Title
A Hierarchical Few-Shot Learning Framework for Visual Navigation of Autonomous Vehicles

Author
Shi, Debo

Publication Date
2023

Peer reviewed|Thesis/dissertation

A Hierarchical Few-Shot Learning Framework for Visual Navigation of Autonomous Vehicles

By

Debo Shi
THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical and Computer Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Iman Soltani, Chair

Junshan Zhang

Chen-Nee Chuah

Committee in Charge

2023

Name: Debo Shi

Student ID: 920249051

A Hierarchical Few-Shot Learning Framework for Visual Navigation of Autonomous Vehicles

Abstract

Autonomous driving is a challenging task given the variety and complexity of drive scenarios. Meta-learning, particularly few-shot learning, is a common approach to tackle a classification task with limited number of training data for each classes. In this project, we focus on a specific scenario of visually navigating vehicles along an unseen route through recognizing the waypoints with only a few available waypoint images and no need to re-train for a new course. To attack the targeted task, we proposed a hierarchical framework with two deep models in which the highlight is the StampNet, a few-shot learning architecture with a covariance estimator for recognizing waypoints. To train the models, we collected an associated indoor dataset with images along the routes in various buildings on UC Davis campus, and quantified the performance of the tests to explore the best hyperparameters. Additionally, to demonstrate the effectiveness of the proposed approach, the framework was implemented on a customized mini vehicle for an online test that run the models in real time to navigate the vehicle in unseen courses.

Acknowledgments

I would like top express my gratitude to my advisor, Professor Iman Soltani for his invaluable mentorship and support throughout my study at UC Davis. I also want to thank my committee members, Professor Junshan Zhang and Professor Chen-Nee Chuah for providing their insightful criticism and suggestion on my thesis. I truly appreciate all the work my committee have put into my thesis.

Special thanks to Dr. Francois Charette at Ford Motor Company for his suggestions in project discussions, and the mini autonomous vehicle platform that used for data acquisition and online testing.

I would also like to acknowledge my collaborators at the Laboratory for AI, Robotics and Automation at UC Davis. Amin Ghafourian Momenzadeh did the preliminary research in this project and proposed the fundamental architecture of the few-shot learning model used in this work, and he also provided helpful suggestions during the project discussion. Also thanks Ian Thomas Chuang for his work on designing and training the lower-level navigation model, as well as modifying the hardware platform. Together with him, Mohammed Shais Khan also helped with collecting the dataset at UC Davis campus. Their effort is indispensable in this project.

Contents

List of Figures

CHAPTER 1

Introduction

In this thesis, we focus on addressing an autonomous driving task using a visual navigation approach that requires minimal learning samples. Specifically, we consider a scenario in which a vehicle must navigate an unseen route that can be segmented into sections connected by waypoints, at which the vehicle should change its driving state. Prior to departure, only 15 images of each waypoint are available to populate the memory. Then, the vehicle will determine if the next waypoint is reached by processing real-time images and make decisions regarding maintaining the current driving state or switching to the next state.

Machine learning based algorithms have been widely applied in machine vision tasks in the domains of classification [**14**, **18**, **32**], segmentation [**13**, **22**, **28**] and generation [**11**, **16**], etc. However, such models often require to be trained with a large amount of data, and the capability of the trained model is limited in solving a specific task covered by the training set. In contrast, humans are able to learn to visually distinguish objects by seeing only a few pictures. This requires the capability to capture the distinct features of an object to recognize it in the future. The objective of meta-learning is to endow a machine learning model with the ability to generalize features based on a small number of training examples, thus enabling the model to adapt to novel tasks with minimal data. In this section, a facet of meta-learning called few-shot learning is utilized in a hierarchical framework to help a vehicle visually navigate throughout unseen paths based on only a few pictures of a set of landmarks.

We tackle the few-shot learning problem under the assumption that there exists a non-linear mapping from the input data to encoding in a latent space where data encodings in a same class are close to the center of the corresponding cluster while the embedded negative samples are relatively far away. In this work, we adopt StampNet, a variational few shot

framework developed earlier in the Laboratory for AI, Robotics and Automation. We train StampNet in a few-shot learning paradigm to generate the feature embedding and predict the corresponding covariance for each landmark. We also explore various metrics to enhance the classification performance. The result of this work is a two-level hierarchical framework where the few-show learning model provides high-level navigation decision making to move the vehicle towards the desired destination, while a lower-level navigation module keeps the vehicle on a drivable path and avoids obstacles.

CHAPTER 2

Related Work

2.1. Hierarchical Learning-Based Approach

The hierarchical control system is a widely used architecture employed in autonomous vehicles, which systematically decomposes the intricate decision-making process into manageable sub-tasks, each executed by specialized modules. Form the decision making aspect, existing hierarchies applied in self-driving [**35**, **39**] are all constructed by a higher-level module that plans the global path and a lower-level module that navigate the vehicle locally, under the guidance of the higher-level system. By modularizing the pipeline, the self-driving task is simplified and makes it possible to leverage the strength of different algorithms to solve the corresponding sub-tasks. There have been various hierarchical structures applied in the vision-based autonomous driving vehicles. Affordance learning method [**4**, **30**] uses a CNN as the lower-level module to map images to a set of predefined perception indicators associate with the affordance of the driving states. On the other hand, reinforcement learning has been adopted to planning the global course [**5**, **15**, **31**]. These approaches are flexible with the different modules since they optimize the models separately, even combining a learning-based model with a classical control algorithm [**31**] is possible. Inspired by these works, our architecture is constructed based on a few-shot learning model and a regression model. Benefited from the flexibility of a modular structure, the capabilities of the sub-modules are maximized by training each with different training methods and different data sampling strategies.

2.2. Visual Waypoint Identification

For a waypoint-based autonomous navigation plan, the driving task fails if the vehicle misses a waypoint or changes the driving state at a wrong time. Thus, correctly identifying

waypoints is a key problem in making high-level decisions. Many visual-based methods have been explored to identify the waypoints. An approach to achieving this objective involves detecting pre-designated visual markers at waypoints [**9**, **23**]. These markers are visually distinct from the environment, so they could be easily detected through simple and generic image processing algorithms. Additionally, more information can be obtained from the markers by scanning the QR code on them [**23**]. However, the application of these methods are limited by aesthetic concerns because the task-specific markers are often inharmonious with the environment. Furthermore, such markers are not available in unseen route. Rather than relying on markers, learning-based image processing technique has also been adopted to extract general waypoint-specific features including corner detection [**38**,**41**] and path segmentation [**6**]. Benefiting from explicitly focusing on hand-engineered features at waypoints, these models are robust to environmental changes, such as random pedestrians, if the targeted features are not affected. What these methods have in common is that they all need laborious preparation for a new path, which makes them hard to deploy to new tasks. On the contrary, our approach can be implemented easily since our few-shot learning based algorithm only need a few images of each waypoint, while featuring reasonable robustness.

2.3. Meta-Learning

Also known as "learning to learn", meta-learning techniques are designed to enable systems to quickly adapt to new tasks and environments, by learning general patterns from a set of previously learned tasks in a data-driven way. Over the past few decades, there has been a significant amount of research exploring meta-learning. Existing meta-learning approaches can be divided into three categories: metric-based, model-based and optimization-based methods. Metric-based techniques [**17**, **33**, **37**] focus on exploring an effective measure to quantify the similarity between data in a feature space. Model-based approaches [**7**,**24**,**29**,**40**] feature a memory component that maintains an adaptive internal state to capture and store task-specific information. Optimization-based algorithms [**1**, **10**, **25**] try

to achieve robust generalizability with an adaptive optimizer. The few-show learning architecture used for our high level navigation module is a variant of metric-based model that leverages a covariance estimator to accomodate potential variations in the driving scenes and hence better quantifying the distance between support and query sets for classification.

CHAPTER 3

Proposed Hierarchical Learning-based Approach

In this chapter, we begin by outlining the mathematical formulation of the task we aim to address in Chapter 3.1. We then present a detailed account of the critical modules in the proposed approach in Chapters 3.2 and 3.3. Additionally, from the perspective of the architecture's hierarchy, we provide an illustration of the higher-level and lower-level models in Chapters 3.4 and 3.5, respectively.

3.1. Problem Formulation

In our autonomous driving scenario, prior to the departure, we will populate the memory slots with images of the waypoints along a predetermined unseen route with the actions to take at the corresponding waypoints. Then the vehicle is expected to navigate through the route based only on the RGB images collected by the stereo-camera system. Since a complex set of driving actions are involved in a autonomous driving task, we break down the problem into two sub-tasks and use two different modules to solve them accordingly. The higher-level navigation module processes the environmental images and generates navigation commands such as "go straight", "turn left" or "turn right" for the subsequent module. The lower level navigation module takes the command as one of the inputs and produce a appropriate steering value based on the current environmental images to avoid obstacles and keep the vehicle on a drivable path to complete the driving actions. Fig. 3.1 shows the overall architecture of the hierarchical framework applied in this work.

The higher-level navigation module aims at identifying the waypoints with only a few sample images, so this sub-task is formulated as a few-shot classification problem, in which the $\mathcal{C}$ waypoints that we expect the vehicle to recognize are independent classes in our datatset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, in which N is the total number of samples and x_i are the samples

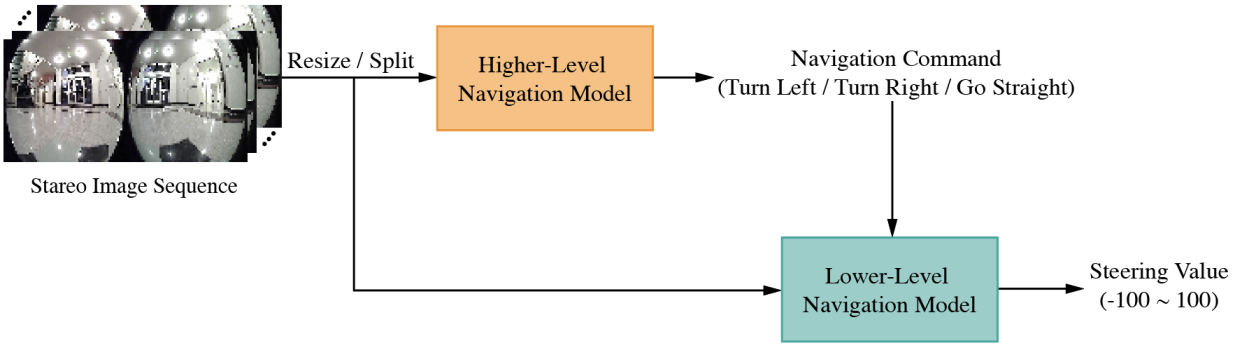


FIGURE 3.1. The Overall Architecture of the Hierarchical Framework

in the dataset and y_i are their corresponding label. Because our goal as part of the higher level control is to determine if the current scene belongs to the class of the upcoming waypoint, this task can be defined as a 2-way n-shot problem, in which the model output can be interpreted as a likelihood representing the presented query belonging to the positive class (upcoming waypoint).

For a traditional supervised learning problem, $\mathcal{D}$ is separated into two subsets: $\mathcal{D}_{train} = \{(x_i, y_i)\}_{i=1}^{N_{train}}$, and $\mathcal{D}_{test} = \{(x_i, y_i)\}_{i=N_{train}+1}^{N_{train}+N_{test}}$, the goal is to optimize parameter θ on $\mathcal{D}_{train}$ and evaluate its generalization capability on $\mathcal{D}_{test}$. Thus, the problem can be formulated as approximating function f with parameter θ as:

$$y \approx f(x;\theta), \text{ where } (x, y) \in \mathcal{D}_{train} \tag{3.1}$$

and

$$\theta = \arg\min_{\theta} \sum_{(x,y)\in\mathcal{D}_{train}} \mathcal{L}(f(x,\theta); y) \tag{3.2}$$

where $\mathcal{L}$ denotes the loss function that quantify the difference between the prediction of f and the ground truth y.

However, as a meta-learning task, because the model is trained to generalize on $\mathcal{D}_{test}$ with unseen classes, $\mathcal{C}$ is divided into $\mathcal{C}_{train}$ and $\mathcal{C}_{test}$ with no intersection. And the dataset will be composed of a group of mini-datasets [**26**] each only contains positive and negative samples denoted as (x_k^{pos}, y_k^{pos}) and (x_k^{neg}, y_k^{neg}), $k \in \mathcal{C}$ for a single class. These samples could

be used to form a support set $\mathcal{S}_k^{sup}$ or a query set $\mathcal{S}_k^{qry}$ used in an episode [**37**]. In this way, the training and testing datasets are denoted as:

$$\mathcal{D}_{train} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_{\mathcal{C}_{train}}\} \tag{3.3}$$

$$\mathcal{D}_{test} = \{\mathcal{D}_{\mathcal{C}_{train}+1}, \mathcal{D}_{\mathcal{C}_{train}+2}, \ldots \mathcal{D}_{\mathcal{C}_{train}+\mathcal{C}_{test}}\} \tag{3.4}$$

where $\mathcal{D}_k = (\mathcal{S}_k^{sup}, \mathcal{S}_k^{qry})$, $k \in \mathcal{C}$.

Having the dataset definition for the meta-learning task, the problem can be formulated as approximating function f with parameter θ as:

$$y \approx f(\mathcal{S}_k^{sup}, x; \theta), \text{ where } (x, y) \in \mathcal{S}_k^{qry} \tag{3.5}$$

and

$$\theta = \arg\min_{\theta} \sum_{\mathcal{D}_i \in \mathcal{D}_{train}} \sum_{(x,y) \in \mathcal{D}_k^{test}} \mathcal{L}(f(\mathcal{D}_k^{train}, x; \theta), y), \text{ where } k \in \mathcal{C} \tag{3.6}$$

In contrary to the higher-level navigation module that makes the decision on whether to change the current driving state, the lower-level system works on maintaining the current driving state e.g. as going straight or turning while avoiding obstacles and keeping the vehicle on a drivable route. For the low level control module, incoming camera images are presented to the lower level model which is also conditioned on the driving command to regress the steering value in order to maintain the current driving state. The condition to the lower lever control is provided by the few-shot driven high level controller. This is similar to how a human driver keeps a car driving in the center of the lane, while paying attention to the landmarks in the environment (e.g. distinct buildings, gas stations, etc.) to make navigation decisions like turn right, go straight, etc. The the lower-level module is trained through a classical supervised classification scheme, with the same separation of dataset as is used in training the higher-level model.

3.2. Feature Extractor

Three-channel RGB images are a typical kind of high-dimensional data that requires excessive computational resource to be directly processed in tasks including image classification, object detection and segmentation. Therefore, in high-level digital image processing pipelines, feature extractors are normally used as the first module to reduce the data dimensionality and convert the raw image to lower-dimensional features. These features can be divided into spectral, geometric and texture features [**19**], containing the concentrated knowledge that could represent information in the image. The output of an optimal feature could be in a low dimension but contain sufficient information for the downstream algorithm, while eliminating redundant and irrelevant information.

Popular feature extractors can be categorized into statistical methods such as principal component analysis (PCA) and Fisher discriminant analysis [**8**] and machine learning methods such as support vector machine [**36**] and neural network [**20**, **21**]. The choice of feature extractor largely depends on the specific task and the type of data being processed. In recent years, with the development of parallel computing architectures and extensive research in neural networks, CNN has been widely adopted as the feature extractor in varies machine vision tasks for its high accuracy and generalizability. In our pipeline, two CNNs (ResNet-50 [**14**] and EfficientNet [**34**]) with different parameter values are applied.

ResNet-50 is a variant of ResNet model proposed by Kaiming, et al. [**14**], the ResNet family is a landmark in the history of CNN because it solves the notorious gradients vanishing/exploding problem brought by increasing the depth of neural networks and the training of deep CNNs. Rather than increasing the number of network parameters by simply stacking convolutional layers, the structure is separated into blocks each has a few layers, and a shortcut connection across a block that acts as identity mapping. Such a short connection is able to solve the model degradation issue by pushing the block to fit a better identity mapping. Compared with its counterpart in plain architecture, ResNets are easier to optimize while adding no extra trainable parameters.

EfficientNet is known for its high efficiency and various options on the parameter scales. Rather than arbitrarily scaling models by their depth, width, or resolution, an effective compound scaling method is introduced to uniformly scale up or down the architecture with a set of fixed scaling coefficients, while achieving a balance between accuracy and efficiency. With the scaling strategy, a series of models from EfficientNet-B0 to EfficientNet-B7 with different scale are presented, and they all outperformed the state-of-the-art counterparts at the time. Fine-tuned on the EfficientNet weights pre-trained on ImageNet, the transfer learning accuracy on common datasets is also impressive. Rather than using a larger model, running inference with a more compact one with fewer parameters can result in a higher frame rate on the test platform with limited computing power. Thus, EfficientNet-B0, which is in the most compact architecture in the EfficientNet family, is chosen as the feature extractor of our lower-level navigation module.

Technically a single feature extractor can be trained with multi-task learning [**3**] techniques to serve for multiple purposes. However, in order to train the few-show learning model more efficiently, the images used to train the model are limited in a small range before the waypoints, which is only a selected portion of the recorded lap (details described in Section 4.4). The limited training data does not cover the entire range where the steering value regression model works in, thus, we utilize an additional CNN in the lower-level navigation system and train this model with a standard supervised training approach that makes use of the entire training set.

3.3. Covariance Estimation

The biggest restriction of few-shot learning tasks is the limited number of samples for each class. A naive end-to-end classification model trained with this kind of data would lead to severer overfitting. So mainstream few-shot learning frameworks aim at embedding data to a latent space and improving the classification result of encoded features. There have been approaches that achieved decent performance: Matching Networks [**37**] proposed an encoder based on bidirectional Long-Short Term Memory (LSTM) [**12**], and used neural attention

mechanism [**17**] for feature weighting to enable rapid learning. However, these methods focus solely on the inter-class knowledge instead of also considering the potential distribution variation of different classes. In order to overcome this limitation and improve the model's feature representation capability, an additional covariance module is incorporated into the model to provide an estimate of the encoding covariance for better generalization and to improve model robustness. This approach was developed in the Laboratory for AI, Robotics and Automation at UC Davis and is adopted in this work to handle potential variations of the landmark scenery e.g. lighting changes, perspective changes, presence of pedestrians and other vehicles to name a few.

Similar recognition patterns in human behavior can help explain the idea behind covariance estimation. Given a task of learning to recognize a new breed of cats with only a few pictures of them, humans are able to perform class-level feature extraction of summarizing key features including the pattern, tail length, ear shape, etc. However, in addition to such capability, humans are also capable of going beyond the provided information and visualize potential variations of the given class, such as imagining their appearances in unseen postures, their dynamic movements, and even how their offsprings possibly look like. These higher-order statistics inferences are based on the biological and physical knowledge learned from the past experiences, that could improve the generalization of the classification model to a large extent, especially in a few-shot scenario.

The covariance estimator in our architecture is expected to learn the generation of possible non-linear variations and produce the covariance encoding based on the original feature encoding. These two encodings are both involved in the subsequent distance measuring algorithm.

3.4. Higher-Level Navigation Model Architecture

At inference time of the few-shot learning model, the input data is a sequence of frames captured by our dual-camera system, and the classification is based on the mapped features of the query images and the support images in a latent space, called stamps. We propose to use

a network architecture developed earlier in the Laboratory for AI, Robotics and Automation at UC Davis, named StampNet and optimized based on it (pipeline shown in Fig. 3.2) to generate the encoding and the corresponding covariance, forming the aforementioned stamps. In our AV application, each stamp contains the information of the last 10 consecutive frames captured by the vehicle camera.

The process of producing the stamps is explained in the following:

First, to process a single frame, we adopt ResNet-50 as the feature extractor followed by a dense layer to convert the high-dimensional 3-channel images to one-dimensional feature encodings in the latent space. In order to represent the environmental features of both the left and right images in the same frame, we encode the two images separately using the the CNN, then concatenate the two feature vectors to form a joint vector of twice the length. The generated encoding of the current frame will be added to the end of a queue of the 10 most recent encodings, and the stamp of the current frame is obtained by averaging the feature embeddings in the queue.

Then, in addition to the average encoding, a trained covariance module is used to estimates the covariance encoding of the average feature encoding, which represents the covariance distribution of the class that the set of presented images belongs to in the latent space. We interpret the covariance encoding as an unbiased estimation of the diagonal covariance matrix, so it can be simplified as a 1-dimensional vector. The stamp is formed by an encoding and the associated covariance.

Finally, a distance measure is required to produce the similarity score for the given support and query data. The most straightforward measure is the Euclidean distance, however, the calculation of Euclidean distance treats each feature equally under the assumption that the features are isotropically Gaussian. For example, given data x and y, the Euclidean distance is calculated as:

$$d_E(x, y) = \sqrt{(x - y)^T (x - y)} \tag{3.7}$$

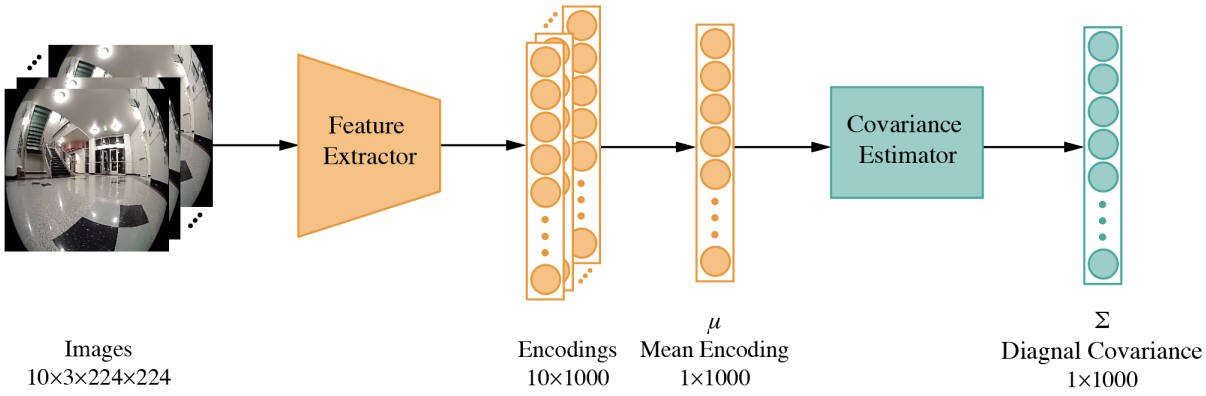


FIGURE 3.2. StampNet Pipeline

Calculating Euclidean distance is easy and efficient, but Euclidean distance does not benefit from the information about the second order statistics (covariance) which is available in our case. There are other distance measures that utilize second order statistics. As an example, Mahalanobis distance first re-scales the data by normalizing each feature to make sure they all have unit variance before calculating the Euclidean distance in the transformed space. For measuring the distance between x and y, the formula is given below:

$$d_M(x, y) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)} \tag{3.8}$$

Where Σ denotes the covariance.

Therefore, the Mahalanobis distance is scale-invariant, and is also capable of representing the correlations among the features. Having these advantages, we use Mahalanobis distance to take both the feature encoding and the covariance vector into account to quantify the similarity between the query and the support data. Given the distance, the classification result is obtained with a three-layer classification network followd by a softmax layer.

3.5. Lower-Level Navigation Model Architecture

The objective of this model is to generate a steering value to keep the vehicle in the desired driving state while maintaining it in the same high level navigation e.g. keep going

straight. The input to the low level controller is stereo images captured by the cameras along with the the naviation command from the higher-level navigation module as a condition. We split the low level driving behavior into three states: go straight, turn left and turn right, driven by a condition input provided by the high level control module.

The architecture of the lower-level-navigation model is depicted in Fig. 3.3 The stereo image is first encoded by EfficientNet-b0, to process the feature encoding and obtain the regression value, the last fully connected layer that comes with the EfficientNet is replaced by a MLP with three layers, that takes the flattened 1280 dimensional vector output of the EfficientNet as the input and generates a single output. Notably, in order to let the model respond differently to different navigation command, we transform the categorical command index into a One-Hot numerical encoding, and concatenate it with the output of the second layer of the MLP. In this way, the navigation command is not introduced until the last MLP module, so the EfficientNet structure does not need to be modified, making it possible to make use of the pre-trained weights, and the feature extractor can focus on generating the features. The MLP is followed by a sigmoid function to map the output to the range from 0 to 1, in which 0 represents the maximum steering to the left and 1 represents the maximum steering to the right. Then, a PWM digital signal with corresponding duty cycle is generated to reflect the steering value in the servo angle to control the actual steering behavior of the vehicle.

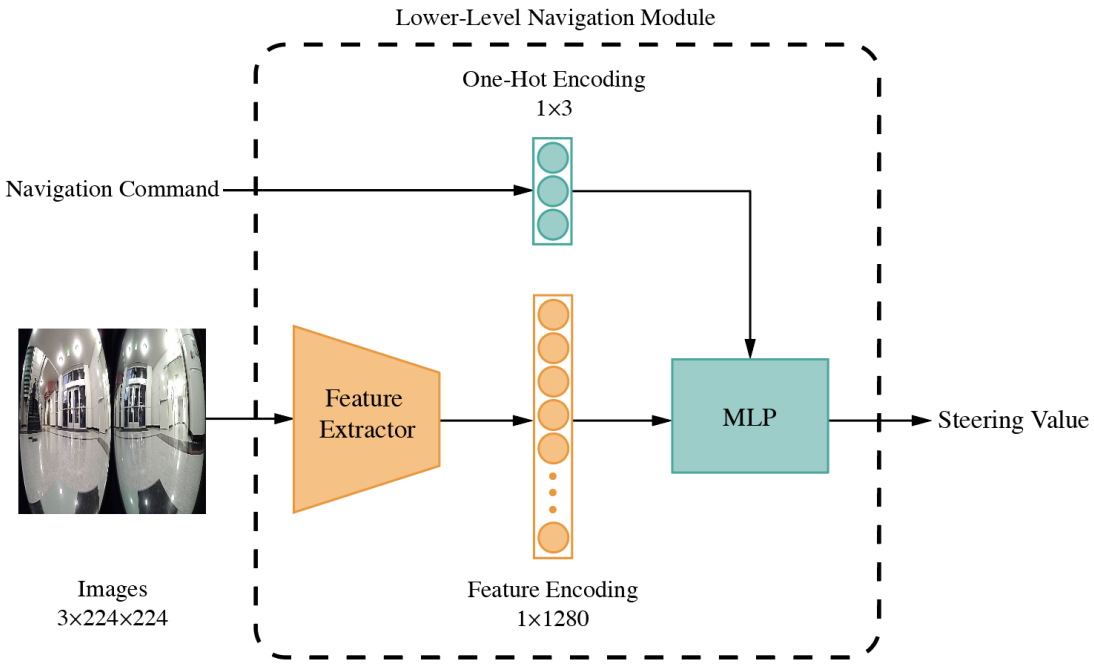


FIGURE 3.3. Lower-Level Navigation Model Architecture

CHAPTER 4

Experiment

In this section, we are going to describe the hardware platform we use for data acquisition and testing, as well as the dataset we created for this project. Then, ablation test results are presented to compare the classification performance under different settings. Finally, an online test is presented to demonstrate the effectiveness of the proposed approach.

4.1. Experimental Setup

The vehicle platform we use is a mini autonomous vehicle (miniAV) whose chasis is based on a 1/6 scale remote controlled 6x6 vehicle that hosts all the other modules. The camera system consists of two ELP 180-degree fisheye cameras that point at the left front and right front respectively in a symmetrical layout with a 120 degrees intersection angle. The computing unit is a mini PC with an Intel i7-10750H and an NVIDIA RTX 2070 graphics card with 8 GB of memory. An Arduino Mega 2560 Rev3 microcontroller board is used to process and pass the control signals from the computing unit or the remote controller to the steering actuator on the mini-AV. A rechargeable battery pack of six 3.7V Li-Fe cells connected in series is included to power the entire system for approximately an hour or operation without an external power source.

A software with user interface (shown in Fig. 4.1) is developed by our collaborator at Ford Motor Company to control the mini-AV, it supports three driving modes: manual, recording and autonomous. In manual mode, the vehicle is fully under manual control. The recording mode is designed for data acquisition, during which the throttle would be set to a preset value to make sure the mini-AV moves at a fixed speed to collect frames with uniform intervals, and the operator only needs to control the steering amplitude to finish the planned

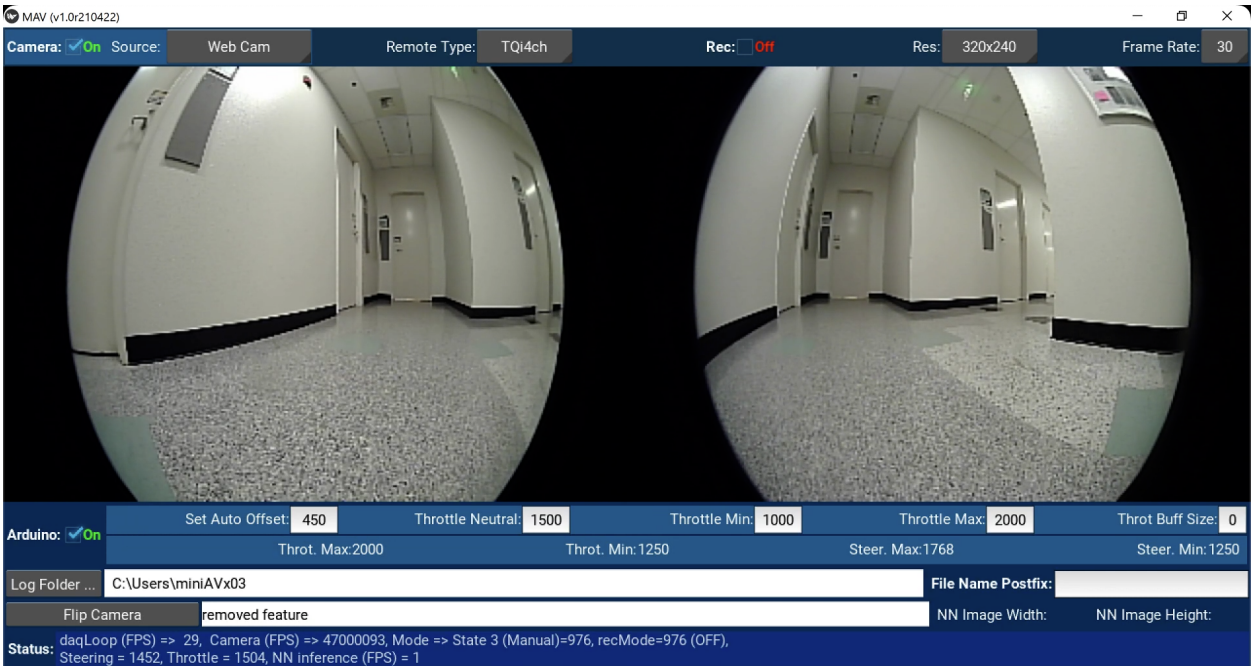


FIGURE 4.1. User Interface of the MiniAV Software

lap. In autonomous mode, the mini-AV is free of any external controls and the driving decisions will all be determined by the two models (high and low level control networks).

4.2. Dataset

The dataset consists of a total of about 268,000 images of 188 laps recorded in 18 different routes located in 12 buildings on UC Davis campus. The predetermined courses are loops with 4 turns, and multiple clockwise and counterclockwise laps along the routes are performed to ensure the number of right turns and left turns are balanced. All environments are indoor dominated by artificial lighting while a few locations have the presence of natural light hence, influenced by time and weather conditions (sunny vs cloudy, etc). Since the environment was in its natural state during data collection, some visual noise such as passing pedestrians and suddenly opened doors were also recorded, that provided natural variations across different laps.

The data is recorded at a frame rate of 25 fps, each including an image in PNG format with the resolution of 320×240 from each of the two cameras, as well as the simultaneous steering value. In order to properly store the collected large and hetergeneous data, the

FIGURE 4.2. A Original Stereo Image from the Dusl-Fisheye Camera System

file format of Hierarchical Data Format version 5 (HDF5) was selected for organizing the structured data in a single file.

To segment the recorded full laps into images for different classes, the steering value is used as the reference to identify the moment when a turn was initiated or completed, and the frames within the range of 15 frames before that point are considered the positive samples for the waypoint, and the frames prior to the positive range are considered negative.

4.3. Data Pre-Processing

Different from the picture we take with a normal camera, the fisheye lens cause severe distortion due to the coverage of a wide viewing angle, thus the valid pixels that contain environmental information are located in a disc-shaped area in the middle of the image, leading to two black zones with invalid pixels on the left and right sides of the image (see Fig. 4.2). In order to remove the black zone as much as possible, we crop out the most left 30 columns and the most right 30 columns of pixels and the valid image after cropping is in the resolution of 260×240.

For the higher-level navigation model, since each side of the images from stereo-camera system is encoded separately, the left and right images are resized from 260×240 to 224×224 to meet the required input resolution of the feature extractor. In contrast, to simplify the process and speed up the inference of the lower-level navigation model, the stereo image is

directly resized without splitting the left and right images before being fed to the feature extractor.

Then, the resized images are passed to the data augmentation function to mimic the potential image variation caused by factors including horizontal body tilt, lighting condition and camera white balance change. It is noted that this data augmentation helps the covariance network and takes place during the training as well as test time. This is different from conventional augmentation techniques which only take place during training. In order to improve the model robustness in the case when random objects appear in the query images but not in support images, or vice versa, we apply random erasing to randomly replace one or more rectangle areas in the field of view with black pixels, in order to intentionally guide the model to focus on features outside the erased areas when a portion of features are missing or covered by objects appeared occasionally. Furthermore, considering the frame rate of 30 during data collecting, the 10 images in the same support or query set are collected in around 0.33 seconds, the variation factors do not apparently change, we apply the same randomly picked augmentation parameter to the images in a same set to match this rule.

4.4. Training

The dataset is separated into training, validation and test sets without overlap. Out of the 18 recorded routes in the dataset, 12 of them with at least 4 laps for each direction are randomly selected for training, 3 of the rest are included in the validation set, and the remaining 3 courses are only used in testing. In validation and testing, the laps are selected to mimic the process and data in online tests: pick a lap to populate the memory with waypoint information, and run inference by traversing all the images from another random lap in the order they were collected. For each validation/testing courses, the process is repeated with all the possible two-lap combinations to cover all the differences between the two selected, avoiding the coincidence of using two laps that are too-similar or too-different.

The high-level navigation model is trained in a few-shot meta-learning learning scheme [**27**], in which the optimization is based on the subsampled dataset named episodes [**37**].

The key technique of training a few-shot learning is mimicking the conditions encountered at test time [**37**], therefore, in each episode, a class k is randomly picked in $\mathcal{C}_{train}$, then s samples of class k are selected from $\mathcal{D}_k$ to form the support sets $\mathcal{S}_k^{sup} = \{(x_i^k, y_i^k)\}_{i=1}^s$ and the query sets $\mathcal{S}_k^{qry} = \{(x_i^k, y_i^k)\}_{i=1}^q$ with q samples. After data pre-processing4.3, the feature extractor performs a dimensionality reduction on each $\mathcal{S}^{sup}$ to map the data to a latent space in the form of 1-dimensional encodings (vectors), denoted as $f(x_i; \theta)$. The samples in $\mathcal{S}^{sup}$ are combined by calculating the mean of their encodings:

$$\mu^{sup} = \frac{1}{s} \sum_{i=0}^{s} f(x_i; \theta) \tag{4.1}$$

Additionally, based on μ^{sup}, the covariance estimator generates the associate covariance estimation for set $\mathcal{S}^{sup}$, as $\sum^{sup} = g(\mu^{sup}; \phi)$. So far, a stamp of a support set can be constructed as $\mathcal{S}^{sup} = (\mu^{sup}, \sum^{sup})$. With multiple stamps, a better representation of class k can be combined by stacking the stamps, written as $S_k = (\mu_k, \sum_k)$.

Similarly, we can also obtain the stamps of S_k^{qry}, which can be either positive or negative stamps, from the same lap or other laps of that route in the same direction. In our practice, training strategies are applied to mimic the actual support/query sets presented in the test as much as possible. The s samples in both the support and query sets are consecutive frames, and the negative query sets are selected within the range of 150 images before the waypoints, which are challenging to the model because they are more similar to the positive samples. According to Eq. 3.8, the similarity between the stamps of the support set of class k and the query stamp x_j is quantified by calculating the Mahalanobis distance:

$$d_M(\mathcal{S}_j, \mathcal{S}_k) = \sqrt{(\mu_j - \mu_k)^T (\Sigma_k^j)^{-1} (\mu_j - \mu_k)} \tag{4.2}$$

where

$$\Sigma_k^j = \frac{\Sigma_j + \Sigma_k}{2} \tag{4.3}$$

With the equation above, if there are a total of n classes, the distance between query stamp S_j and all the stamps of all the classes can be obtained respectively. The probability of S_j belong to class k can be derived through a softmax function over negative distances:

$$p(y_j = k|x_j) = \frac{\exp\big(-d(\mathcal{S}_j, \mathcal{S}_k)\big)}{\sum_{k'=1}^{n} \exp\big(-d(\mathcal{S}_j, \mathcal{S}_{k'})\big)} \tag{4.4}$$

Note that negative log probabilities of query sets to positive classes are accumulated for the loss for each episode.

The model was trained with a graphics card of NVIDIA RTX A6000 that features 48GB of memory. To maximize the memory utilization, we used the batch size of 3 and construct the support and query size with 10 frames. The training process is composed of two stages: pre-training and fine-tuning, during which the learning rate is initialized as $1e^{-4}$ and reduced by a factor of 0.5 every 10 epochs. In pre-training, only the covariance module is trained while the parameters within the feature extractor are frozen. After pre-training for 15 epochs, the model is then fine-tuned for 40 epochs with all the parameters released. The model is trained for 16 episodes every epoch, and validated every other epochs.

4.5. Performance Metric

In our offline test, we designed a customized metric to quantify the few-shot learning model. Since the autonomous driving task is separated into sub-tasks (turn left, turn right, go straight), we segment the test laps into positive and negative sections. Each positive section consists of 15 positive frames of a waypoint and a negative section of a waypoint includes all the negative frames between the end of the last positive section and the beginning of the positive region of this waypoint(Fig. 4.3). Rather than taking every output probability as the smallest unit, we consider an entire section as a unit in our metric to better reflect if a navigation command is issued properly. A section would be counted as positive if at least one peaks happen in the section, otherwise it would be a considered as a negative recognition. Note that the threshold for identifying peaks is 0.5.

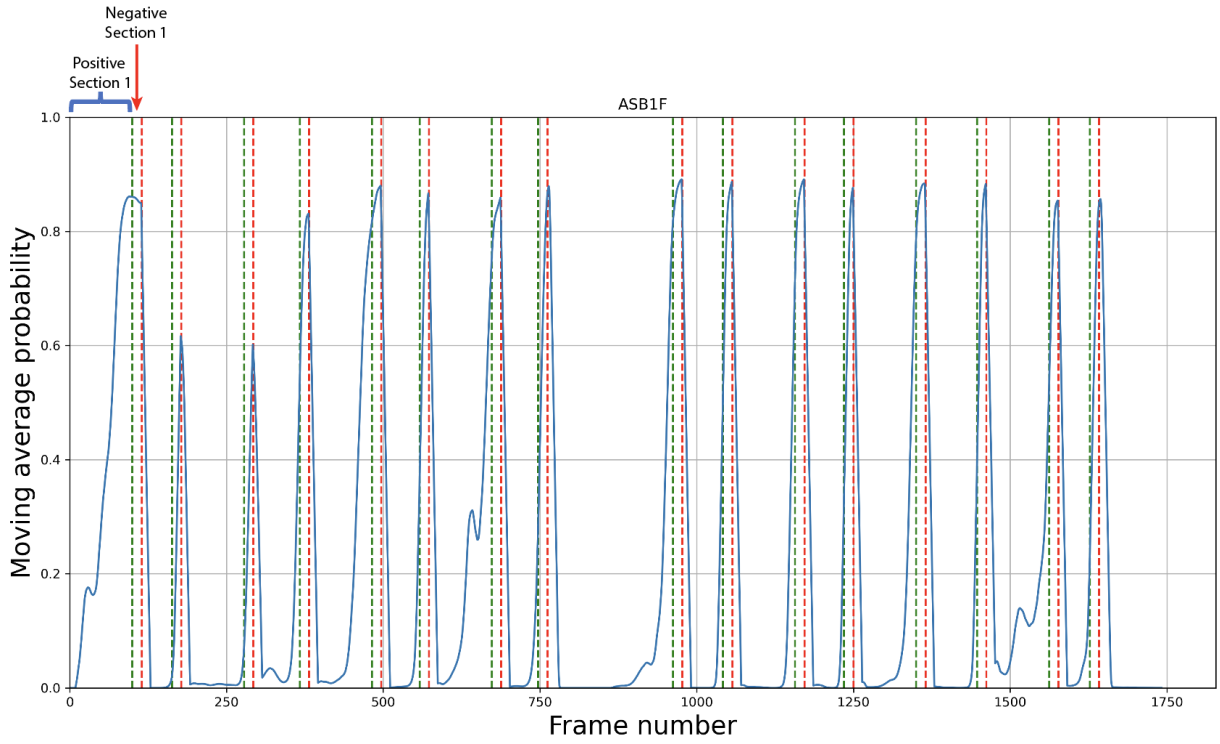


FIGURE 4.3. Moving Average Probabilities along a 16-Waypoints Route

In practice, to reduce the operation error during data collection, we expand the positive section by including 15 frames before and 9 frames after as the buffer region, thus each positive section has 39 frames. Additionally, before calculating the score, the raw output is smoothed by a moving average with window size of 10.

Having the criteria, the testing result is obtained by calculating the F1 score, which is the harmonic mean of the precision and recall:

$$F_1 = \frac{2}{recall^{-1} + precision^{-1}} = \frac{2TP}{2TP + FP + FN} \tag{4.5}$$

4.6. Ablation Study

In the ablation study, we evaluate few-shot learning models trained under different settings and compare their performance on the test set. We compared the performance of models trained based on 3 different ResNet-50 weights including: 1.Weights pre-trained with

Weights	Covariance Module	F1 Score
Unsupervised Trained	√	0.868
	-	0.857
Supervised Trained	√	0.823
	-	0.813
From scratch	√	0.667

TABLE 4.1. Ablation test result.

a classical supervised learning scheme; 2.Weights pre-trained with an unsupervised training method of SwAV (Swapping Assignments between Views) [**2**]; 3.Randomly initialized weights. To prove the superiority of the covariance module, we also evaluated the models trained with/without the covariance module in the architecture.

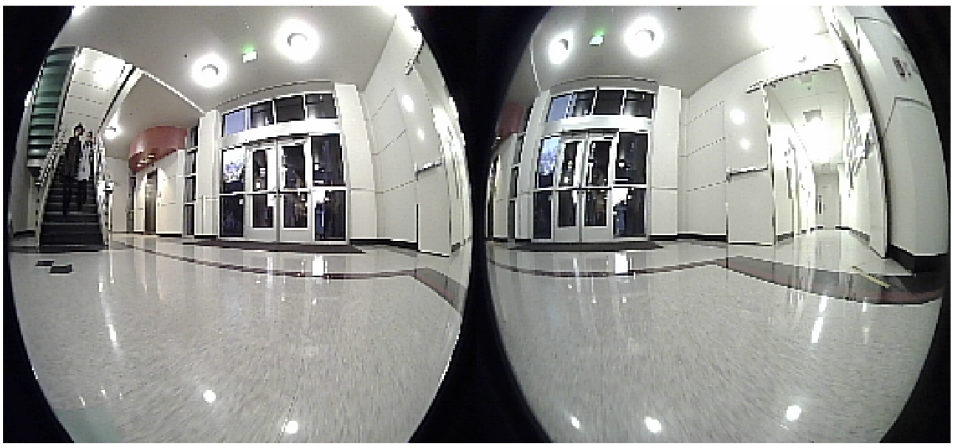
In this test, models are evaluated offline on the testing set that consists of 3 unseen courses in training. The scores are summarized in Table. 4.1. As evident from the results, compared to the case where training initiated with the model trained from scratch with randomly initialized weights, loading each of the two pre-trained weights and performing transfer-learning with our own data brought a significant improvement of at least 0.15 on F1 score. In the scope of pre-trained weights, the SwAV version trained with unsupervised learning approach has an F1 score 0.04 higher than the classical supervised trained weights. This shows that the performance of our few-shot learning model is also affected by the method with which the pre-trained weights were trained, although they are both fine-tuned by our meta-learning approach on the same new data. Supervised trained feature extractor is shown to be more capable of extracting meaningful features with limited number of training samples. With the same initialization of weights, the score of the architecture with the covariance estimation module is higher than the version without the module, which supports the fact that the proposed covariance estimator is boosting the performance of the few-shot learning model and achieves the best performance with the SwAV weights..

4.7. Online Test

In addition to the offline test, we implemented the proposed framework on the MiniAV and performed an online test in an unseen building. The tested course is also a loop with

4 turns in an indoor environment. To prepare for the requirements of the algorithm, we first manually manipulated the MiniAV along the planned course for once, and populate the memory with waypoint stamps and the corresponding navigation commands before departure. Then, in testing time, the stereo images captured by the camera system were fed to the models for real-time inference. Notably, unlike the training set in which the environmental features are similar in different laps since they were recorded continuously in a short period time, the online test was performed two hours after the collection of the memory lap that used to populate the memory. As the comparison in Fig. 4.4, the lighting condition has significantly changed. Fig. 4.4(a) shows a frame in the memory lap recorded at dusk when the sunlight was visible through the glass door, as well as causing a strong reflection on the tiles. However, the online test was done at night when all the natural light had gone, when the overall brightness of the field of view was lower, the sample frame at the same point is shown in Fig. 4.4(b). Even though the environmental features varied at test time, our approach was still capable of navigating the MiniAV to complete the course, a video of the online test can be found online at Appendix 1. The higher-level navigation model recignized the waypoints in time and produced the correct commands to the subsequent module, and the lower-level navigation model produced proper steering values to complete the desired driving action. During the sections where the MiniAV was supposed to go straight, the lower-level navigation module could output a steering value instantly to shift the car back to the middle of the hallway when the vehicle was approaching to the wall. Furthermore, the decision was not interfered by a random passerby appeared in the course.

(a) Memory Lap



(b) Testing Lap

FIGURE 4.4. Comparison of the Environment of Memory Lap and Testing Lap

CHAPTER 5

Conclusion

In this work, we demonstrate the proposed few-shot learning model is capable of navigating an autonomous vehicle by recognizing waypoints with only a few available positive images, and it can be embedded in the hierarchical framework with a regression model to achieve a fully autonomous navigation through an unseen course with minimal preparation. The online testing result shows that our approach can be deployed quickly and easily in an autonomous driving task which only relys on RGB cameras, without any complicated sensors such as GPS, LiDAR, radar, etc.

Through ablation study, the covariance estimator in our few-shot learning architecture shows satisfying performance in achieving higher scores by utilizing the model's generalization ability to propose feature variance. It is also potential in being adopted in other meta-learning approaches to tackle tasks in other domains.

Although our approach performed reasonably in the defined autonomous navigation task, it has explicit assumption that the images for the incoming waypoint are collected by the same vehicle with the same camera system, which limits the application scenarios of our approach. Furthermore, the navigation would fail if the waypoint images differ dramatically from the testing environment, such as from sunny to rainy weather, from summer to winter, etc. We also noticed that although the lower-level navigation model is capable of keeping the vehicle driving in the middle of hallways, it is clumsy in avoiding random obstacles in the way. In the future, we are planning to incorporating algorithms such as generative models to further improve the robustness of the few-shot learning model, as well as developing a more complicated mechanism to plan the path smartly during the autonomous navigation.

CHAPTER 6

Appendix A

Online Videos

(1) Title: "Online Test at the First Floor of Academic Surge at UC Davis"

Creator: Debo Shi

Description: This third-person view video shows the how the hierarchical framework runs in real time on the MiniAV to autonomously navigate it through a complete loop.

Link: https://ucdavis.box.com/s/oanhriklqc1xzhdb25mrje70e6o32kq7 (Box)

Bibliography

[1] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, *Learning to learn by gradient descent by gradient descent*, Advances in neural information processing systems, 29 (2016).

[2] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, *Unsupervised learning of visual features by contrasting cluster assignments*, Advances in neural information processing systems, 33 (2020), pp. 9912–9924.

[3] R. Caruana, *Multitask learning*, Springer, 1998.

[4] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, *Deepdriving: Learning affordance for direct perception in autonomous driving*, in Proceedings of the IEEE international conference on computer vision, 2015, pp. 2722–2730.

[5] J. Chen, Z. Wang, and M. Tomizuka, *Deep hierarchical reinforcement learning for autonomous driving with distinct behaviors*, in 2018 IEEE intelligent vehicles symposium (IV), IEEE, 2018, pp. 1239–1244.

[6] J. T. Daniel, *Autonomous Navigation of an Electric All-Terrain Vehicle Along Waypoint-Defined Trails*, PhD thesis, The University of North Carolina at Charlotte, 2022.

[7] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, *Rl* $\hat{2}$*: Fast reinforcement learning via slow reinforcement learning*, arXiv preprint arXiv:1611.02779, (2016).

[8] R. O. Duda, P. E. Hart, et al., *Pattern classification*, John Wiley & Sons, 2006.

[9] R. K. Fachri, M. Z. Romdlony, and M. R. Rosa, *Multiple waypoint navigation for mobile robot using control lyapunov-barrier function (clbf)*, in 2022 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom), IEEE, 2022, pp. 230–235.

[10] C. Finn, P. Abbeel, and S. Levine, *Model-agnostic meta-learning for fast adaptation of deep networks*, in International conference on machine learning, PMLR, 2017, pp. 1126–1135.

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial networks*, Communications of the ACM, 63 (2020), pp. 139–144.

[12] A. Graves and A. Graves, *Long short-term memory*, Supervised sequence labelling with recurrent neural networks, (2012), pp. 37–45.

[13] K. He, G. Gkioxari, P. Dollár, and R. Girshick, *Mask r-cnn*, in Proceedings of the IEEE international conference on computer vision, 2017, pp. 2961–2969.

[14] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[15] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, *Navigating occluded intersections with autonomous vehicles using deep reinforcement learning*, in 2018 IEEE international conference on robotics and automation (ICRA), IEEE, 2018, pp. 2034–2039.

[16] T. Karras, T. Aila, S. Laine, and J. Lehtinen, *Progressive growing of gans for improved quality, stability, and variation*, arXiv preprint arXiv:1710.10196, (2017).

[17] G. Koch, R. Zemel, R. Salakhutdinov, et al., *Siamese neural networks for one-shot image recognition*, in ICML deep learning workshop, vol. 2, Lille, 2015.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, Communications of the ACM, 60 (2017), pp. 84–90.

[19] M. Kunaver and J. Tasic, *Image feature extraction-an overview*, in EUROCON 2005-The International Conference on" Computer as a Tool", vol. 1, IEEE, 2005, pp. 183–186.

[20] F. Lauer, C. Y. Suen, and G. Bloch, *A trainable feature extractor for handwritten digit recognition*, Pattern Recognition, 40 (2007), pp. 1816–1824.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[22] J. Long, E. Shelhamer, and T. Darrell, *Fully convolutional networks for semantic segmentation*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 3431–3440.

[23] A. Martinez Acosta, *Intelligent autonomous inspections using deep learning and detection markers*, (2022).

[24] T. Munkhdalai and H. Yu, *Meta networks*, in International conference on machine learning, PMLR, 2017, pp. 2554–2563.

[25] A. Nichol, J. Achiam, and J. Schulman, *On first-order meta-learning algorithms*, arXiv preprint arXiv:1803.02999, (2018).

[26] A. Parnami and M. Lee, *Learning from few examples: A summary of approaches to few-shot learning*, arXiv preprint arXiv:2203.04291, (2022).

[27] S. Ravi and H. Larochelle, *Optimization as a model for few-shot learning*, in International conference on learning representations, 2017.

[28] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, in Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18, Springer, 2015, pp. 234–241.

[29] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, *Meta-learning with memory-augmented neural networks*, in International conference on machine learning, PMLR, 2016, pp. 1842–1850.

[30] A. Sauer, N. Savinov, and A. Geiger, *Conditional affordance learning for driving in urban environments*, in Conference on robot learning, PMLR, 2018, pp. 237–252.

[31] S. Shalev-Shwartz, S. Shammah, and A. Shashua, *Safe, multi-agent, reinforcement learning for autonomous driving*, arXiv preprint arXiv:1610.03295, (2016).

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

[33] J. Snell, K. Swersky, and R. Zemel, *Prototypical networks for few-shot learning*, Advances in neural information processing systems, 30 (2017).

[34] M. Tan and Q. Le, *Efficientnet: Rethinking model scaling for convolutional neural networks*, in International conference on machine learning, PMLR, 2019, pp. 6105–6114.

[35] N. D. Van, M. Sualeh, D. Kim, and G.-W. Kim, *A hierarchical control system for autonomous driving towards urban challenges*, Applied Sciences, 10 (2020), p. 3543.

[36] V. Vapnik, *The nature of statistical learning theory*, Springer science & business media, 1999.

[37] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al., *Matching networks for one shot learning*, Advances in neural information processing systems, 29 (2016).

[38] J. Wang, X. Ruan, and J. Huang, *Hdpp: High-dimensional dynamic path planning based on multi-scale positioning and waypoint refinement*, Applied Sciences, 12 (2022), p. 4695.

[39] J. Wang, Y. Wang, D. Zhang, Y. Yang, and R. Xiong, *Learning hierarchical behavior and motion planning for autonomous driving*, in 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2020, pp. 2235–2242.

[40] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, *Learning to reinforcement learn*, arXiv preprint arXiv:1611.05763, (2016).

[41] S. Yang, S. Kang, M. Kim, and D. Kim, *Human-aware waypoint planner for mobile robot in indoor environments*, in 2022 Sixth IEEE International Conference on Robotic Computing (IRC), IEEE, 2022, pp. 287–291.