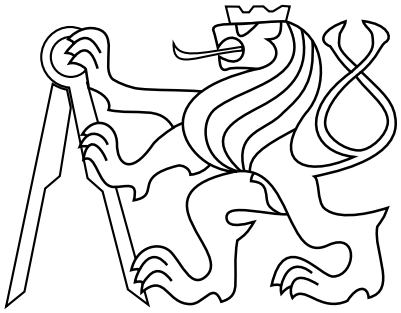
Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Cybernetics
Multi-robot Systems



Implementation of a Neural Network for Autonomous Trail Following

Bachelor's Thesis

Yevhenii Kubov

Prague, May 2022

Study programme: Cybernetics and Robotics

Supervisor: Ing. Matouš Vrba

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kubov Yevhenii** Personal ID number: **492322**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Implementation of a Neural Network for Autonomous Trail Following

Bachelor’s thesis title in Czech:

Implementace neuronové sítě pro autonomní sledování cesty

Guidelines:

Implement a neural network for autonomous trail following using data from RGB cameras placed onboard a flying micro aerial vehicle (MAV). The network should output the direction of the trail in the image to be used for navigation of the MAV so that it may follow the trail. The implementation should be able to run on the onboard computer of the MAV in ROS.
Evaluate performance of the neural network using standard evaluation metrics.
Evaluate robustness of the resulting navigation system in realistic simulations. The task is motivated by our research in swarm robotics (see http://mrs.felk.cvut.cz/research/swarm-robotics) and by search & rescue operations.

Bibliography / sources:

[1] A. Giusti et al., "A Machine Learning Approach to the Visual Perception of Forest Trails for Mobile Robots," ICRA, 2016.
[2] Seungho Back, Gangik Cho, Jinwoo Oh, Xuan-Toa Tran and Hyondong Oh , "Autonomous UAV Trail Navigation with Obstacle Avoidance Using Deep Neural Networks," JIRS, 2020.
[3] N. Smolyanskiy, A. Kamenev, J. Smith and S. Birchfield, "Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness," IROS, 2017.
[4] Bruna G. Maciel-Pearson, Patrice Carbonneau, Toby P. Breckon, "Extending Deep Neural Network Trail Navigation for Unmanned Aerial Vehicle Operation Within the Forest Canopy," TAROS, 2018.

Name and workplace of bachelor’s thesis supervisor:

Ing. Matouš Vrba Department of Cybernetics FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **06.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. Matouš Vrba	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

. Date of assignment receipt

Student’s signature

Acknowledgments

Firstly, I would like to express my gratitude to my supervisor Ing. Matouš Vrba for his great support and providing all the necessary information during writing this thesis. I'm grateful to the MRS team for guiding me during the experiments.

I would also like to thank my parents, who made it possible for me to study at CTU in Prague and for their support.

Finally, I would like to express my respect and deep gratitude to my friends, relatives and other people who now, no matter what, defend their home and democracy in Ukraine.

Author statement for undergraduate thesis

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date ....

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne ....

Abstract

The problem of autonomously following a trail by a robot based on images from an onboard monocular camera is tackled in this thesis. A robotic system that solves the task of flying through a forest along a man-made dirt trail is presented. It is accomplished by using a classification deep convolutional neural network for determining in which direction relative to the trail is the camera pointing. The output of this classifier is then used to command the robot to follow the trail. It was implemented to run online onboard an MRS multi-rotor micro aerial vehicle. Part of the implementation is also an algorithm for path planner trajectory generation. Performance and robustness was tested in simulations, followed by real-world experiments. The implemented system showed good practical results and can be used as a starting point for more complex navigation and surveillance applications.

Keywords Unmanned Aerial Vehicles, Robotic Perception, Deep Learning, Convolutional Neural Networks

Abstrakt

Tato práce je zaměřená na problematiku sledování lesní cesty pomocí obrázku z monokulární kamery, připevněné na bezpilotní helikoptéře nebo pozemním vozidle. Je představen systém, řešící úlohu navigace podél stezky v lese. Toho bylo dosaženo s využitím klasifikační hluboké konvoluční neuronové sítě pro určení směru natočení helikoptéry vzhledem k cestě. Systém byl implementován s minimálním zpožděním, aby mohl být zapojen ve zpětné vazbě s plánováním trajektorie helikoptéry v rámci MRS UAV systému. Součastí implementace je algoritmus na generování bodů trajektorie pro plánovač. Výkon a robustnost byly otestovány v simulaci a následně během experimentů v reálném světe. Implementovaný systém prokázal dobré praktické výsledky a může být použit jako výchozí bod pro komplexnější navigační a průzkumné aplikace.

Klíčová slova Bezpilotní Prostředky, Robotické Vnímání, Hluboké Učení, Konvoluční Neuronové Sítě

Abbreviations

FOV Field of View

GPS Global Positioning System

LiDAR Light Detection and Ranging

MAV Micro Aerial Vehicle

MRS Multi-robot Systems Group

ROS Robot Operating System

UAV Unmanned Aerial Vehicle

RGB Red Green Blue additive colour model

USB Universal Serial Bus

IR Infrared

CNN Convolutional Neural Network

SAR Search and Rescue

GPU Graphics Processing Unit

API Application Programming Interface

Contents

Chapter 1

Introduction

1.1 Unmanned Aerial Vehicles

Flying robots, also called UAVs have been used since the first half of the 20th century [26]. They were still controlled by a human operator, but allowed for dangerous tasks to be performed remotely, like aerial target practices. Originally designed for military purposes, they evolved into a vast industry. Since that time, electronic hardware has become much more compact, power-efficient, cheap and advanced. Also, aircraft designs have strongly evolved after decades of research, battery power improved, and software got more advanced. Nowadays, all these factors allow for a wide usage of compact and relatively inexpensive vehicles in many industries and research.

UAVs can be equipped with different payloads and sensors, depending on their task (Fig. 1.1). For example, thermal cameras are used for border control [4], [12], [32], detecting animals in wildlife [19], [20], or forest fires [15], [31]. LiDARs can be used for 3D mapping of a designated area [5], and for navigation in an obstructed environment. Digital cameras are utilised for crops analysis [9], [11], area mapping [23], surveillance [32], monitoring of different constructions or power lines [16].

Figure 1.1: MRS MAV equipped with firefighting and thermal imaging modules.

UAVs are also important for Search and Rescue (SAR) missions. Benefits of using them in such operations are portability, fast deployment time, and generally only minimal qualification required to operate them, thanks to flight computers with advanced software [25], [30]. They can be used instead of human teams, reducing the risk for their life or health in dangerous environments. UAVs may be also cheaper than alternatives, especially manned aircraft, and faster than rescue teams or ground vehicles. They are usually not only less

expensive, but also provide data logging from all sensors, including GPS and associated image data from high resolution cameras. This data can be used even for later review. For SAR missions, two types of UAVs are typically used: fixed-wing aircraft and multi-rotor helicopters. The first ones provide higher speed and longer flight time, but are less agile and take longer time to take off and land the vehicle. Multi-rotors have the ability to hover, fly close to the ground, in buildings, caves and other complex terrains. In this thesis, a multi-rotor MAV will be used as an airframe.

1.2 Convolutional neural networks

Neural networks (also called artificial neural networks) are algorithms, whose design was inspired by biological neural networks in human and animal brains. Their structure consists of nodes called neurons, and connections between them. Every connection has its weight. Neurons are grouped into layers, each layer has its unique number of them. Neurons which receive a value (stimuli) on input from outside of the network are input neurons. Those which receive values from other neurons, process this data, and output the result to other neurons are hidden neurons. Those which output the result outside the network, are called output neurons. The mentioned weights of connections between the neurons can be adjusted through a learning process. During learning, a network learns to identify needed templates in the input and produce a correct decision according to the task.

Convolutional neural networks have been experiencing tremendous growth during the last 10 years, allowing for many previously unsolved problems to be tackled [1]. They are used in military, healthcare, aerospace, social media, science and other applications. This approach allowed for faster and more accurate analysis of many diseases even on early stages [10], using measurements performed on the patient, which are then fed to a neural network. In aerospace and automotive engineering, neural networks are often used as component fault detectors and for improved guidance systems [6]. In electronics manufacturing and computer science, their capabilities help to expose failures when producing the chips, synthesise voice, compress data and solve many other tasks [7], [18]. In military applications, they help to identify hostile objects or enemies [22].

There are different types of neural networks. Several examples are presented in this chapter.

1.2.1 Segmentation networks

Among the most popular types are segmentation neural networks. Their goal is to divide an image into multiple segments. In such architecture, each pixel refers to some class or object type. This type is often used for biomedical applications. A good example is the U-Net architecture, frequently used in light microscopy [21].

1.2.2 Recurrent networks

This type of networks is used for problems, such as language translation and speech recognition. They are designed to handle sequential data on input (Fig. 1.2).

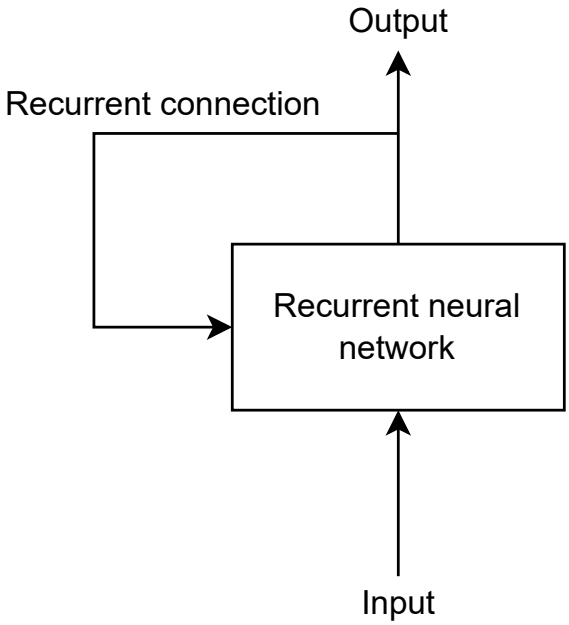


Figure 1.2: Example of a recurrent network architecture.

1.2.3 Generative adversarial networks

GAN networks consist of two neural networks, that contest with each other. Each network's gain is another network's loss. For example, in the problem of generating realistic-looking images, one network called the discriminator identifies how much the input image is "realistic", while the other (generator) generates this input and adjusts it to "fool" the discriminator.

1.2.4 Classification networks

One of the most popular are classification neural networks. In such use-case given an input image, a neural network must identify to which class does the image belong. A good example is the Alexnet architecture for classification on 1000 classes subset from the ImageNet dataset [28]. Back in 2012, it won the ImageNet large-scale visual recognition challenge. This type of a neural network is used in this thesis. However, a more modern and suitable architecture from [17] is employed.

1.3 Trail following

In this thesis, the problem of trail following using image from monocular camera onboard a multi-rotor MAV is tackled, including the implementation of a working algorithm, solving the task. Following the man-made forest path is natural for humans, because it is usually the most efficient way to get through this complex terrain. Such policy, in most cases, minimises the travel time and possible injury to a person (Fig. 1.3). The same applies to robots. Human paths are freely passable, unlike random trajectories in a forest, and it is a reason to stay on them.

Figure 1.3: A random path in a forest is generally challenging to pass.

Trail following is an important task for autonomous navigation of robots. Suggested use-cases are search and rescue missions, efficient navigation through forests and mapping of the area. Motivation for this task is a situation when there is no opportunity to communicate with and control the vehicle manually or when an autonomous mission is highly preferred. The goal is to allow for a quadcopter or an unmanned ground vehicle (UGV) to navigate through a forest using computer vision techniques. Having the image from the onboard camera, the vehicle should determine in which direction to travel when flying through a forest, utilise the trail. It must strictly follow the human path.

Algorithms solving related problems like lane-following, lane-departure and lane-assist for cars on public roads, were introduced in 1990's [33] and are commonly used in personal vehicles since the early 2000's [3]. But there is a clear distinction between the lane on a road and a forest trail. In the first case, the lane is marked with contrast symbols and lines, making the task solvable by simple segmentation algorithms, based on in-image contrast and colour variance, image saliency [27]. Forest trail images provide smaller amount of distinguishable features (Fig. 1.4) and it may be challenging even for humans to determine the direction of travel [17].

(a) Image of a trail from the dataset [17].

(b) Image of a road taken by myself.

Figure 1.4: The difference between a forest trail on Fig. 1.4a and a road on Fig. 1.4b.

One of the first effectively solved by neural networks but still topical problems is classification. Neural networks are able to learn and then identify features corresponding to predefined classes and combinations of those features [28]. This makes it possible to effectively classify which class an image belongs to.

Treating the trail following task as a classification problem is a different approach that tackles it. For this approach, classes like "Left", "Right" and "Straight" can be introduced [17] (Fig. 1.5). It allows to estimate the current direction of the vehicle, given the probabilities of these classes and to plan the trajectory of the MAV accordingly.

(a) Class "left".

(b) Class "right".

(c) Class "straight".

Figure 1.5: Example images of different classes.

This thesis focuses on implementation of the trail-following algorithm using a classification convolutional neural network. The implementation should run online in the Gazebo simulation environment as well as onboard an MAV in a real-world deployment. Therefore, delay of the algorithm must be sufficiently small. The task is to design an algorithm that autonomously provides an MAV with a relatively safe direction and speed of movement. The MAV is equipped with a PX4 flight computer, an Intel NUC companion computer, and an Intel RealSense camera. The environment may contain obstacles, but it is assumed that the trail is obstacle-free.

Implemented neural network can be used as an entry point for a more sophisticated surveillance and navigation algorithms. For more complex environments a possible enhancement is to use it alongside with an obstacle avoidance algorithms and lateral correction [2], [8], [14].

Chapter 2

Methodology

In this thesis, an architecture suggested by Giusti, Guzzi, Ciresan, *et al.* [17] will be used. It consists of 4 convolutional layers, each followed by a hyperbolic tangent activation function and max-pooling layer, and then a fully connected layer with 200 hidden neurons. Network processes images from a camera, attached to a vehicle. Input layer is formed by $3 \times 101 \times 101$ neurons. Therefore, the input RGB rectangular image must be anisotropically resized to a size 101×101 pixels (square) to be fed directly to the network. After going through all the hidden layers and the softmax output layer, it produces 3 probabilities of each class, which sum to 1. Based on these probabilities, it is possible to determine at which direction is the camera most probably pointed. Given the fact that in the dataset, the cameras for "left" and "right" classes were pointed 30° from the centre, interpolation is also possible based on these probabilities.

2.1 Convolutional Neural Network

2.1.1 Convolutions

Convolution is a fundamental mathematical operation used in a wide range of image processing techniques. In the context of Convolutional Neural Networks, convolution of an input matrix I with a so called "kernel" matrix K is applied to obtain an output matrix O. The kernel is typically smaller and is applied to submatrices of I to extract features corresponding to K in different regions of I. The convolutional layer of a CNN typically also contains a bias term w_0. The kernel matrix K and the bias w_0 are parameters that are learned during the training phase using the backpropagation algorithm, described in section 2.2. Let us define

$$I = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ x_{r1} & x_{r2} & x_{r3} & \dots & x_{rn} \end{bmatrix}, \tag{2.1}$$

$$K = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}. \tag{2.2}$$

3×3 kernel size was chosen for illustration, but different sizes can be used.

Convolution is performed by "stamping" a kernel onto the input data, starting from the upper left angle and thus creating a linear combination of input array members and kernel weights. The result of the first application of the convolution kernel will be

$$\begin{aligned} O(1,1) = x_{11} \cdot w_{11} + x_{12} \cdot w_{12} + x_{13} \cdot w_{13} + \\ x_{21} \cdot w_{21} + x_{22} \cdot w_{22} + x_{23} \cdot w_{23} + \\ x_{31} \cdot w_{31} + x_{32} \cdot w_{32} + x_{33} \cdot w_{33} + w_0 \,. \end{aligned} \tag{2.3}$$

And the equation for the convolutional kernel stamp starting on i-th row and j-th column of the input image (where the convolution is defined) is

$$O(i,j) = w_0 + \sum_{k=1}^{m} \sum_{l=1}^{n} I_{i+k-1,j+l-1} \cdot K_{k,l} \,, \tag{2.4}$$

where K has m rows and n columns, i runs from 1 to $M - m + 1$, j runs from 1 to $N - n + 1$. To produce the output array, the filter must slide through the whole input image.

2.1.2 Hyperbolic tangent

For a neural network to not act as a linear classifier, a nonlinearity should be introduced in its hidden layers [13]. It allows the network to solve more complex tasks and increase its performance. It is also a nature-inspired approach. Nonlinearity is introduced using nonlinear activation layers, added after each convolutional layer. The most popular activation functions are Rectified Linear Unit (ReLU), Leaky ReLU, Sigmoid and Hyperbolic Tangent. The last one is used in this thesis. Hyperbolic Tangent has a very similar shape to the Sigmoid and maps the output only to the range of [-1, 1] (Fig. 2.1). It is calculated using the following equation:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \,, \tag{2.5}$$

where x is the real input value obtained after the convolution stamp. Activation function is applied to every member of O from the equation (2.4) to form the input I for the next layer.

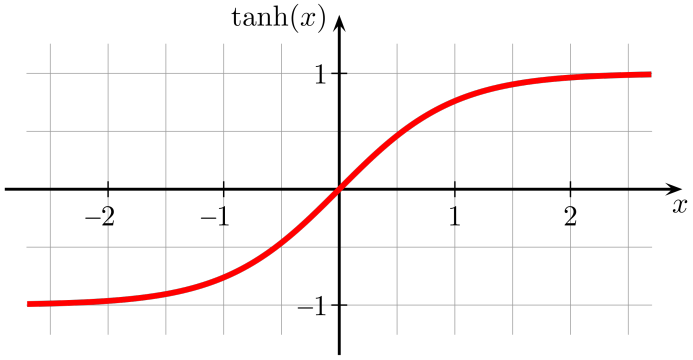


Figure 2.1: A Hyperbolic Tangent function.
Author: Geek3, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=4198479

2.1.3 Max-pooling

Max-pooling is an operation applied to some part of the input data array taking only the maximum value of this area. It is typically used in the form of a rectangular filter, which

slides through the whole image. It produces only one output value from each filter-sized input area. Thus, it can be used for downsampling the image, taking only the most significant values to the output. In this way, the learning process of the neural network is sped up because the amount of learnable weights is decreased. Also, better resistance to distortions and affine transformations is obtained [24].

2.1.4 Softmax

The softmax function is widely used in neural network architectures as the last layer. It has the same amount of outputs as inputs. The softmax function may have any real values on input, including positive, negative, zero, but its output values are always in the range $[0, 1]$ and they always sum to 1. These properties allow the output to be interpreted as a probability distribution of the corresponding classes. This layer normalises the output, which is from R^n to a probability distribution.

The softmax function is defined as:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}, \tag{2.6}$$

where z_i are elements of the real input vector, $\sigma(\vec{z})$ is the output vector, n is the number of elements.

2.2 Backward propagation of error

Backward propagation is a way to calculate the gradient $\frac{\partial J}{\partial \mathbf{w}}$ of a loss function J with respect to the vector of weights $\mathbf{w}$. The calculated gradient has the same length as the weights vector. It represents how much each weight affects and contributes to the value of the loss function. This knowledge is used to change the weights in a way that minimises the loss.

The first part of backpropagation is a forward pass. Given the input data and the weights, output of the neural network is calculated and compared with ground truth through the loss function. Then, a backward pass starts. Partial derivatives are calculated sequentially through each layer starting from the last one and after multiplication give the total gradient $\frac{\partial J}{\partial \mathbf{w}}$.

The architecture used in this thesis consists of convolutional layers, max-pooling layers, a hyperbolic tangent activation function and fully connected layers (Fig. 2.2). Max-pooling chooses one input with the maximum value and feeds it directly to the output, other inputs are ignored. It doesn't have learnable parameters affecting the gradient. During backpropagation, the gradient is only propagated to the maximal input, the remaining non-maximal inputs have a zero gradient. The derivative of the hyperbolic tangent is $\frac{d\tanh(x)}{dx} = 1 - \tanh(x)^2$. In the convolutional layer, back propagation is the convolution of the input feature map with the upstream gradient (Fig. 2.3). In the figure, $\gamma(y)$ is any function, for example the activation

function. In this case, backpropagation through the convolution will be:

$$\begin{aligned}
\frac{\partial \gamma}{\partial w_{11}} &= \frac{\partial \gamma}{\partial y_{11}} x_{11} + \frac{\partial \gamma}{\partial y_{12}} x_{12} + \frac{\partial \gamma}{\partial y_{21}} x_{21} + \frac{\partial \gamma}{\partial y_{22}} x_{22}\,, \\
\frac{\partial \gamma}{\partial w_{12}} &= \frac{\partial \gamma}{\partial y_{11}} x_{12} + \frac{\partial \gamma}{\partial y_{12}} x_{13} + \frac{\partial \gamma}{\partial y_{21}} x_{22} + \frac{\partial \gamma}{\partial y_{22}} x_{23}\,, \\
\frac{\partial \gamma}{\partial w_{21}} &= \frac{\partial \gamma}{\partial y_{11}} x_{21} + \frac{\partial \gamma}{\partial y_{12}} x_{22} + \frac{\partial \gamma}{\partial y_{21}} x_{31} + \frac{\partial \gamma}{\partial y_{22}} x_{32}\,, \\
\frac{\partial \gamma}{\partial w_{22}} &= \frac{\partial \gamma}{\partial y_{11}} x_{22} + \frac{\partial \gamma}{\partial y_{12}} x_{23} + \frac{\partial \gamma}{\partial y_{21}} x_{32} + \frac{\partial \gamma}{\partial y_{22}} x_{33}\,.
\end{aligned} \tag{2.7}$$

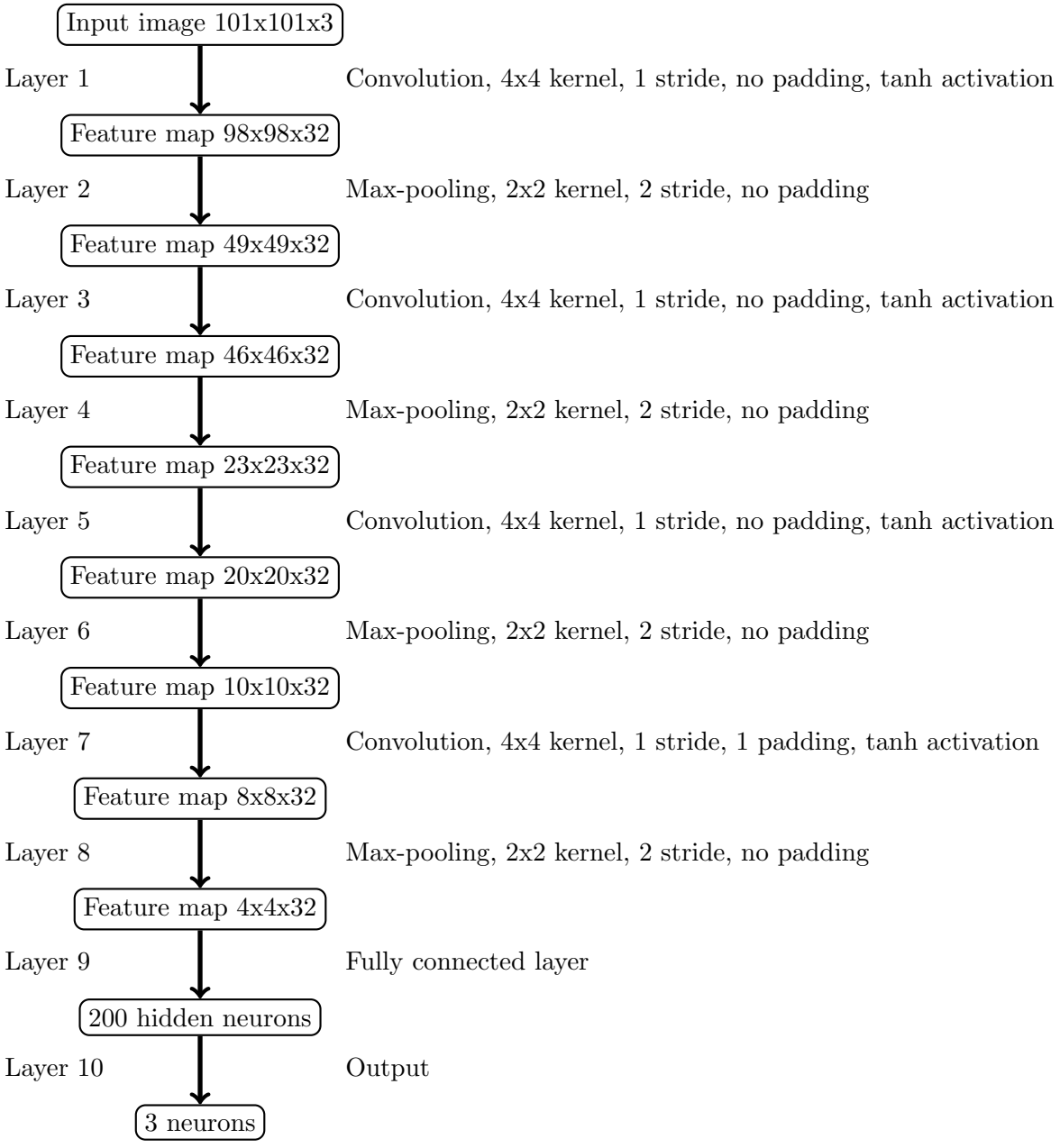


Figure 2.2: The employed neural network architecture [17].

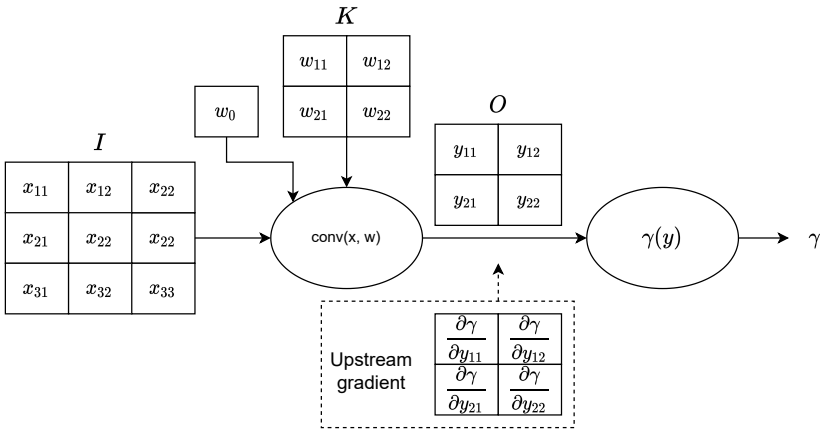


Figure 2.3: Upstream gradient in convolutional layer.

2.3 Loss function

To train the neural network, it is necessary to define a suitable loss function. It estimates the penalty of the difference between a ground truth and a prediction of the neural network. The loss function outputs one real value based on this data. Then, using the back propagation, the penalty gets minimised. In this thesis, I use a cross-entropy loss criterion. Its equation is

$$Loss = -\sum_{x=0}^{n} p(x) \cdot \log q(x)\,, \tag{2.8}$$

where n is the number of classes, $p(x)$ is the desired probability of the class (ground truth), $q(x)$ is the prediction from the neural network.

Chapter 3

Implementation

3.1 System hardware

Main hardware elements of the MAV platform used in the experiments are shown on the Fig. 3.1.

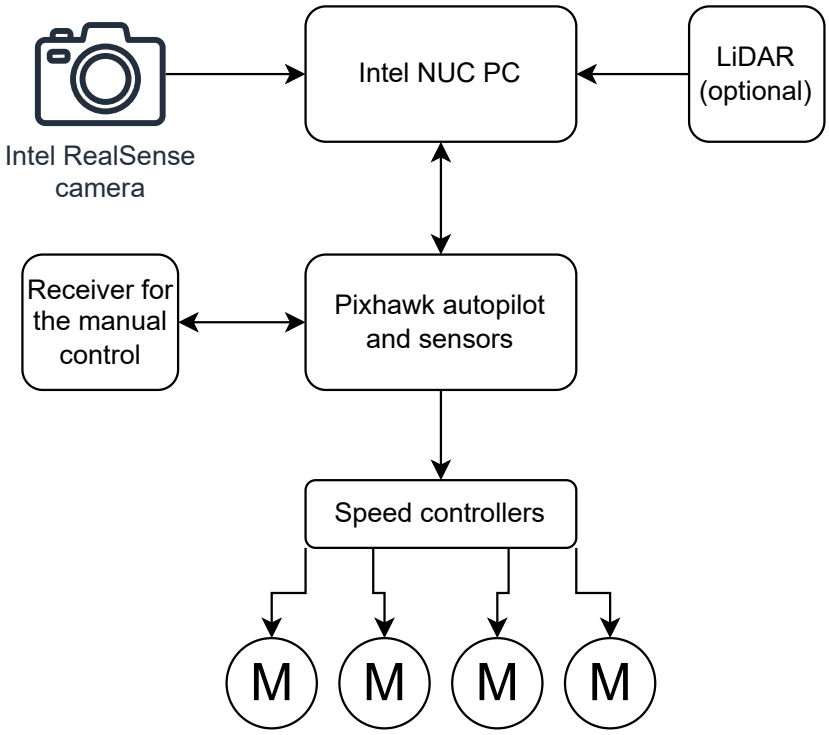


Figure 3.1: Scheme of the employed system.

3.1.1 Pixhawk flight controller

Pixhawk is a low-cost advanced flight computer with open-source hardware. There are different variations of form factors, featuring different amount of input/output ports. Pixhawk is very flexible in terms of attachable peripherals, stable and well-tested. Most essential sensors like accelerometers, gyro, digital compass (magnetometer) and barometer are already part of the main board. MRS vehicles have these boards flashed with open-source PX4 autopilot software. Features like advanced regulators, estimators, interface for controlling the MAV and others are already implemented in this software, usually only minor tweaking is needed. Thus,

an abstraction from the MAV hardware is created, allowing the vehicle to be controlled using high-level commands.

3.1.2 Intel NUC companion computer

NUC is a compact high-performance computer, capable of running demanding AI and machine learning software. It is possible to install the whole MRS system on it including ROS software to command trajectories, speed and other parameters to a flight controller and act as a main high-level computational unit. ROS runs inside the Ubuntu operating system on this computer, so other software can be used simultaneously. Different peripherals can be connected to the computer via USB. For this thesis, a RealSense camera is connected, from which my algorithm (written in the Python programming language) receives images for processing.

3.1.3 Intel RealSense D435 camera

The D435 is a powerful camera capable of taking normal RGB pictures as well as depth images. It has a wide FOV, which is perfect for robotic applications. Also, its stereo imagers feature global shutter, which is important in low-light conditions or during fast movements. The camera consists of 4 modules: a right imager, an IR projector, a left imager and an RGB module. However, for this thesis only the RGB module will be used. Its sensor has a resolution of 2 MP and produces 1920×1080 images at the frame rate of 30 frames per second.

3.2 Dataset

For this task, the dataset provided by the authors of the CNN architecture was chosen [17], but any similar dataset can be used. Requirement is that every image must be labeled to one of the three classes. The used dataset was acquired by a hiker, wearing three head-mounted cameras: one pointing straight ahead, one pointing 30 degrees left, one pointing 30 degrees right. In the code, there will be classes LEFT (for the camera pointed 30 degrees left), STRAIGHT (forward camera), and RIGHT (for the camera pointed 30 degrees right). Dataset is divided in such a way that approximately 75% of it is used for training, and 25% for validation. So, the neural network estimates the current direction relative to the trail, and based on this knowledge, further decision can be made.

The dataset was provided by the authors only as a set of photos. For the preparation of these photos into usable form, I implemented a dedicated program in the Python programming language. The program iterates through all images from the dataset and performs transformations so that the images correspond to the required format. In this case, they are resized to 101 by 101 pixels and saved in a suitable format for the training program.

3.3 Neural network code

The implementation of the neural network was not provided by the authors of the architecture. Therefore, it was created by me. I have chosen the Python programming language for this task. It has a lot of available powerful frameworks, especially for machine learning and neural networks, which are internally implemented in faster low-level languages like C++.

The PyTorch framework was chosen. It is a powerful open-source machine learning framework with a Python API. PyTorch is capable of running on a GPU to accelerate tensor computing, however, a CUDA-capable Nvidia GPU must be used. In this thesis, it was used on the MAV's onboard PC without the hardware acceleration.

The neural network is trained for 90 epochs, with a batch size of 512, but even a larger size could be considered if the GPU has enough memory. The Adam optimizer is used, with an initial learning rate of 0.005. A scheduler is set, for reducing the learning rate by 5% after each epoch. The criterion for training is the cross-entropy loss, as defined in Sec. 2.3.

During the training process, the accuracy on the validation dataset reached the value of 90% (Fig. 3.2). The process was stopped after 90 epochs because the validation loss has converged (Fig. 3.3). Further training will cause overfitting and worse performance.

3.3.1 Training algorithm

After the initialization of the neural network model, optimizer, scheduler, and criterion, the program enters a 90-epoch loop. Then, the iterative training begins. One step of the optimisation consists of the forward pass followed by the backpropagation and an update of the weights based on the calculated gradient. After the optimisation step, the accuracy and loss function metrics are evaluated on the validation dataset (see Fig. 3.2 and Fig. 3.3) and the learning rate is updated by the scheduler. In the end, after a sufficient amount of iterations, the weights of the neural network are saved for further usage.

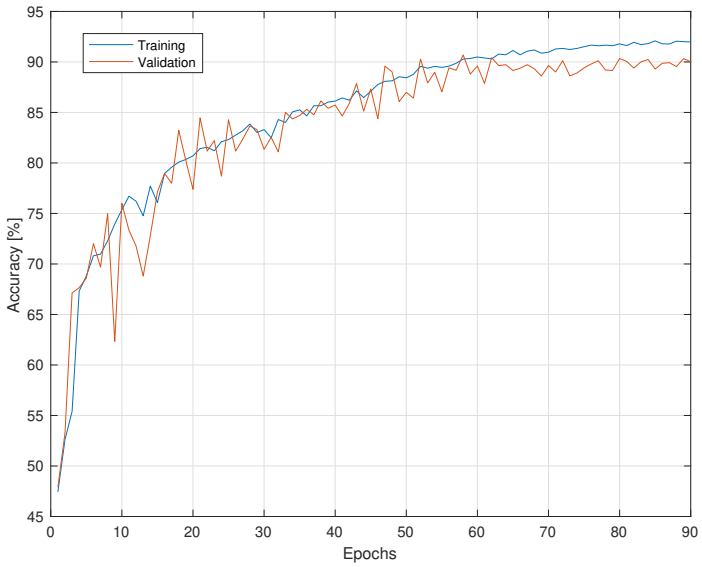


Figure 3.2: Accuracy during the training process.

3.4 Navigation algorithm

Two different approaches were tested in this thesis. The first is angular and forward velocity generation, the second is path generation in combination with a collision free pathfinding algorithm.

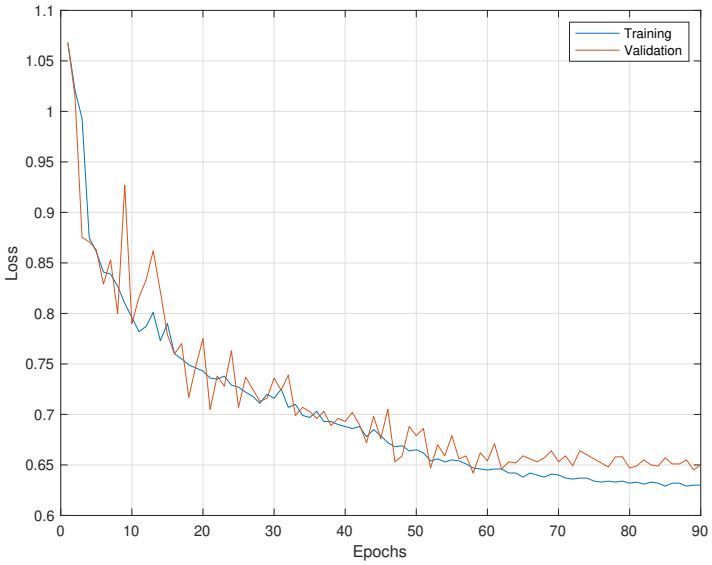


Figure 3.3: Loss during the training process.

3.4.1 Velocity generation

Generating velocities is the most intuitive way to utilise the neural network for trail following. In this case, input to the vehicle are only two values: angular velocity for heading correction and forward velocity for moving along the path when it is safe according to the neural network outputs. Angular velocity ω can be calculated using a simple formula

$$\omega = (\mathrm{p(RIGHT)} - \mathrm{p(LEFT)}) \cdot \omega_{max} \,, \tag{3.1}$$

where p(RIGHT) is the probability of the current direction being "right", p(LEFT) is the probability of the current direction being "left", ω_{max} is the maximum desired angular velocity. Forward speed v_x is calculated according to a formula:

$$v_x = \mathrm{p(STRAIGHT)} \cdot v_{max} \,, \tag{3.2}$$

where p(STRAIGHT) is the probability of the current direction being "straight" and v_{max} is the maximum desired longtitudinal velocity.

3.4.2 Path generation

MRS ROS-based system allows to control a UAV using paths. They are represented as a sequence of geometric poses, which a vehicle should take. The implemented neural network does not allow to predict a future direction of travel and gives an output only for the current position. Therefore, it is possible to predict only one point of the path on every image pass through the neural network. This point can be immediately sent to a path planner. Such approach has a big advantage: the algorithm can be run simultaneously with obstacle avoidance and other features. Path planner can then decide whether suggested direction is safe or other maneuver must be taken to evade a collision. Each point consists of heading and x,

y, z coordinates. These points are represented relative to the MAV's current position with tilt-correction (xy plane is always parallel to the ground and z is always perpendicular).

Heading α for the path point is calculated according to the formula:

$$\alpha = (\mathrm{p(RIGHT)} - \mathrm{p(LEFT)}) \cdot \alpha_{max} \,, \tag{3.3}$$

where α_{max} is the maximum desired increment in radians. Step in the x-axis r_x relative to the current position is calculated as

$$r_x = \mathrm{p(STRAIGHT)} \cdot \cos(\alpha) \cdot r_{max} \,, \tag{3.4}$$

where r_{max} is the maximum desired step length. Step in the y-axis r_y is calculated as

$$r_y = \mathrm{p(STRAIGHT)} \cdot \sin(\alpha) \cdot r_{max} \,. \tag{3.5}$$

Altitude is considered to remain constant so for each point it is set to $(1.8\,\mathrm{m} - actual\_height)$ above the ground. A similar altitude was used in the dataset. Parameters α_{max}, r_{max} can be tuned for better performance.

Such policy was used during the experiments because the ability to combine the algorithm with a path planner is a priority.

3.4.3 Filtering of the neural network output

The implemented neural network produces results online, at 30 Hz. During the operation, a lens flare or other short-term disturbance can occur. It leads to rapid changes in the prediction and combined with the path generation creates wrong, potentially unsafe points in the path. Therefore, the output must be filtered. To get rid of high frequency changes, a low-pass filter is used. In this thesis, a simple yet effective low pass filter was employed: a moving average filter. For the frame number n, it also remembers $k-1$ previous predictions and outputs the average of these k predictions. Value $k = 15$ showed good results during tests and thus was used in this work. Its formula is

$$y_n = u_n + u_{n-1} + u_{n-2} + ... + u_{n-k+1} \,, \tag{3.6}$$

where y_i is the output of the filter for the frame number i, u_i is the raw output from the neural network for the frame number i.

3.4.4 Pathfinder

Pathfinder is a part of MRS system that creates a trajectory for the vehicle to travel to the desired position. However, when no obstacle avoidance technique is used, the generated trajectory may lead to a collision with an obstacle. To increase the safety during the experiments a LiDAR was used. It allows to create a map of unsafe points online (those where obstacles are present) and to avoid them when planning a trajectory.

Chapter 4

Simulation

4.1 Software

For simulation of such complex application, a dedicated software is needed. The main tool that was used for simulation and further usage on a real robot is the Robot Operating System (ROS). It is a framework for robotic applications which offers abstraction from hardware, and even contains already implemented functions for communication between vehicles, sensors, cameras and other used hardware, both real and virtual. The communication is performed using high-level messages. For every hardware unit a node is created. These nodes can subscribe (receive information) or publish to some topic. A topic represents a virtual pipe, through which the information is transferred. For example, there is a program running on the robot PC. The program is subscribed to the topic `"/image"`, where images are published by the driver node `"/camera"`. Then, it processes the images, makes a decision, and commands the speed using the `"/speed"` topic of the control node `"/drone"` [29]. Specifically in the MRS UAV system there are topics also for trajectory, odometry and other features.

Another tool used for simulation is Gazebo. It offers real-time graphical visualisation of the ongoing experiment. Realistic scenarios can be created in this simulator, its engine allows for shaders, different lighting conditions, and even physics simulation. Therefore, a virtual model of the use-case scene and conditions is usually designed, including the vehicle itself. Such model makes it possible to conveniently test the designed software before proceeding to real-world experiments as the cost of a mistake in the real world can be high.

4.2 Simulation setup

For trail following simulation I decided to use the MRS pre-configured setup for one-vehicle simulation (available at `https://github.com/ctu-mrs/simulation`). In my case, the system runs inside a singularity container. Default simulation does not include the onboard camera. Thus the front-facing camera was added to the robot model. The "Baylands" world (publicly available at `http://models.gazebosim.org/`) was selected for simulation, because it has a section of forest path in it. Overview of this world is in Fig. 4.1.

The code must be modified to run in simulation. The only needed change is renaming the topic, from which the images are received to match the name of the camera in the simulation.

4.3 Simulation results

During the experiment in simulation, performance of the implemented algorithm was tested in close to real-world conditions. The MAV successfully managed to follow the trail

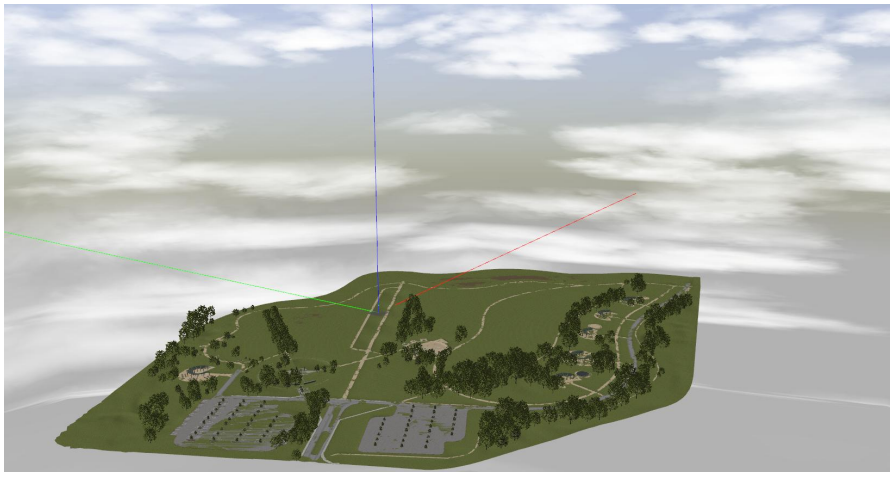
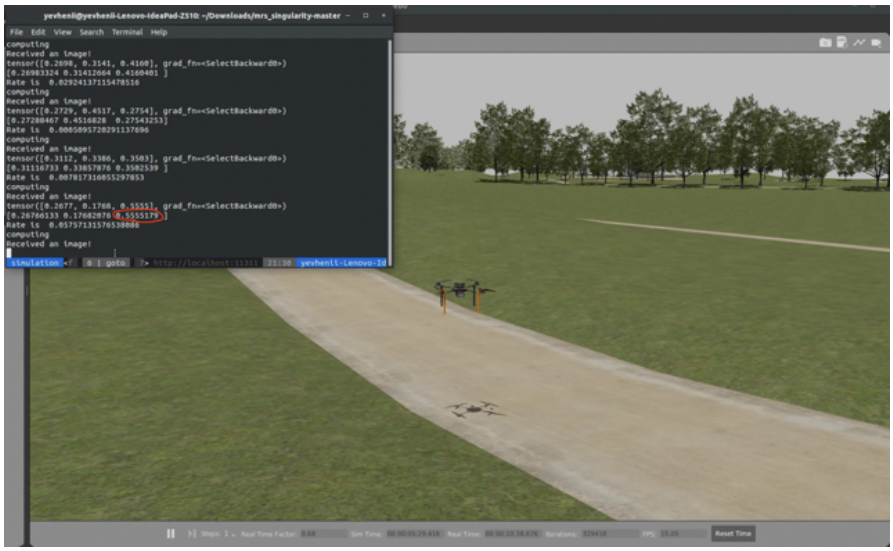
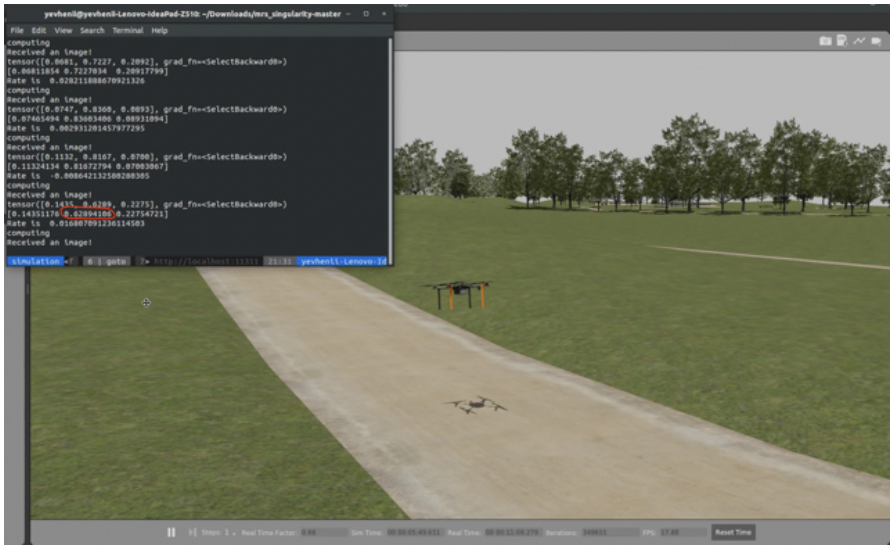


Figure 4.1: "Baylands" world. Image from `docs.px4.io`.

without getting lost (Fig. 4.2). There were some oscillations during the heading correction, but these can be removed by using a more complex heading regulation and fine tuning. The overall performance is good and the proposed methods for navigation along the path worked as expected. The vehicle stayed on the trail for as long as the simulation was going, without fails. However, the used map is relatively simple and has no obstacles or confusing factors. It was used to check the basic performance. Further and more detailed testing is conducted in real-world conditions, described in Sec. 5.2.



(a) The neural network predicts that the MAV is most likely pointed right relative to the path, when it is actually pointed right. Vehicle turns left.



(b) Prediction "straight" is most likely, the MAV is pointed straight. Vehicle moves forward.

Figure 4.2: Example images from the simulation.

Chapter 5

Real-world evaluation

5.1 Neural network performance test

For evaluation of the neural network performance in real-world forest conditions, a preliminary experiment was conducted. The on-board computer and the camera were powered from a battery and manually moved through a forest. Data was outputted in real-time to a laptop and the prediction correctness was evaluated (Fig. 5.1).

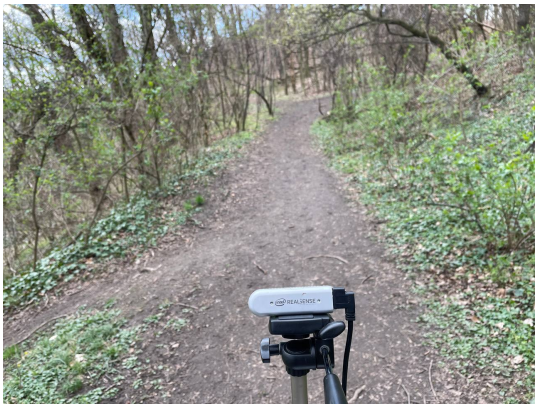
The testing showed good neural network performance. The accuracy of determining the "left" and "right" classes in the tested situations was 100%. There was once a situation though, where the neural network was outputting only a small probability of the "straight" class, when looking straight. Probabilities of the "left" and the "right" were the same, close to 50%. But the issue was clearly dependent on the tilt angle of the camera, after tilting it a few degrees down, the problem was solved. A possible source of the issue could be not only the neural network fail-case, but also a lens flare.

5.2 Complete tests on the vehicle

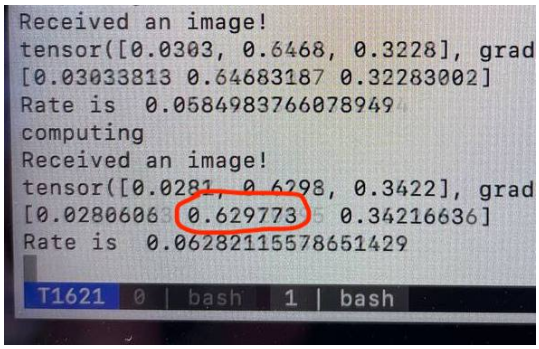
A real-world evaluation was conducted in a forest near the Czech town Temešvár. Together with the MRS team, suitable forest trails were found, where the algorithm was then thoroughly tested. The trails were of different complexity (visually) and contained slight obstacles on the sides (Fig. 5.2).

The first trail (Fig. 5.2a) was a 2.3 m wide partly dirty asphalt road with turns, surrounded by trees and bushes. Due to the limited FOV of the camera and the relatively large width of the trail, when the vehicle was in the centre of the road, it saw only a grey dull pattern with no features and because of that outputed 100% probability of "straight" class even when not pointed straight. But as soon as the MAV got close to the road edge, the neural network recognised it and gave the command to turn in the opposite direction. Then, it started flying along the path and slowly got closer to the other side and the same situation happened. However, this "zig-zag" behaviour did not affect the performance too much. But in some situations, flying close to the road edge is dangerous and during this test, the path planner prevented the vehicle from executing several waypoints due to their proximity to an obstacle (Fig. 5.3a). The problem of flying close to the sides can be solved by using a camera with a wider FOV, which can capture both sides simultaneously. The autonomously travelled distance on the first path was 130 m (Fig. 5.3b).

The second trail (Fig. 5.2b) was a 2-2.3 m wide completely dirt road with one sharp turn and fences on sides. It was hard for the MAV to stay on it. A lot of distracting features were present on the road and the trail itself was not well maintained. When analysing the images, it can be seen that the contrast between the trail and dry grass on the sides is very low.

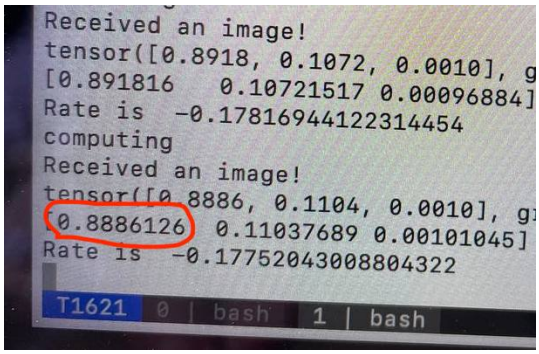


(a) Camera pointed straight.



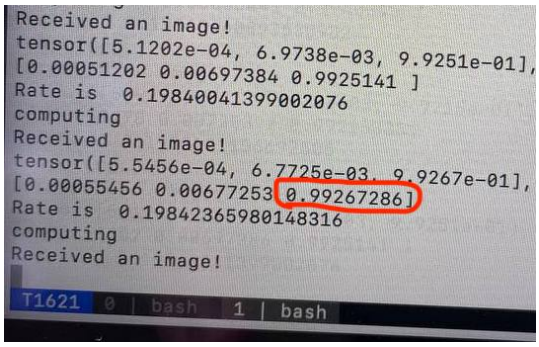
(b) Prediction: "straight" is most likely.

(c) Camera is pointed left.



(d) Prediction: "left" is most likely.

(e) Camera is pointed right.



(f) Prediction: "right" is most likely.

Figure 5.1: Neural network performance in the preliminary experiment.

Probably, colour adjustments of the camera are required and also including such low-contrast images to the dataset would help. Also, there was a ditch on the road which was classified as a trail by the neural network (Fig. 5.4a). However, when the obstacle avoidance prevented it from flying into the bush, the real trail reappeared in the field of view and the vehicle managed to return on track. At the end, it got distracted again. This trail was not followed with much success, but after the mentioned improvements, the results can be expected to improve. The total distance travelled autonomously on this path was 40 m (Fig. 5.4b).

The third trail (Fig. 5.2c) was a 2.8 m wide gravel and dirt road with slight turns.

(a) Trail 1.

(b) Trail 2.

(c) Trail 3.

Figure 5.2: The three trails used for testing in the real-world experiments.

(a) The MAV avoiding an obstacle.

(b) A satellite image of the travelled path from Google Maps.

Figure 5.3: The first experiment.

Contrast between the road and sides on the gravel part was better than on previous trails. On the dirt part, the contrast was much lower, causing the distinction between the trail and the sides to be less clear. But the distinction was still sufficient for the CNN to correctly classify the images. Its performance was better than on the other trails. The MAV flew a long way through it. However, in the end it got confused by logs on the ground and the vehicle started "following" one of them (Fig. 5.5a). It may have happened due to the road and the sides being both covered in dirt and from the vehicle camera perspective they were not distinguishable. The total distance travelled autonomously was 160 m on this trail (Fig. 5.5b). Images from the MAV including the prediction are presented in Fig. 5.7.

During the experiments, the vehicle was mapping the environment using the onboard LiDAR sensor. These maps are presented in Fig. 5.6. Approximately every meter, the position and the direction of the drone are visualised using a red arrow. The performance of the implemented algorithm can be seen from them. The obstacle avoidance did not change the heading, so all the heading corrections during the flight were made only by the trail following program. According to those images, the vehicle performed very good when flying through

(a) The MAV got confused by a ditch, photo before recovery.

(b) A satellite image of the travelled path from Google Maps.

Figure 5.4: The second experiment.

the first and the third trail. The arrows are pointed along the safe path.

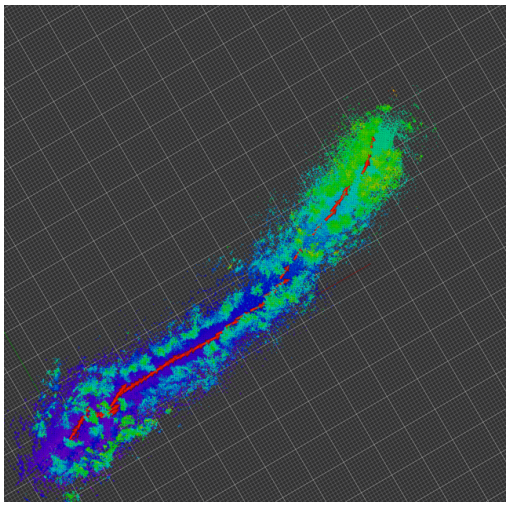
Experiment number	Travelled distance (m)	Flight time (minutes)
1	130	6:50
2	40	3:00
3	160	6:11

Table 5.1: Experiment statistics.

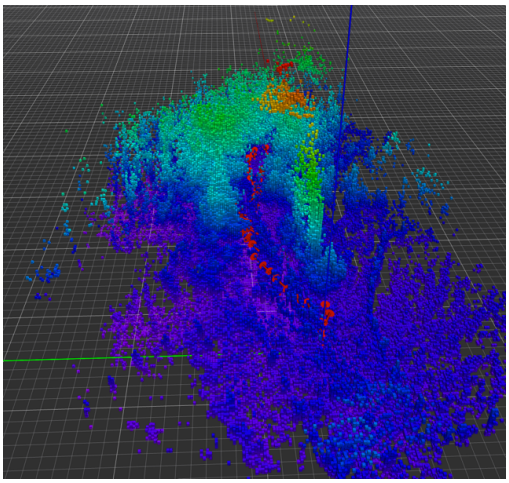
(a) MAV got confused by the log.

(b) A satellite image of the travelled path from Google Maps.

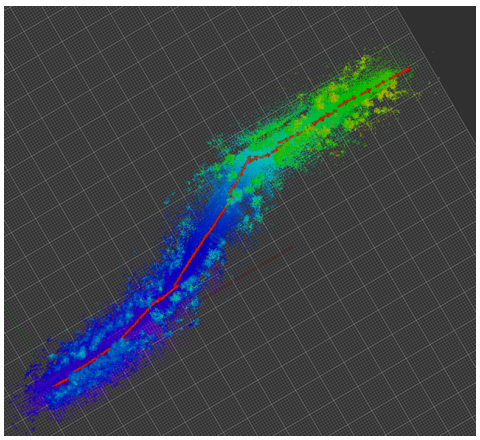
Figure 5.5: The third experiment.



(a) The first trail.



(b) The second trail.



(c) The third trail.

Figure 5.6: Maps of the environment, obtained using the onboard LiDAR sensor during flight. Poses of the MAV are marked with red arrows, occupied volume is marked with blue-green squares. Resolution of the major grid is 10 m per square.

Figure 5.7: Onboard images and classification results.

Chapter 6

Conclusion

In this thesis, a 3-class classification convolutional neural network was implemented to solve the trail following problem utilising a purely visual approach. Using only an RGB camera is a cheap solution and no other hardware than an onboard PC with a camera is required to run the program. Also, the neural network is supplemented by an algorithm, which generates path points according to the prediction. The program is able to run in the Gazebo simulator and in real-world conditions online and was tested in both virtual and real environments.

During the training process, the neural network showed up to 90% accuracy on validation. However, the dataset that was employed for training, was acquired by other cameras than those used in this thesis. During the experiments, the vehicle was able to fly long distances on curvy trails (up to 160 m) but got stuck when the trail was not contrast enough or distracting factors appeared in the image. This problem can be caused by the different FOV of the used camera, colour rendering, image sensor quality and even season. Therefore, the results can be improved by creating a new larger dataset with conditions close to the expected and training the network on it. Also, increasing the field of view could help, because more features can be captured by the camera.

Another possible improvement can be the usage of a segmentation neural network, as it allows to predict also the shape of the future trajectory. However, it may be challenging to distinguish the road from the sides during segmentation, and requires a big manually labeled dataset. Therefore, in this thesis only the classification approach was taken.

The implemented algorithm is a good baseline and an addition for more complex navigation solutions in forest conditions. This was demonstrated during the experiments, when the LiDAR-based obstacle avoidance and path planning run alongside with the trail-following neural network. It prevented the UAV from flying too close to dangerous obstacles like trees or bushes.

The MAV flew autonomously up to 160 m in a challenging forest environment using the proposed approach and the implemented program.

Chapter 7

References

[1] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[2] S. Back, G. Cho, J. Oh, X.-T. Tran, and H. Oh, "Autonomous uav trail navigation with obstacle avoidance using deep neural networks," *Journal of Intelligent & Robotic Systems*, vol. 100, no. 3, pp. 1195–1211, 2020.

[3] W. Chen, W. Wang, K. Wang, Z. Li, H. Li, and S. Liu, "Lane departure warning systems and lane line detection methods based on image processing and semantic segmentation: A review," *Journal of traffic and transportation engineering (English edition)*, vol. 7, no. 6, pp. 748–774, 2020.

[4] R. I. H. Abushahma, M. A. Ali, N. A. Abd Rahman, and O. I. Al-Sanjary, "Comparative features of unmanned aerial vehicle (uav) for border protection of libya: A review," in *2019 IEEE 15th International Colloquium on Signal Processing & Its Applications (CSPA)*, IEEE, 2019, pp. 114–119.

[5] Y.-C. Lin, Y.-T. Cheng, T. Zhou, R. Ravi, S. M. Hasheminasab, J. E. Flatt, C. Troy, and A. Habib, "Evaluation of uav lidar for mapping coastal environments," *Remote Sensing*, vol. 11, no. 24, p. 2893, 2019.

[6] I. Yaqoob, L. U. Khan, S. A. Kazmi, M. Imran, N. Guizani, and C. S. Hong, "Autonomous driving cars in smart cities: Recent advances, requirements, and challenges," *IEEE Network*, vol. 34, no. 1, pp. 174–181, 2019.

[7] M. Goyal, K. Tatwawadi, S. Chandak, and I. Ochoa, "Deepzip: Lossless data compression using recurrent neural networks," *arXiv preprint arXiv:1811.08162*, 2018.

[8] B. G. Maciel-Pearson, P. Carbonneau, and T. P. Breckon, "Extending deep neural network trail navigation for unmanned aerial vehicle operation within the forest canopy," in *Annual Conference Towards Autonomous Robotic Systems*, Springer, 2018, pp. 147–158.

[9] J. Yue, H. Feng, X. Jin, H. Yuan, Z. Li, C. Zhou, G. Yang, and Q. Tian, "A comparison of crop parameters estimation using images from uav-mounted snapshot hyperspectral sensor and high-definition digital camera," *Remote Sensing*, vol. 10, no. 7, p. 1138, 2018.

[10] Z. Zhou, M. M. Rahman Siddiquee, N. Tajbakhsh, and J. Liang, "Unet++: A nested u-net architecture for medical image segmentation," in *Deep learning in medical image analysis and multimodal learning for clinical decision support*, Springer, 2018, pp. 3–11.

[11] J. A. Paredes, J. González, C. Saito, and A. Flores, "Multispectral imaging system with uav integration capabilities for crop analysis," in *2017 First IEEE International Symposium of Geoscience and Remote Sensing (GRSS-CHILE)*, IEEE, 2017, pp. 1–4.

[12] S. Pedrozo, "Swiss military drones and the border space: A critical study of the surveillance exercised by border guards," *Geographica Helvetica*, vol. 72, no. 1, pp. 97–107, 2017.

[13] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *towards data science*, vol. 6, no. 12, pp. 310–316, 2017.

[14] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield, "Toward low-flying autonomous mav trail navigation using deep neural networks for environmental awareness," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 4241–4247.

[15] C. Yuan, Z. Liu, and Y. Zhang, "Fire detection using infrared images for uav-based forest fire surveillance," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 2017, pp. 567–572.

[16] Y. Zhang, X. Yuan, Y. Fang, and S. Chen, "Uav low altitude photogrammetry for power line inspection," *ISPRS International Journal of GEO-information*, vol. 6, no. 1, p. 14, 2017.

[17] A. Giusti, J. Guzzi, D. Ciresan, F.-L. He, J. P. Rodriguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, D. Scaramuzza, and L. Gambardella, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, 2016.

[18] L. R. Gómez and H.-J. Wunderlich, "A neural-network-based fault classifier," in *2016 IEEE 25th Asian Test Symposium (ATS)*, IEEE, 2016, pp. 144–149.

[19] L.-P. Chrétien, J Théau, and P Ménard, "Wildlife multispecies remote sensing using visible and thermal infrared imagery acquired from an unmanned aerial vehicle (uav).," *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. 40, 2015.

[20] S. Lhoest, J. Linchant, S. Quevauvillers, C. Vermeulen, and P. Lejeune, "How many hippos (homhip): Algorithm for automatic counts of animals with infra-red thermal imagery from uav," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 40, 2015.

[21] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, Springer, 2015, pp. 234–241.

[22] H. Wu, H. Zhang, J. Zhang, and F. Xu, "Typical target detection in satellite images based on convolutional neural networks," in *2015 IEEE International Conference on Systems, Man, and Cybernetics*, IEEE, 2015, pp. 2956–2961.

[23] F. Nex and F. Remondino, "Uav for 3d mapping applications: A review," *Applied geomatics*, vol. 6, no. 1, pp. 1–15, 2014.

[24] D. Yu, H. Wang, P. Chen, and Z. Wei, "Mixed pooling for convolutional neural networks," in *International conference on rough sets and knowledge technology*, Springer, 2014, pp. 364–375.

[25] D. Erdos, A. Erdos, and S. E. Watkins, "An experimental uav system for search and rescue challenge," *IEEE Aerospace and Electronic Systems Magazine*, vol. 28, no. 5, pp. 32–37, 2013.

[26] J. F. Keane and S. S. Carr, "A brief history of early unmanned aircraft," *Johns Hopkins APL Technical Digest*, vol. 32, no. 3, pp. 558–571, 2013.

[27] P. Santana, L. Correia, R. Mendonça, N. Alves, and J. Barata, "Tracking natural trails with swarm-based visual saliency," *Journal of Field Robotics*, vol. 30, no. 1, pp. 64–86, 2013.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[29] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "Ros: An open-source robot operating system," in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.

[30] M. A. Goodrich, B. S. Morse, D. Gerhardt, J. L. Cooper, M. Quigley, J. A. Adams, and C. Humphrey, "Supporting wilderness search and rescue using a camera-equipped mini uav," *Journal of Field Robotics*, vol. 25, no. 1-2, pp. 89–110, 2008.

[31] L. Merino, F. Caballero, J. Martinez-de Dios, and A. Ollero, "Cooperative fire detection using unmanned aerial vehicles," in *Proceedings of the 2005 IEEE international conference on robotics and automation*, IEEE, 2005, pp. 1884–1889.

[32] A. R. Girard, A. S. Howell, and J. K. Hedrick, "Border patrol and surveillance missions using multiple unmanned air vehicles," in *2004 43rd IEEE conference on decision and control (CDC)(IEEE Cat. No. 04CH37601)*, IEEE, vol. 1, 2004, pp. 620–625.

[33] P. H. Batavia, *Driver-adaptive lane departure warning systems*. Carnegie Mellon University, 1999.