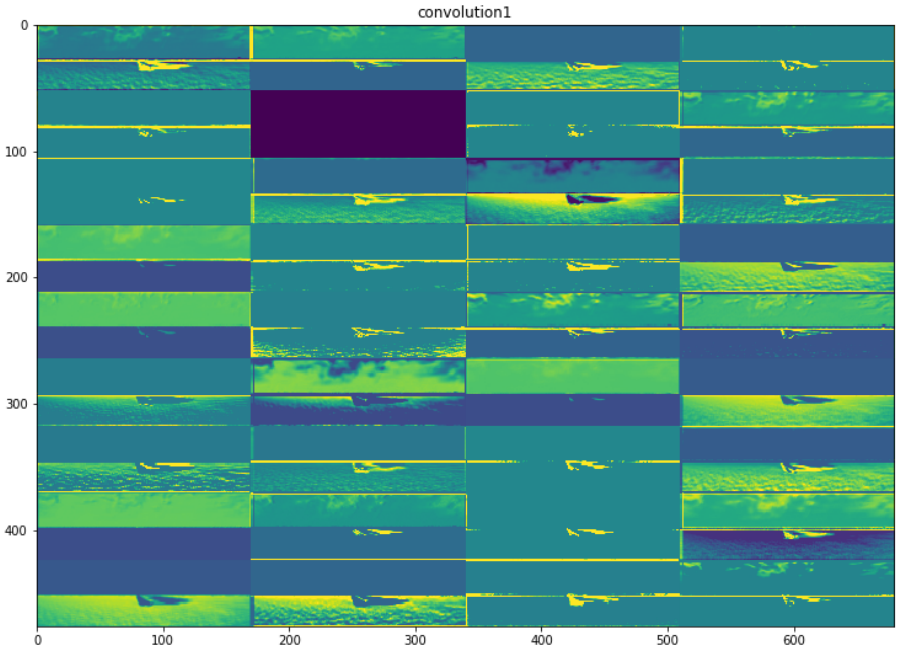
Two different scenarios were planned for the evaluation of the network. In the first scenario, the network predicts the heading angles throughout the landing flight but predictions were not given to controller as feedback on the other hand in the second scenario controller uses prediction as a feedback signal.

3.1.8.1 Neural network prediction

For the first scenario, frames of a prerecorded landing flight were used in time order and results are given in Fig. 3.7. It can be seen that for the approach and glide phases of landing, predictions are noisy according to flare phases. Training data set was collected from 10 different landing cases and for all landing cases, flight path converges to the same path for the flare phase. This means that the training contains more flare phases of data relative to the approach and glide phases. Therefore, the predictions in this phase are more accurate than the other phases.



Convolution0



Convolution1

Figure 3.6 : Visualization of first two convolutional layer activations.

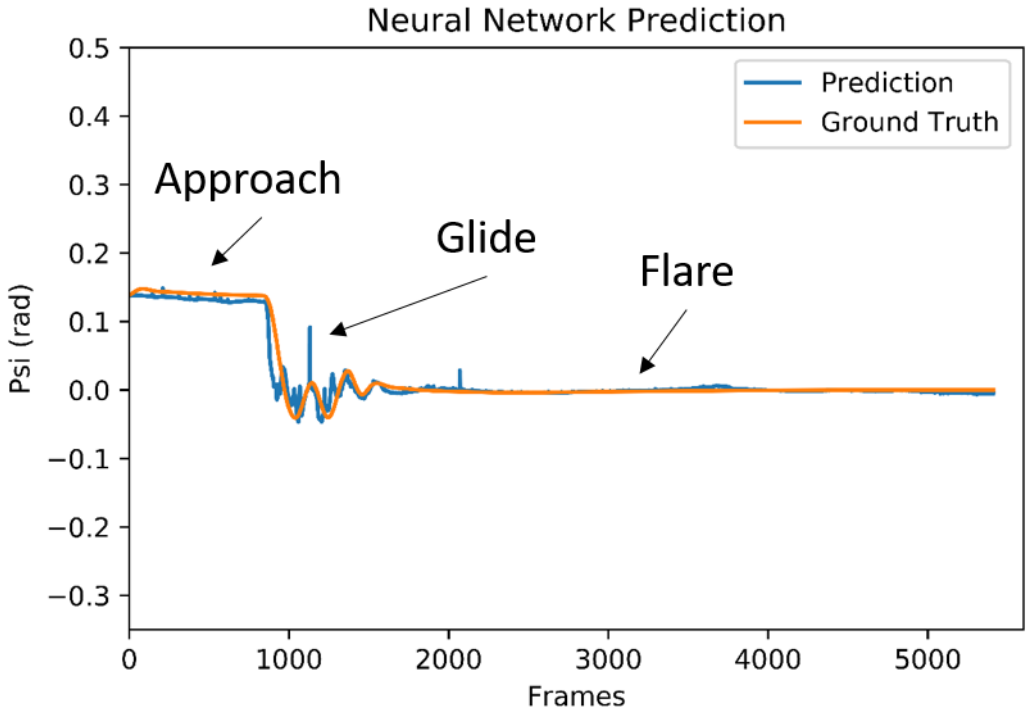


Figure 3.7 : Neural network prediction performance for the heading angle.

3.1.8.2 Neural network control

In the second scenario, the signal connection from the heading measurement sensor to the model has been replaced by the prediction of the network. The prediction scheme is fed a downscaled screenshot from the Unreal environment which is used as an input to a pre-trained DNN. Network predicts the heading angle and outputs the prediction to the model, closing the loop as shown in Fig. 3.8

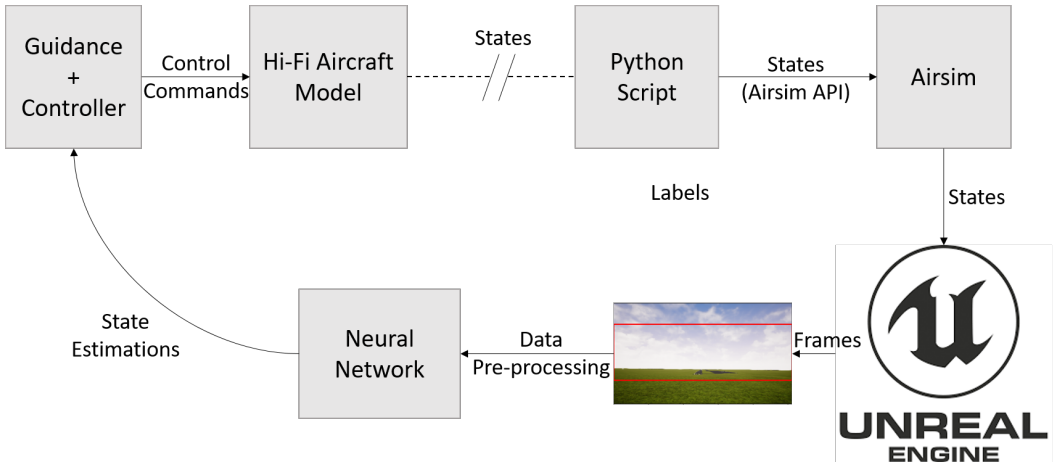


Figure 3.8 : Neural network feedback control framework.

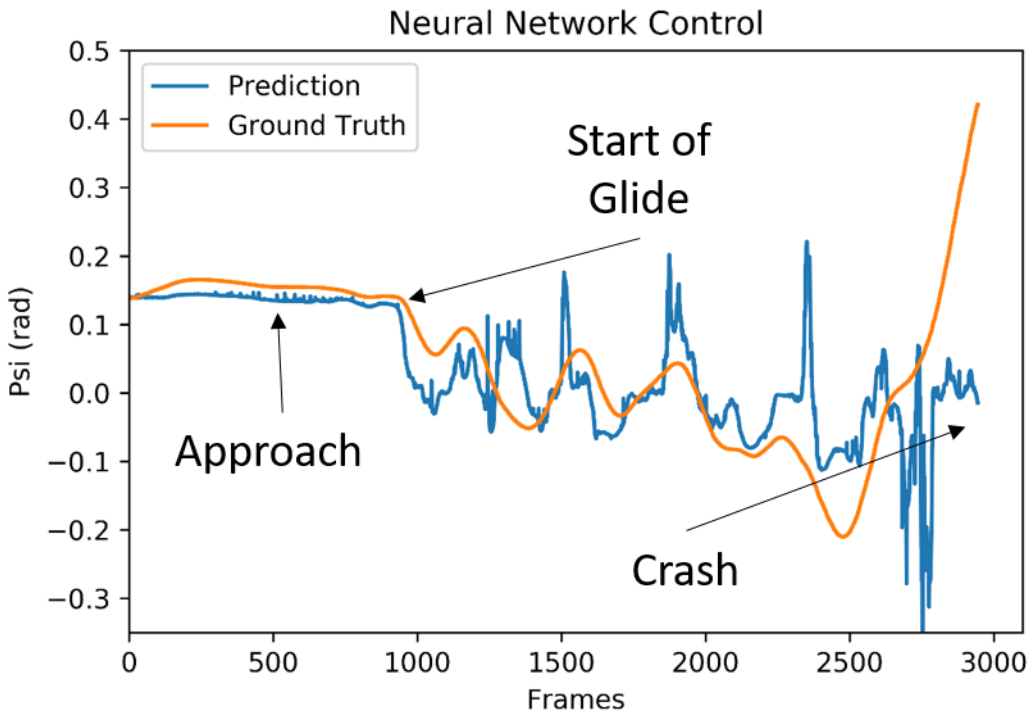


Figure 3.9 : Neural network control performance for the heading angle.

In can be seen in Fig. 3.9 that at the start of the glide, trained policy shows poor performance and at the end crash occurs. We know that to achieve given desired heading angle, aircraft performs both roll and yaw maneuvers. In our aircraft model, roll maneuver more effective than the yaw maneuver to correct any error in heading angle. When the trained model predicts the heading angle with some error, aircraft directly starts to rolling motion in order to compensate the error which leads to a significant change in the deviation of the future observation. This is a kind of sequential decision-making problem that current prediction effects the future observations. If the future observation deviates from the trained dataset, the performance of the policy reduces. We know that initial policy is a suboptimal policy because it is trained on a flight dataset that created by using a controller. The controller always stays in the correct flight path and never sees the significant changes in deviation of observations. So, when the policy deviates from nominal conditions it cannot recover since dataset does not have any recovery conditions. To be able to solve deviations on observations we used a dataset aggregation algorithm. Additionally, during the data aggregation to imitate rolling like maneuver, 2 more cameras added to the simulator with angles of ± 20 degree bank angle as shown in Fig. 3.10. Since all three cameras directed to

the same heading angle corresponding labels for new cameras will be the same as the middle camera.

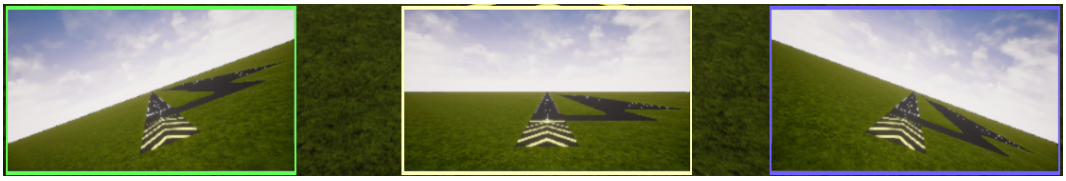


Figure 3.10 : Triple camera.

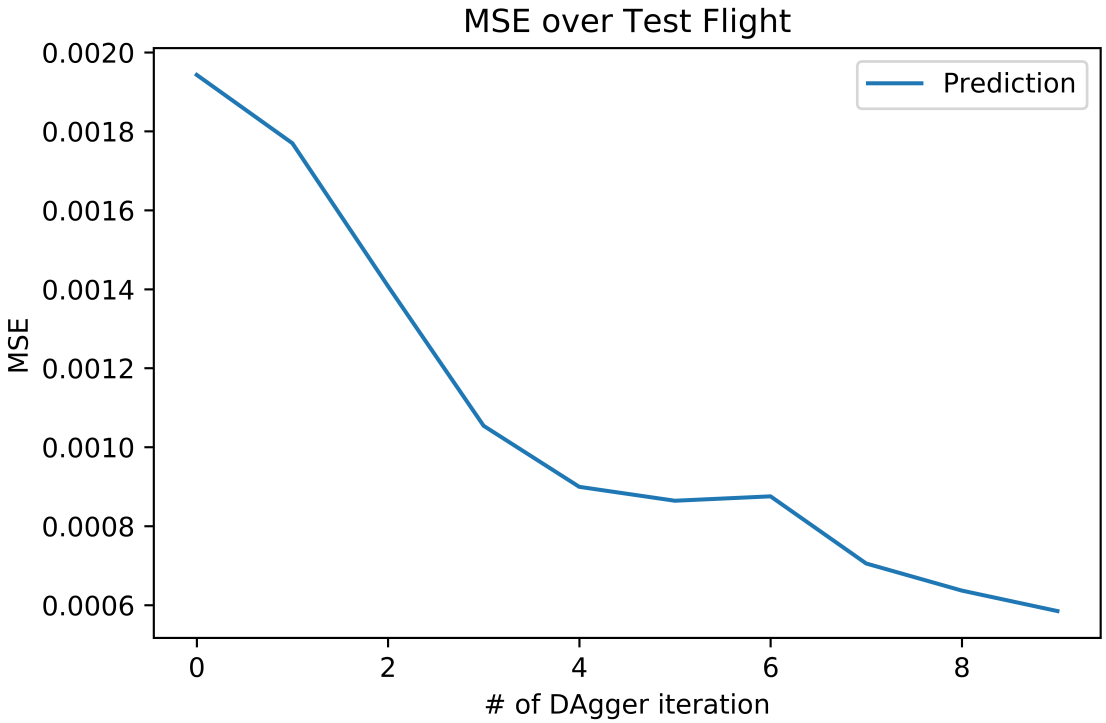


Figure 3.11 : Mean squared error over test flight for each DAgger iteration.

3.1.8.3 Results and analysis

Three different test track defined to evaluate the performance of the trained model as shown in Fig. 3.12

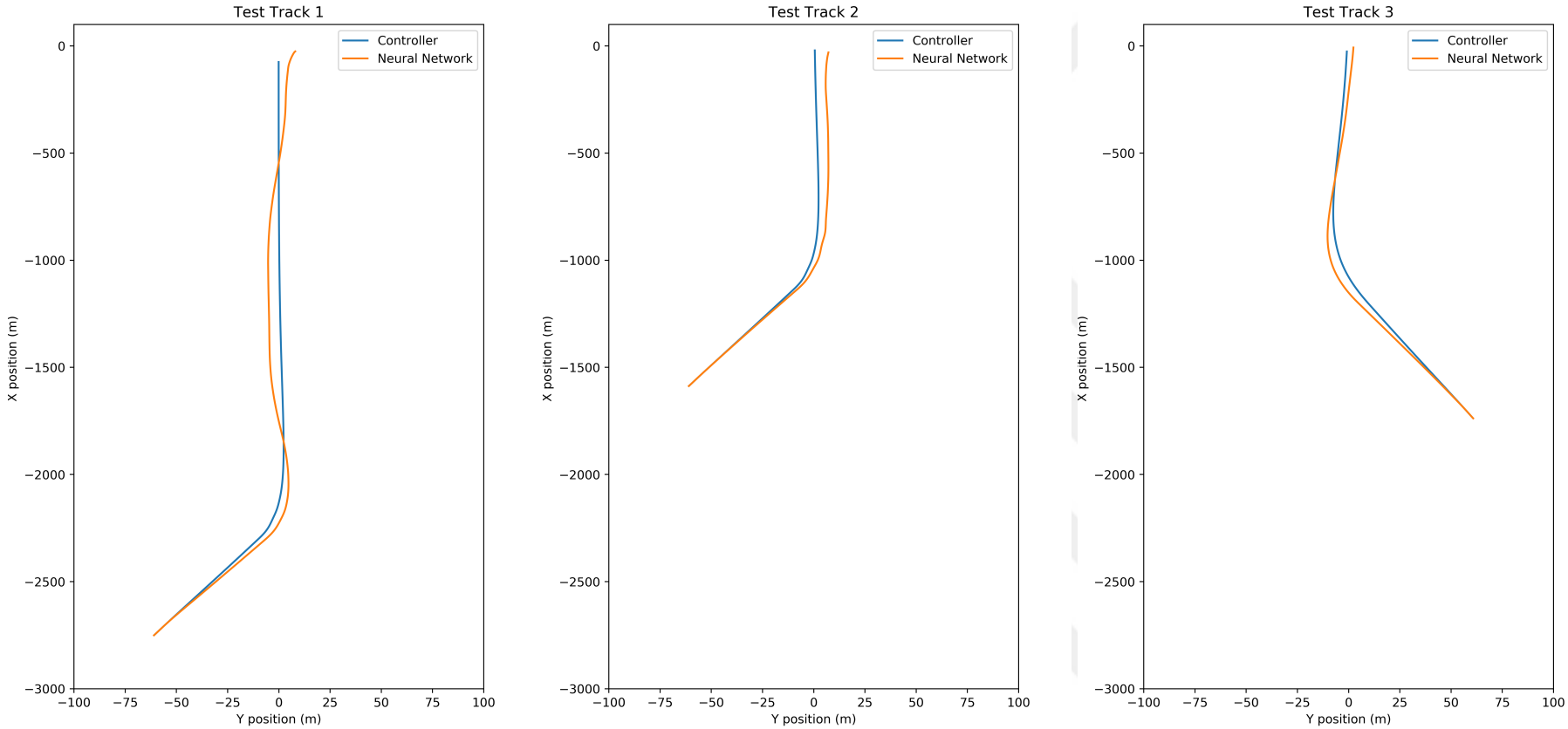


Figure 3.12 : Test tracks.

At the beginning of the approach phase guidance algorithm calculates the initial desired heading angle according to the initial position and simulation starts with calculated orientation. Because of that reason error between desired and current heading angle is zero at the starting of simulations. Desired heading angle does not change until the start of the glide phase. Trajectory differences at approach phase are mainly due to the prediction error of the neural network and since there is no roll maneuver at this phase performance of neural network stays in reasonable range. At the glide phase guidance algorithm starts change desired heading angle which leads to rolling roll maneuver. Performance of neural network decreases at that stage but still remains in a suitable range. For the flare phase, guidance algorithm tries to compensate trajectory differences in long-term and neural network follows the path. In the final stage frames, the runway is closer and bigger. So the performance of the convolution filters slightly reduces and after zero point trajectory diverges. Since there is no touchdown scenario in dataset this is an expected result.

3.2 Steering and Speed Prediction for Self-Driving Cars

3.2.1 Driving policies

We begin with giving definitions of the used terms in order to explain driving policies in detail.

A set of states S for the car in this paper is an environment model and $s \in S$ is one of the states for the car in that environment. Observation of the state s is defined as $\phi(s) \in \Phi(S)$ where $\Phi(S)$ is the observation set for all states. $a(s) \in A(S)$ will be driving action at observation $\phi(s)$ where $A(S)$ is the set of all possible actions.

A set of driving policies Π is defined as:

$$\Pi : \Phi(S) \rightarrow A(S) \tag{3.3}$$

which is a mapping from state observations $\phi(s)$ to driving actions $a(s)$ such as steering, throttle, brake, etc.

Two distinct driving policies are defined throughout the paper. The first one is a reference/expert policy $\pi^* \in \Pi$ that drives the car with a reasonable performance that

we want to imitate. A reference policy in an autonomous driving scenario is usually chosen as actions of a human driver. Variants of DAgger algorithms, however, have mislabeling problem in case of the human driver, since drivers do not have feedback feelings from their actions and they can give incorrect reactions to the given states. In order to overcome the mislabeling problem, we have used a rule-based controller which contains speed and steering controllers, as a reference/expert policy in this paper. The second policy is a primary policy $\pi_0 \in \Pi$ that is trained to drive a car. This policy is a sub-optimal policy according to the reference/expert policy since it is trained on a subset of observation set $\Phi(S)$.

Training a primary policy to mimic a reference policy is called imitation learning or learning by demonstration. One of the most common methods for imitation learning is based on supervised learning techniques. The loss function for the supervised learning is [9]:

$$l_{supervised}(\pi, \pi^*, D_0) = \frac{1}{N} \sum_{i=1}^{N} ||\pi(\phi(s_i)) - \pi^*(\phi(s_i))||^2 \tag{3.4}$$

$wherel_{supervised}$ refers to l^2-Norm between trained and reference policy actions.

A primary policy is defined as a policy that minimizes the supervised l^2-Norm equation as follows.

$$\pi_0 = \arg\min_{\pi} l_{supervised}(\pi, \pi^*, D_0) \tag{3.5}$$

Minimization of the loss function can be challenging since it is known that the relation between image frames and driving actions is highly nonlinear. So, we have used a deep neural network architecture to find an optimal solution for the primary policy.

3.2.2 Network architecture

The earlier works in end-to-end learning for self-driving cars focus on computing only the steering angle from a single image or a sequence of images. In order to reach a higher level of autonomy in the end-to-end framework, the longitudinal control component is also required. In this work, we utilize the multi-task model proposed in [7] as our baseline, which is capable of generating both longitudinal and lateral

control inputs for the car. In addition, we utilize a speed controller rather than the classical throttle/brake commands for the longitudinal control. The steering action is predicted from the raw image inputs taken from the cameras located in front of the vehicle through convolution layers, and the speed is predicted from a sequence of speed profiles through a Long-Short Term Memory (LSTM) layer. There exists a single-direction coupling between the longitudinal controller (speed controller) and the lateral steering actions. In particular, the speed of the vehicle has a significant impact on the prediction model, since entering a turn with low speed represents different dynamics for the lateral controller when compared to a high-speed maneuver. Moreover, the straight trajectory dominates the whole other trajectory types (e.g. turn left, turn right), therefore, the trained network will be biased toward the straight trajectory. In order to recover from this issue, we decided to define various trajectory types including all major maneuvers such as straight, turn left, turn right and low and high-speed scenarios, by which the devised model will learn the other less-occurring maneuvers.

The model architecture is shown in Fig 3.13. It takes the current observation and the past speed profile and returns steering action, speed action and the class of the trajectory segment. The multi-task network predicts the steering angle through a visual encoder using a stack of convolution and fully-connected layers. In the first two convolution layers (Conv1 and Conv2), large kernel size is adopted in order to better capture the environment features, which is suitable for the front-view camera. Inputs and kernels of the each convolution layer is denoted by "$\#channels@input\,height \times input\,width$" and "$kernel\,height \times kernel\,width \times \#channels$" and each fully connected layer is denoted by "$FC-$ size of neurons". The speed and trajectory class are predicted through a concatenation of visual encoder and feedback speed features. The speed features are extracted by an LSTM layer followed by fully-connected layers. ReLU (Rectified Linear Unit) is used as the activation function for all layers. Mean absolute error is the loss function for both speed and steering angle predictions as regression problems. On the other hand, the cross-entropy applies to the trajectory classifier as a classification problem.

The multi-class classifier highlighted in Fig. 3.13 extends the safeDAgger method to a novel algorithm devised in this paper. The trajectory classes are defined as follows:

$$c(\pi, \phi(s)) = \begin{cases} 1, & \text{Safe Trajectories} \\ 2, & \text{Unsafe LL} \\ 3, & \text{Unsafe HL} \\ 4, & \text{Unsafe LR} \\ 5, & \text{Unsafe HR} \\ 6, & \text{Unsafe LS} \\ 7, & \text{Unsafe HS} \end{cases} \tag{3.6}$$

where X in "Unsafe XY" refers to Low(L) and High(H) speeds and Y refers to going Straight(S), Left(L) and Right(R) turns, respectively. Low and high speeds with combinations of left, straight and right turns covers almost all unsafe trajectories. Same combinations also applicable for safe trajectories but since it is not needed to call expert policy in safe trajectories, we define only one class for the safe trajectories.

The multi-class classifier takes the partial observation of the state $\phi(s)$ which contains the visual perception and the past speed profile and returns a label indicating in which part of the trajectory the policy will likely to deviate from the reference policy π^*.

The labels for training the model was generated through one-hot-encoding method, defined by sequential decisions; first, it was decided whether the policy is safe by measuring its distance from the reference policy through l^2-Norm metric, which is as follows:

$$c_{safe}(\pi, \phi(s)) = \begin{cases} 0, & ||\pi(\phi(s)) - \pi^*(\phi(s))|| > \tau_{safe} \\ 1, & \text{otherwise} \end{cases} \tag{3.7}$$

where τ_{safe} is a predefined threshold and can be chosen arbitrarily. Furthermore, in order to distinguish between low-speed and high-speed turn trajectories, steering threshold τ_{turn}, speed thresholds for turn maneuver $\tau_{speed,turn}$ and straight trajectory $\tau_{speed,straight}$ are defined heuristically based on the response of the car dynamics in these trajectories. The threshold values for this work is depicted in Table 3.2.

Table 3.2 : Threshold values in labeling process.

Parameter	Threshold value
τ_{safe}	0.5
τ_{turn}	0.25°
$\tau_{speed,turn}$	$10\ m/s$
$\tau_{speed,straight}$	$13.75\ m/s$

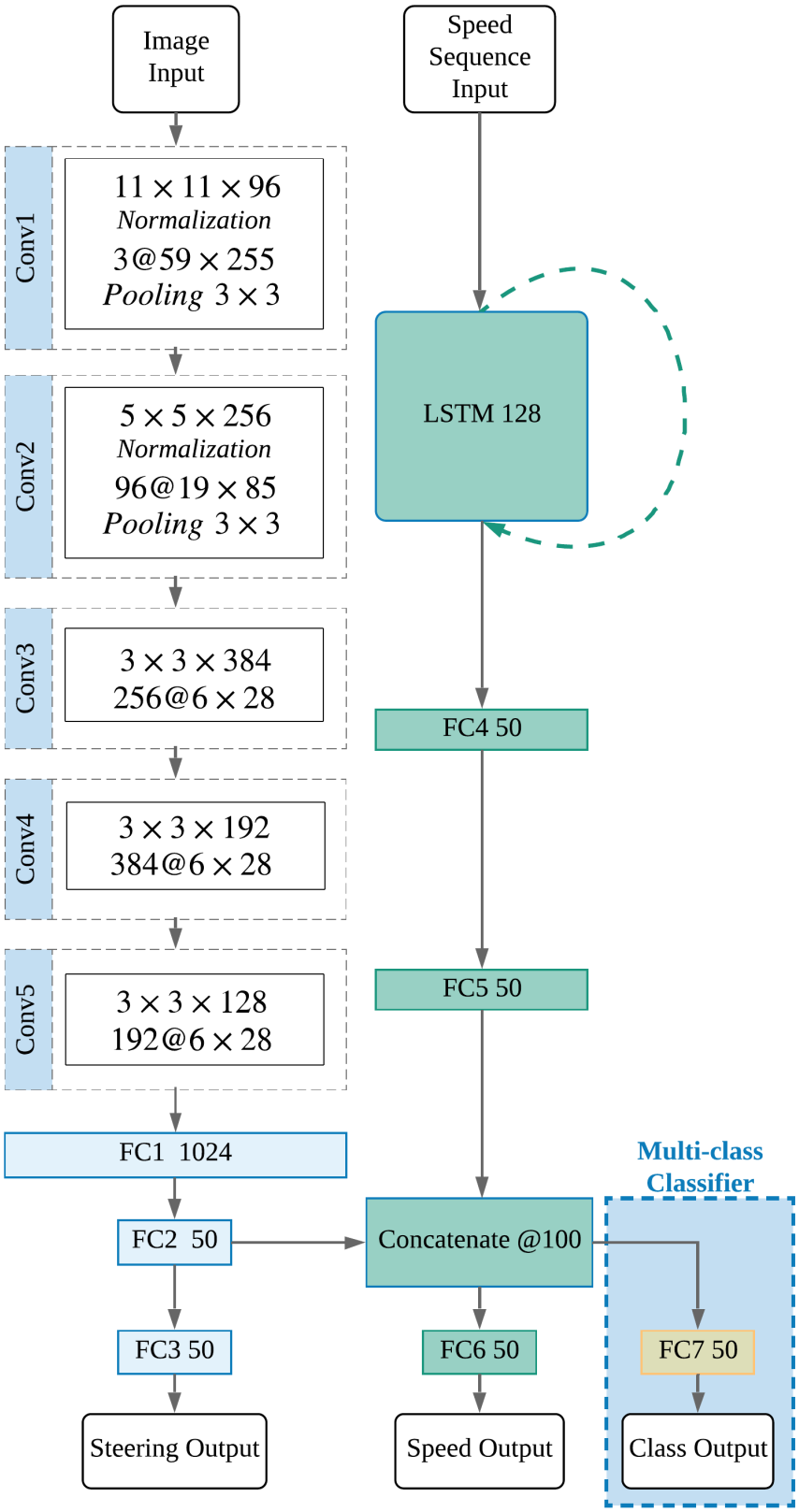


Figure 3.13 : Sample-efficient Selective SafeDAgger model.

where τ_{safe} as 0.5 yields 0.25° for the steering angle and $1\,m/s$ for the speed difference between the network prediction and expert/reference policy output.

3.2.3 System setup

3.2.3.1 Simulator

AirSim is used in this work, which is an Unreal Engine Plugin based simulator for drones and cars established by Microsoft to create a platform for AI studies to develop and test new deep learning, computer vision and reinforcement learning algorithms for autonomous vehicles with photo-realistic graphs in simulations [17]. It has built-in APIs for interfacing with Python coding language which is one of the dominant coding languages of AI research. It is suitable also to be used for deep learning approaches in car perception algorithms since simulator has a photo-realistic environment. Furthermore, custom environments or scenarios can be created by using the engine editor.

The road track for the training process of the algorithm is devised in a way to capture all defined scenarios in this work. The geometry of the custom created training track is shown in Fig. 3.14, in which all the trajectory classes are illustrated.

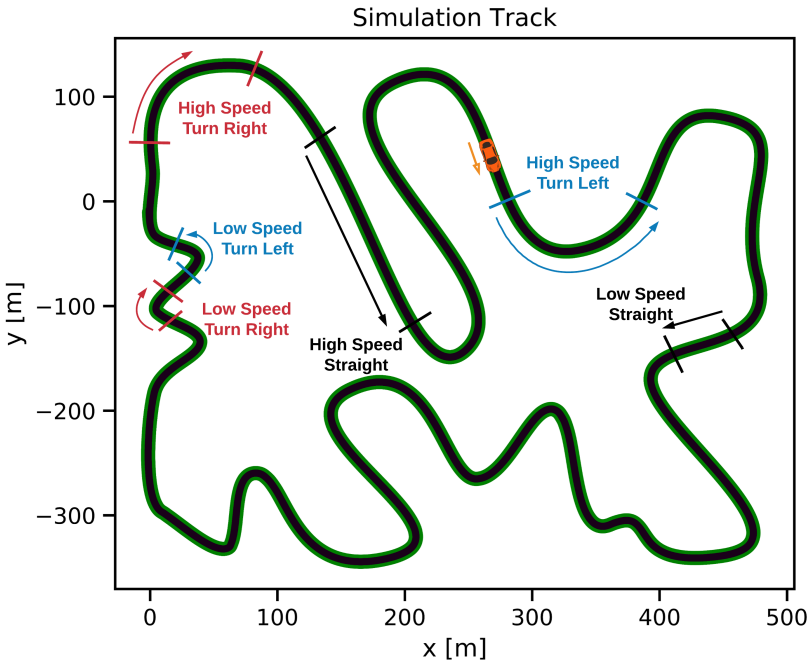


Figure 3.14 : Train set track.

Representative power of the training set can be increased by collecting data from unseen observations. With that reason, 2 additional cameras were added to the front-facing camera with an angle of α to imitate turning-like maneuvers as shown in Fig. 3.10. Airsim APIs provide ground truth labels for the front-facing camera frames, but ground truth labels for the left and right cameras should be adjusted with a threshold as in Eqn. 3.8.

$$\begin{bmatrix} L_l \\ L_r \end{bmatrix} = \begin{bmatrix} L_{c_{steering}} + \alpha & L_{c_{speed}} - p_{speed} \\ L_{c_{steering}} - \alpha & L_{c_{speed}} - p_{speed} \end{bmatrix} \tag{3.8}$$

where L_l, L_r, $L_{c_{steering}}$ and $L_{c_{speed}}$ refer to the ground truth for the left and right cameras, center camera steering and speed actions respectively. In the turning case, the ground truth speed of the vehicle is adjusted by a parameter p_{speed} which is chosen as 4 m/s heuristically.

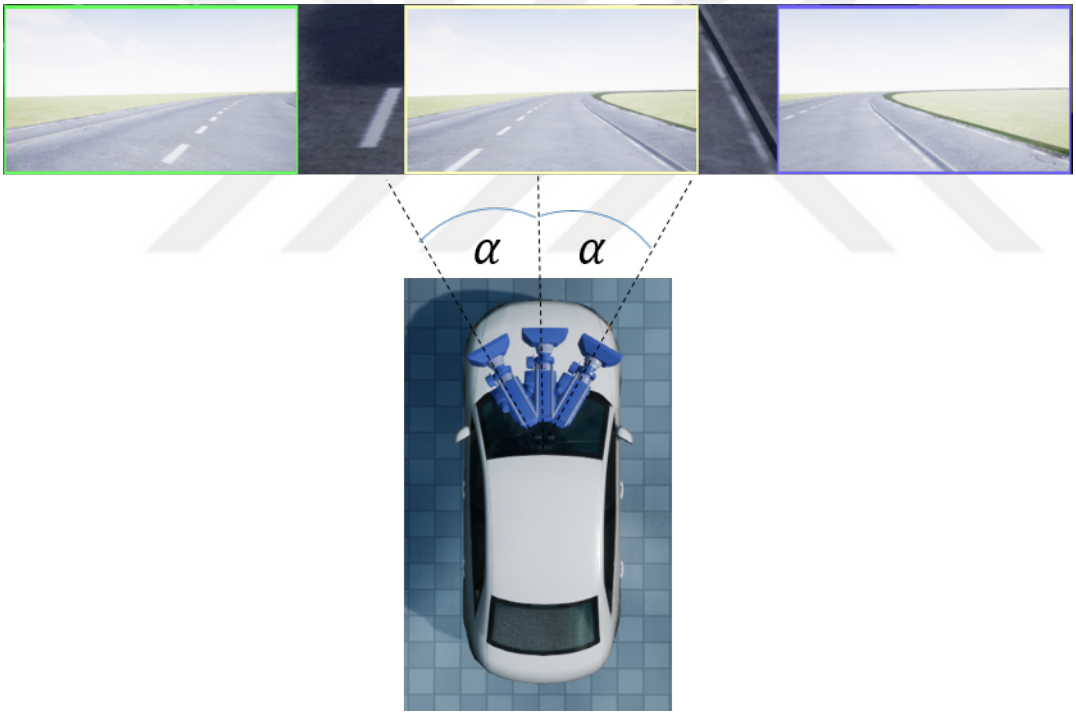


Figure 3.15 : 3 camera view with an α angle.

3.2.3.2 Data preprocessing

A couple of techniques were utilized in the preprocessing phase. In order to reduce the computational cost, the input raw image was down-sampled to the size of 144 256 3 (RGB) and a Region of Interest (ROI) defined with the size of 59 255 to cover almost the entire road and ignore the features above the horizon. Moreover, to improve the

convergence rate and robustness of the neural network model, the processed image was normalized to the range [0,1] and augmented by randomly changing the brightness of the image with a scale of 0.4. The normalization was done by dividing all image pixels by 255. Also, the brightness change was done by transforming the image from RGB to HSV space and transform back to RGB space with the scaled value.

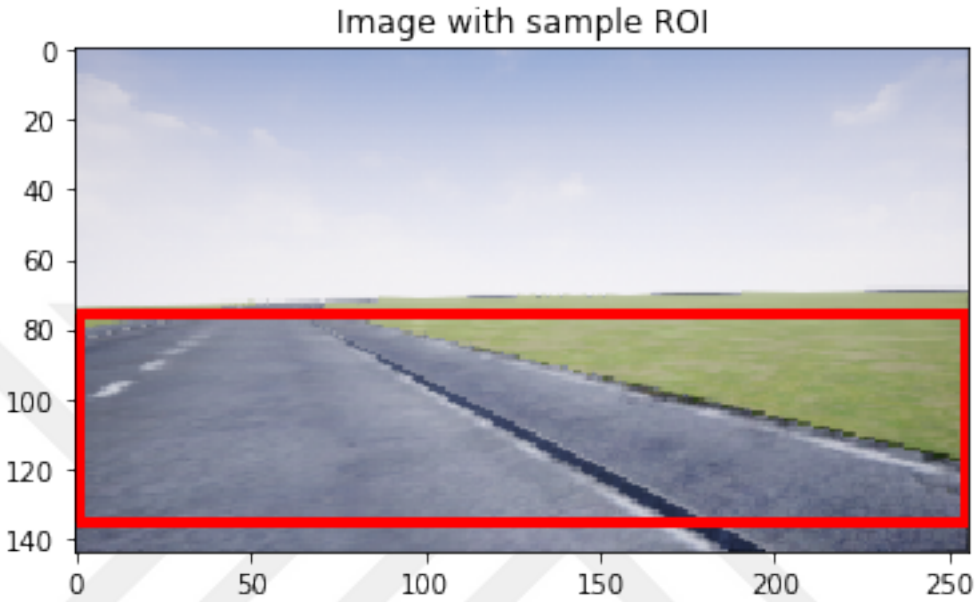


Figure 3.16 : Region of Interest.

3.2.3.3 Expert policy

In order to automatize the data collection part of the algorithm, a rule-based expert policy is defined as follows (see Fig. 3.17):

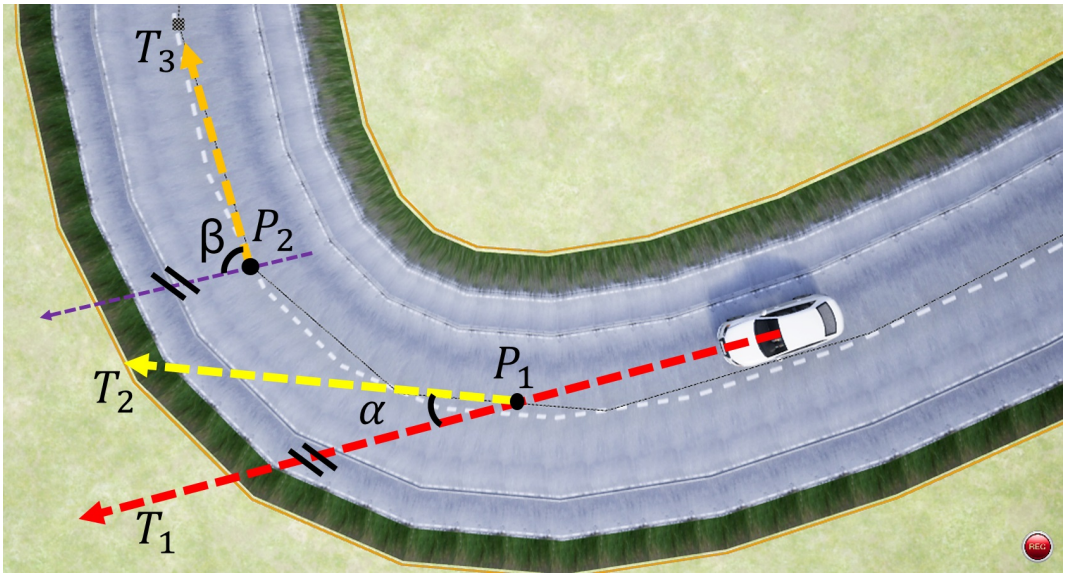


Figure 3.17 : Expert policy.

For the steering action, T_1 is a tangent line to the road spline at the position of the car and P_1 is a point on road spline with l_{ref} distance along spline from that positions. Tangent line at P_1 according to road spline is T_2. Angle between T_1 and T_2 which is α will be expert steering action as shown in equation 3.9.

$$a_{steering} = \alpha = \arccos\left(\frac{T_1 \cdot T_2}{\|T_1\| \, \|T_2\|}\right) \tag{3.9}$$

For the speed action, P_2 is a point on the road spline with a distance l_{P_2} as shown in equation 3.10 from the position of the car along the road spline,

$$l_{P_2} = l_{ref} V_{current} k_{steering} \tag{3.10}$$

where $V_{current}$ is current speed and $k_{steering}$ is a fine tuned constant. Tangent line at P_2 according to the road spline is T_3. Expert speed action will be: be

$$a_{speed} = V_{cruise} - \beta k_{speed} \tag{3.11}$$

where V_{cruise} is a pre-defined cruise speed, k_{speed} is a fine tuned gain and β is an angle between T_2 and T_3.

For our implementation, the parameters are chosen as $l_{ref} = 1$ m, $k_{steering} = 5$, $V_{cruise} = 13.8$ m/s and $k_{speed} = 10$.

3.2.4 Training

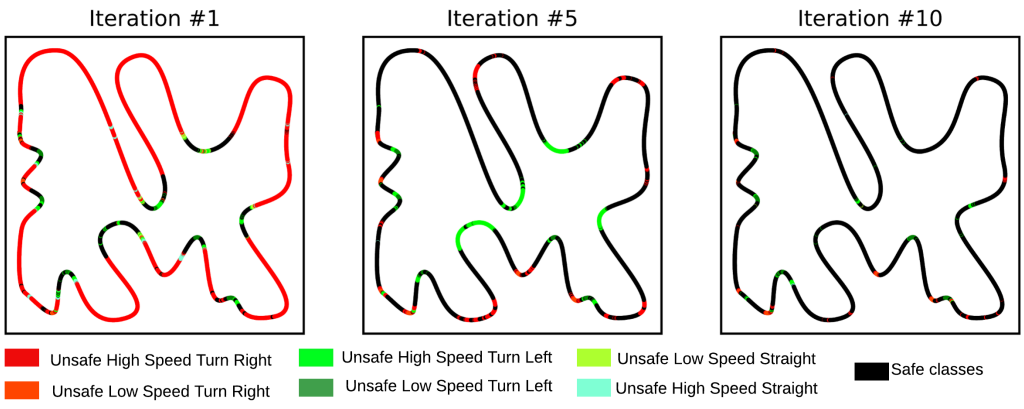


Figure 3.18 : Convergence rate of the proposed model; It shows the improvement of the model as the number of dataset aggregation iterations increases.

For the training of the primary policy π_0 in the algorithm, dataset D_0 which contains 2800 image data were collected by using expert policy π^*. Nesterov Adam Optimizer

(Nadam) was used as an optimizer for the training of the network with the initial learning rate of 10^{-5} and moment of 0.99. The Training was continued for 10 epochs with the batch size of 32.

Trained primary policy π_0 is tested on the pre-collected dataset to classify trajectories and calculate the l^2-Norm of each sample in the dataset. Weakness of the network over trajectory segments is determined by a coefficient of weakness which is defined as in equation 3.12,

$$c_i = \frac{N_{L2_i}}{N_i} \times \mu_{L2,i} \tag{3.12}$$

where μ, σ are mean and standard deviations for the l^2-Norm of the samples for class$_i$. N_{L2_i} is the total number of samples in class$_i$ that l^2-Norm of samples fall in the region of one σ away from the mean μ. N_i is the total number of samples in class$_i$.

Once the weakness coefficients are calculated, trajectory classes are sorted according to their weakness coefficients and the two of the most dominant unsafe classes will be chosen for data aggregation as shown in Table 3.3. Additionally, the classes with the mean l^2-Norm lower than 1, will be chosen as allowable classes other than the dominant 2 classes.

In Table 3.3 it can be seen that weakness coefficients for the class *LS* and *HS* are quite low and never chosen as weak classes. The initial dataset for the training of the policies is biased toward *LS* and *HS* classes and l^2-Norms in those classes are low, which lead to low weakness coefficients. Moreover, training track does not have many samples from class *LR* so that weakness coefficients for the class *LR* is also low.

Table 3.3 : Coefficient of weakness for each class.

# Iter.	*LL*	*HL*	*LR*	*HR*	*LS*	*HS*
1	0.004	**0.321**	0.019	**0.694**	0.002	0.010
2	**0.505**	0.122	0.037	**0.278**	0.001	0.023
3	**0.635**	0.264	0.028	**0.607**	0.001	0.062
4	**0.751**	0.515	0.046	**0.646**	0.001	0.010
5	0.018	**0.678**	0.034	**0.755**	0.001	0.010
6	0.009	**0.752**	0.039	**0.849**	0.000	0.006
7	0.717	**0.790**	0.038	**0.780**	0.001	0.004
8	0.028	**0.787**	0.017	**0.794**	0.001	0.006
9	**0.670**	0.634	0.011	**0.713**	0.001	0.005
10	0.012	**0.768**	0.020	**0.809**	0.001	0.003

After determination of the weak and allowable classes, the data aggregation phase begins. In the phase of data aggregation, policy π_i drives the car in order to collect 10 batches of data in dominant classes. If policy faces with dominant classes, the expert policy takes control of the car and samples are taken in that time labeled and reserved for aggregation. If policy π_i faces with allowable classes which are actually unsafe classes, it continues to drive the car. For all the other unsafe classes, the expert policy takes control of the car with the query limit of 10 batch-size. When the number of query reaches that limit or 10 batches of data is collected, data aggregation freezes and training starts with the new aggregated dataset D_i.

After the training, π_i becomes π_{i+1} and determination of dominant weak classes on the pre-collected data are repeated and relevant data will be collected. This process will be repeated for 10 iterations. As shown in Fig. 3.18, in the dataset aggregation iteration number 1, a big fraction of dataset is unsafe, and as it proceeds to recover from the most problematic cases, the model error converges. The progress of this process can be seen from iteration number 1 to 10.

3.2.5 Results

In Fig. 3.19, we present the performance of the Selective SafeDAgger with using metric of l^2-Norm in each class during the training process. For the first iteration, *HR* and *HL* are chosen as weak classes and data for new dataset comes from those classes by querying expert policy. İt is clearly seen that in the second iteration l^2-Norms drops for all classes by using aggregated dataset. It should be noted that the performance of the policy for the other classes is also increased without querying expert policy for those classes which are not the case for the SafeDAgger. Sequential decision making is the main idea behind this behavior. In SafeDAgger, when policy shifts from nominal conditions, the expert policy is called and the new dataset is collected until the safety criterion is met which leads to an unnecessary query of the expert policy. On the other hand, Selective SafeDAgger tries to solve the problem from the beginning by finding problematic classes. In addition, it can be seen that after the seventh iteration norm of all classes drops below the allowable threshold which means that resultant dataset covers almost all trajectory classes as seen in Fig. 3.18.

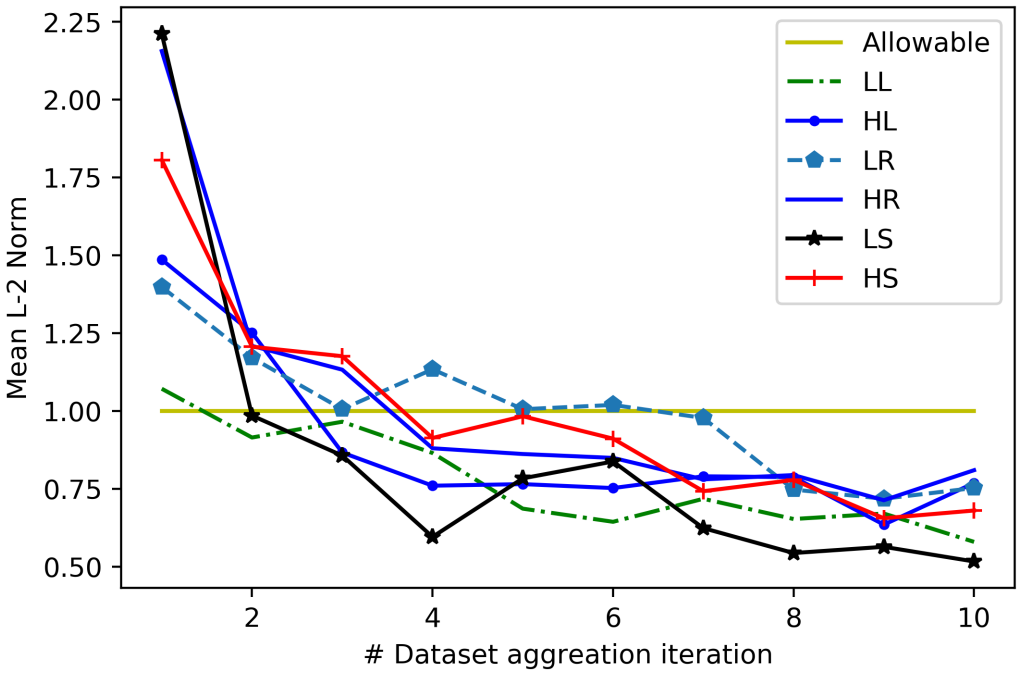


Figure 3.19 : Performance of the Selective SafeDAgger algorithm for all classes at each aggregation iteration.

Table 3.4 : Query to expert.

	Selective SafeDAgger						SafeDAgger
	LL	*HL*	*LR*	*HR*	*LS*	*HS*	unsafe
Iteration 1	0	127	38	155	0	0	320
Iteration 2	0	44	0	228	0	48	320
Iteration 3	19	63	0	238	0	0	320
Iteration 4	27	12	0	281	0	0	320
Iteration 5	0	165	0	155	0	0	320
Iteration 6	31	189	0	100	0	0	320
Iteration 7	0	93	0	227	0	0	320
Iteration 8	2	162	0	156	2	5	320
Iteration 9	83	0	0	237	0	0	320
Iteration 10	0	205	0	115	0	0	320
Total	3200						3200

The trained model is tested at each iteration by taking 10000 samples from the environment and mean l^2-Norms are calculated, accordingly. Fig. 3.20 shows that selective SafeDAgger method has better performance in all iterations than the SafeDAgger method even though both methods have the same amount of query to the expert as seen Table 3.4.

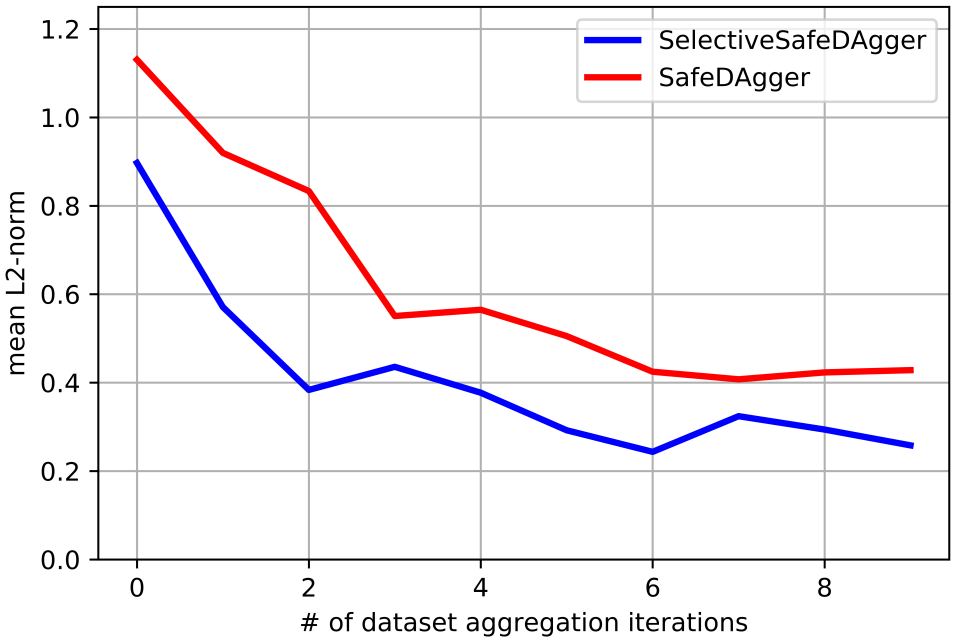


Figure 3.20 : l^2-Norm of prediction and ground truth over 10000 samples at each iteration.

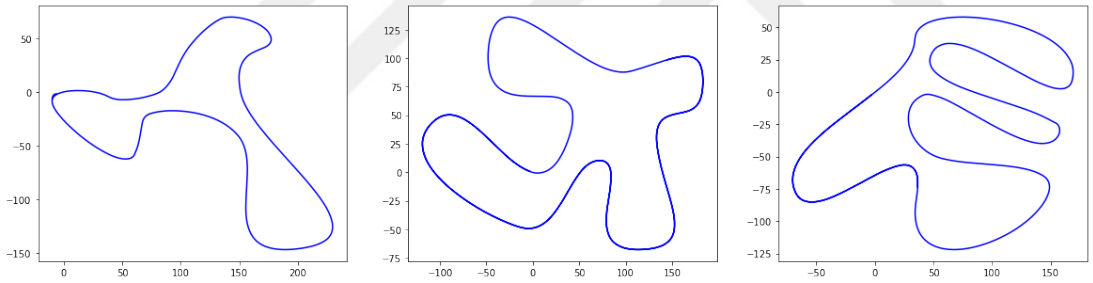


Figure 3.21 : Geometry of test tracks.

Table 3.5 : Mean l^2-norm on unseen test tracks.

	Selective SafeDAgger	SafeDAgger
1. Test Track	**0.4794**	0.5518
2. Test Track	**0.3295**	0.4986
3. Test Track	**0.3254**	0.3632

In order to evaluate the generalization performance of the proposed method, 3 unseen test tracks were devised and used to test the proposed method. The generalization performance of the Selective SafeDAgger is depicted in Table 3.5, which shows its superiority over SafeDAgger method. The selective property of the proposed algorithm

will define the unsafe cases that dominate all other classes, which results in faster convergence of the model error compared to other other dataset aggregation methods.

4. CONCLUSIONS

In this work, we implemented a Selective SafeDAgger algorithm which is sample-efficient in the selection of dataset aggregation samples. The proposed algorithm evaluates the performance of the trained policy and determines the weakness of the policy over different observation classes and recovers the policy from those specific observation classes. Verification of algorithm done with two experiments which are steering and speed prediction for self-driving cars and visual landing of an F-16 aircraft. Result of those two experiments showed that our method outperforms the SafeDAgger algorithms in term of sample-efficiency and convergence rate.

REFERENCES

[1] **Leonard, J.**, **How, J.**, **Teller, S.**, **Berger, M.**, **Campbell, S.**, **Fiore, G. and ...Karaman, S.** (2008). A perception-driven autonomous urban vehicle, *Journal of Field Robotics*, *25*(10), 727–774.

[2] **Chen, Z. and Huang, X.** (2017). End-to-end learning for lane keeping of self-driving cars, *2017 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, pp.1856–1860.

[3] **Pomerleau, D.A.** (1989). Alvinn: An autonomous land vehicle in a neural network, *Advances in neural information processing systems*, pp.305–313.

[4] **Bojarski, M.**, **Del Testa, D.**, **Dworakowski, D.**, **Firner, B.**, **Flepp, B.**, **Goyal, P. and ...Zhang, J.** (2016). End to end learning for self-driving cars, *arXiv preprint arXiv:1604.07316*.

[5] **Xu, H.**, **Gao, Y.**, **Yu, F. and Darrell, T.** (2017). End-to-end learning of driving models from large-scale video datasets, *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp.2174–2182.

[6] **Kim, J. and Canny, J.** (2017). Interpretable learning for self-driving cars by visualizing causal attention, *Proceedings of the IEEE international conference on computer vision*, pp.2942–2950.

[7] **Yang, Z.**, **Zhang, Y.**, **Yu, J.**, **Cai, J. and Luo, J.** (2018). End-to-end Multi-Modal Multi-Task Vehicle Control for Self-Driving Cars with Visual Perceptions, *2018 24th International Conference on Pattern Recognition (ICPR)*, IEEE, pp.2289–2294.

[8] **Ross, S.**, **Gordon, G.J. and Bagnell, J.A.** (2011). A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, *AISTATS*.

[9] **Zhang, J. and Cho, K.** (2017). Query-Efficient Imitation Learning for End-to-End Simulated Driving, *AAAI*.

[10] **Menda, K.**, **Driggs-Campbell, K.R. and Kochenderfer, M.J.** (2018). EnsembleDAgger: A Bayesian Approach to Safe Imitation Learning, *CoRR*, *abs/1807.08364*.

[11] **Thomas, S.**, **Kwatny, H.G. and Chang, B.C.** (2004). Nonlinear reconfiguration for asymmetric failures in a six degree-of-freedom f-16, *American Control Conference, 2004. Proceedings of the 2004*, volume 2, IEEE, pp.1823–1828.

[12] **Nguyen, L.T.**, **Ogburn, M.E.**, **Gilbert, W.P.**, **Kibler, K.S.**, **Brown, P.W. and Deal, P.L.** (1979). Simulator study of stall/post-stall characteristics of a fighter airplane with relaxed longitudinal static stability.[F-16].

[13] **Stevens, B.L.**, **Lewis, F.L. and Johnson, E.N.** (2015). *Aircraft control and simulation: dynamics, controls design, and autonomous systems*, John Wiley & Sons.

[14] **Napolitano, M.R.** (2012). *Aircraft dynamics: From modeling to simulation*, J. Wiley.

[15] **Singh, S. and Padhi, R.** (2009). Automatic path planning and control design for autonomous landing of UAVs using dynamic inversion, *American Control Conference, 2009. ACC'09.*, IEEE, pp.2409–2414.

[16] **Pashilkar, A.A.**, **Ismail, S.**, **Ayyagari, R. and Sundararajan, N.** (2013). Design of a nonlinear dynamic inversion controller for trajectory following and maneuvering for fixed wing aircraft, *Computational Intelligence for Security and Defense Applications (CISDA), 2013 IEEE Symposium on*, IEEE, pp.64–71.

[17] **Shah, S.**, **Dey, D.**, **Lovett, C. and Kapoor, A.** (2018). Airsim: High-fidelity visual and physical simulation for autonomous vehicles, *Field and service robotics*, Springer, pp.621–635.

CURRICULUM VITAE

Name Surname: Yunus BICER

Place and Date of Birth: Adiyaman - 13/03/1992

E-Mail: biceryu@itu.edu.tr

EDUCATION:

- **B.Sc.:** 2016, Istanbul Technical University, Faculty of Aeronautics and Astronautics, Aeronautical Engineering Department
- **M.Sc.:** 2019, Istanbul Technical University, Faculty of Aeronautics and Astronautics, Aeronautics and Astronautics Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2019 - Present, Research Assistant at Faculty of Aeronautics and Astronautics, Istanbul Technical University.

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- Bicer, Y., Moghadam, M., Sahin, C., Eroglu, B., Ure, N.K. "Vision-based UAV Guidance for Autonomous Landing with Deep Neural Networks." *2019 AIAA Science and Technology Forum and Exposition (AIAA SciTech 2019)*, San Diego, California, USA, January 7-11 2019
- Bicer, Y., Alizadeh, A., Ure, N.K., Erdogan, A., Kizilirmak, O. "Sample Efficient Interactive End-to-End Deep Learning for Self-Driving Cars with Selective Multi-Class Safe Dataset Aggregation." *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2019)*