**SLOVAK UNIVERSITY OF TECHNOLOGY
IN BRATISLAVA**

FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

Reg. No.: FCHPT-číslo práce

Touchless Drone Control

BACHELOR THESIS

2024 **Ivana Dukayová**

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

Reg. No.: FCHPT-číslo práce

Touchless Drone Control

BACHELOR THESIS

Study programme:	Process Control
Study field:	Cybernetics
Training workspace:	Institute of Information Engineering, Automation, and Mathematics
Thesis supervisor:	Ing. Martin Klaučo, PhD.

2024 **Ivana Dukayová**

Slovenská technická univerzita v Bratislave
Ústav informatizácie, automatizácie a matematiky

Fakulta chemickej a potravinárskej technológie
Akademický rok:
Evidenčné číslo: I

STU FCHPT

ZADANIE BAKALÁRSKEJ PRÁCE

Študent:

ID študenta:

Študijný program: riadenie procesov

Študijný odbor: kybernetika

Vedúci práce:

Vedúci pracoviska: prof. Ing. Michal Kvasnica, PhD.

Konzultant:

Miesto vypracovania: Ústav informatizácie, automatizácie a matematiky

Názov práce:

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Rozsah práce: 30

Zoznam odbornej literatúry:

1. J. Liu – D. M. de la Peña – P.D. Christofides: Distributed Model Predictive Control of Nonlinear Process Systems, AIChE Journal, 55, 1171-1184, 2009, ISSN 1547-5905
2. L.H. Swartz – C.L.E. Dynamic real-time optimization of distributed MPC systems using rigorous closed-loop prediction, Computers & Chemical Engineering, 122, 356-371, 2019, ISSN 0098-1354.
3. M. Bakošová – M. Fikar: Riadenie procesov. 2008. ISBN 978-80-227-2841-6.

Termín odovzdania bakalárskej práce: 08. 05. 2022

Dátum schválenia zadania bakalárskej práce: 21. 04. 2022

Zadanie bakalárskej práce schválil: prof. Ing. Michal Kvasnica, PhD. – garant študijného programu

Acknowledgment

Abstract

This project focuses on identifying gestures using a drone's camera and controlling the drone based on these gestures. Our target area is to create a reliable system for gesture recognition and drone control without the need for creating complex mathematical models.

Our work includes using the drone's camera to capture gestures and identifying these gestures using the MediaPipe library. We aim to recognize various gestures, including those that control drone movements such as ascents, descents, rotations, and taking photographs. For identification, we have implemented a neural network model that we trained on a dataset containing various hand gestures. In conjunction with the OpenCV library for image processing, the model is capable of recognizing and classifying gestures in real-time based on footage from the drone's camera.

Our work has enabled us to successfully identify various gestures and control the drone based on these recognized gestures. The project also demonstrates how gesture recognition can be a practical and interesting method for interacting with drones, allowing people to intuitively control these devices.

Abstrakt

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As robots become more common in our daily lives, there has been an increase in research on human-robot interaction. To enable more human-like interaction with robots, automatic gesture recognition systems are available and can replace traditional human-machine interactive devices such as keyboards, mouse, joysticks, etc. Hand gestures can make our lives and the lives of differently abled persons easier. It can be applied to virtual environments, robotics, home automation, clinical operations, game control, desktop/tablet applications, delivery service, sign language and more.

Drones are widely used in most of these applications. They are commonly used for sports event coverage, ariel photography and quickly transporting equipment to emergency areas. Drones are remotely controlled robots operated via a remote control device or smartphone. Developing automatic systems that can recognize hand gestures would greatly improve the ability to interact with drones intuitively, with a very low likelihood of errors. The drone should be able to recognize specific human gestures and respond accordingly.Our goal is to develop a real-time hand gesture recognition system for human-drone interaction.

Chapter 2

Theory

2.1 Machine Learning

Due to the advancement in the field of Artificial Intelligence (AI), the ability to tackle entire problems of machine intelligence. Nowadays, Machine Learning (ML) is becoming a hot topic due to the direct training of machines with less interaction with a human. The scenario of manually feeding the machine is changing in the modern era, it will learn automatically. The methods of learning can be roughly categorized into two main groups - supervised learning and unsupervised learning. Nowdays we know of other methods like reinforcement learning, semi-supervised learning, and many hybrid approaches.

2.1.1 Supervised Learning

When the data takes the form of input variables and target values for the output, supervised learning is used. The mapping function from the input to the output is learned by the algorithm. It's a costly method for problems where data is limited since large-scale labeled data samples are available.

- Classification
  One of a known number of categories is represented by the output variable. For instance, "dog" or "cat," "positive" or "negative."

- Regression
  The value of the output variable is continuous or actual. For instance, "geographical location," "price,"

2.1.2 Unsupervised Learning

podporne vektory, k means, dbscan

2.2 Neural Networks

An artificial intelligence technique called a neural network trains computers to process information in a manner similar to that of the human brain. It uses connected nodes or neurons arranged in a layered pattern to mimic the organization of the human brain. Computers can utilize this adaptive approach to learn from their errors and keep getting better. As a result, artificial neural networks make an effort to more accurately answer challenging problems, such as document summarization and face recognition.

Neurons are small, basic building components of neural networks. Each neuron takes inputs which it processes and outputs it as an input for next neuron. Neurons in the input layer, as well as in the output layer are defined. The neuron connections carry weights. The higher the weight, the bigger impact to the other neuron that is connected. A neuron normally takes the mean value of the connected neurons based on the connection weights. All neurons have transfer function, that determines whether or not to accept the triggered inputs from the connected neurons. The neuron decides, sets its value, and triggers the next neurons with that value. The value that the transfer function typically returns is a mix of the neuron's triggered value and current value. The majority of the work is done by the hidden neurons. However, as we are only concerned with the input and output neurons and are unaware of what occurs in the neurons between them, we refer to them as hidden neurons. That being said, most neural networks are multilayered, which means there are also hidden layers of neurons besides the input and output neurons.

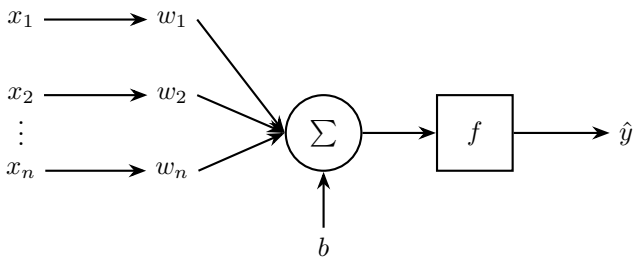


Figure 2.1: Schematic representation of a neuron with inputs, weights, a bias, an activation function, and an output.

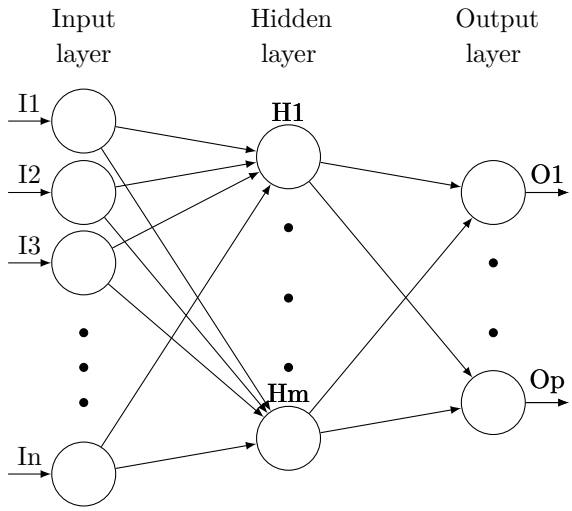


Figure 2.2: Structure of neural network

Feedforward Neural Network

Weights are assigned after an input layer has been identified. Larger weights contribute more significantly to the output than smaller ones, helping to determine the relative importance of each variable. After that, each input is multiplied by its corresponding weight before being added together. The output is then determined by passing it through an activation function. The node "fires," or activates, when its output surpasses a predetermined threshold, forwarding data to the network's subsequent layer. As a result, one node's output ends up becoming the next node's input. This neural network is classified as a feedforward network because of the way that data is passed from one layer to the next.

Activation Functions

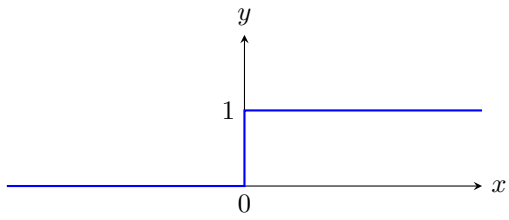
A weighted sum of signals received from the neurons on the preceding layer is fed into neuron's activation function. After that, the activation function's output is passed onto next layer of the network. These are main types of activation function used: threshold, sigmoid, rectifier, hyperbolic tangent, softmax

Threshold Function

The output signal computed by threshold functions varies based on whether the input is above or below a predefined threshold.

The formal definition of a deep learning threshold function in mathematics is as follows:

$$f(x) = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{if } x \geq 0 \end{cases} \tag{2.1}$$



The threshold function is sometimes referred to as a unit step function, as the image above illustrates. In computer programming, threshold functions are comparable to boolean variables. Their calculated value is either 0 (which is equal to False) or 1 (which is comparable to True).

Sigmoid Function

The data science community is familiar with the sigmoid function from its application in logistic regression. It tells us the probability of whether a

Any value can be entered into the sigmoid function, but it will always return a value between 0 and 1.

The sigmoid function has the following mathematical definition:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.2}$$

For evaluating a single metric (feature) the sigmoid function has threshold value of 0.5 for binary decisions.

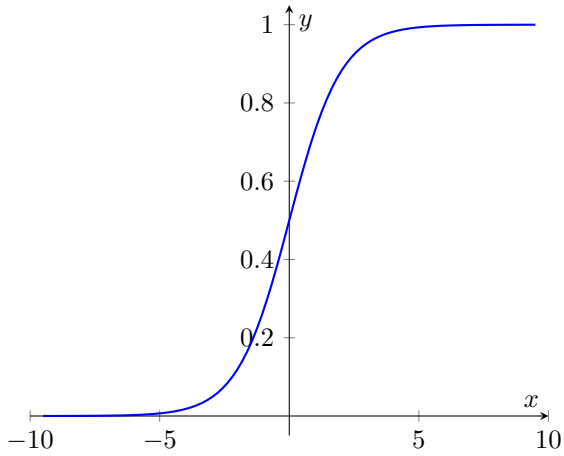


Figure 2.3: sigmoid

The smoothness of the sigmoid function's curve gives it an advantage over the threshold function. This implies that derivatives can be computed at any point on the curve.

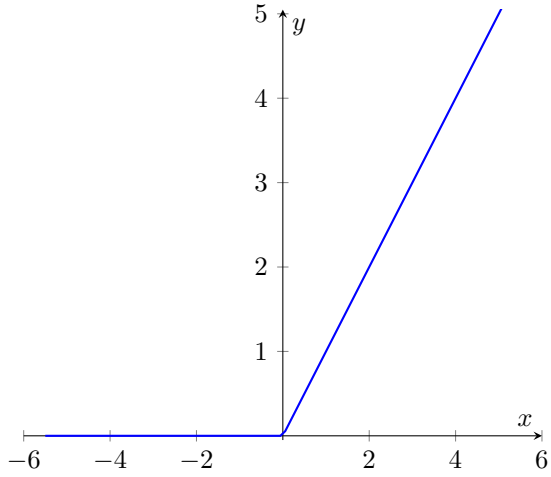
Rectifier Function

Unlike the sigmoid function from, the rectifier function does not have the same smoothness property. It is still highly used in the deep learning community.

The definition of the rectifier function outputs 0 if the input value is less than 0. The function outputs the input if it isn't. It helps to maintain mathematical stability and keep learned values from being stuck around 0 or jumping into infinity.

$$Relu(x) = max(0, x) \tag{2.3}$$

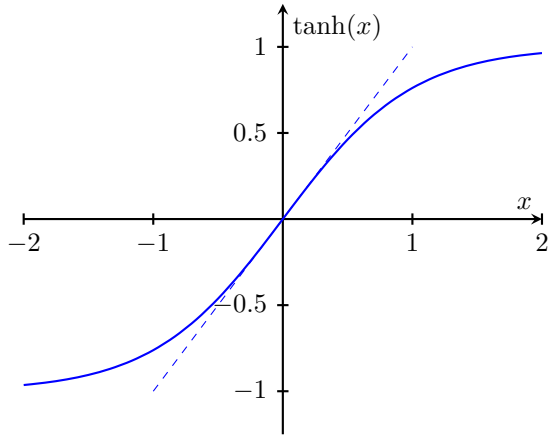
Rectified Linear Unit activation functions, or ReLUs for short, are a common term for rectifier functions.



Hyperbolic Tangent Function

The activation function based on a trigonometric identity is the hyperbolic tangent function. Below is its mathematical definition:

$$\varphi(x) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.4}$$



The hyperbolic tangent function resembles the sigmoid function, but all of its output

values are shifted down.

SoftMax Function

An activation function called Softmax is used to convert numbers or logits (outputs of neural network) into probabilities. A vector containing the probabilities of every possible result is the result of a Softmax. The vector's probabilities add up to one for every possible outcome or class.

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K \tag{2.5}$$

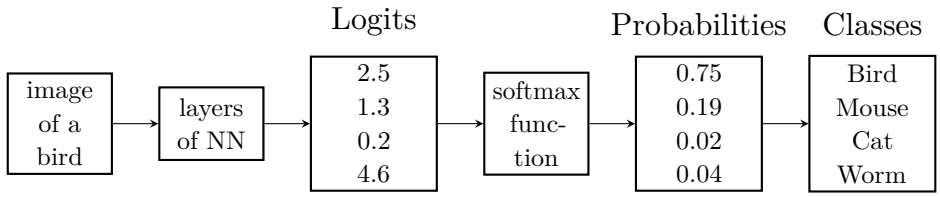


Figure 2.4: A diagram of the NN softmax classification process.

Training A Neural Network

The weights of the connections between the nodes are randomly chosen, and neural networks then use a technique known as backpropagation to learn new, improved connection weights based on the inputs and outputs that are predetermined . Neural networks can therefore learn. The backpropagation algorithm works as follows:

1. Forward Pass: The input data is passed through the network (from input to output layers), calculating the output of each layer until the final predictions are made.
2. Loss Calculation: The difference between the predicted outputs and the actual labels is calculated using the loss function specified.
3. Backward Pass (Backpropagation): The gradient (partial derivatives) of the loss function with respect to each weight is calculated backward from the output layer to the input layer. This involves applying the chain rule from calculus to propagate the error backward through the network.

4. Weight Update: The weights are updated using an optimization algorithm (like Adam). The optimizer uses the gradients calculated during backpropagation to adjust the weights in a direction that minimizes the loss. Usually the weights are computed using a learning rate, errors, neuron inputs, and the previous weight.

This entire process is repeated for a specified number of epochs or until another stopping criterion is met (like the early stopping callback, which halts training if the validation loss doesn't improve for a set number of epochs).

Cost functions

As we train the model, we'll want to evaluate its accuracy using a cost (or loss) function. Loss function tells us about how good our algorithm models our dataset.The loss error is calculated for each training sample, while the loss function represents the whole set of m samples. The cost function represents the average loss error for each sample. It is a function that assesses how well a machine learning model performs with a given set of data. The error between expected and predicted values is quantified by the Cost Function and shown as a single real number. Cost functions can be formed in a variety of ways, depending on the nature of the problem. For a model using a formula ,

$$\hat{y} = wx$$

where $\hat{y}$ is predicted value, x a vector of data used for prediction and training and w the weight.

The commonly used loss function is the mean squared error (MSE). In the equation below,

i represents the index of the sample, $\hat{y}$ is the predicted outcome, y is the actual value, and m is the number of samples.

$$MSE = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2 \tag{2.6}$$

Crossentropy function
Other names for cross-entropy loss include logistic loss, log loss, and logarithmic loss. Every predicted class probability is compared to the actual class probability, and a loss function is used to penalize the probability according to its deviation from the

actual expected value. Because the penalty is logarithmic, large differences near to 1 will result in a large score, and small differences approaching 0 will result in a small score. Cross-entropy loss in a perfect model is 0. What is meant by cross-entropy is

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{2.7}$$

where M is number of classes, y is the binary indicator if class of label c is the correct classification for observation o and p is the predicted probability.

When dealing with labels that are one-hot encoded, such as 3-class classification problem, where the values are [1,0,0], [0,1,0], and [0,0,1], categorical cross-entropy is employed. Labels in sparse categorical cross-entropy are encoded with integers, such as [1], [2], and [3] for a 3-class problem.

There are many other types of cost functions used in ML, e.g. Root Mean Square error (RMSE), Binary Cross Entropy Loss Function, Categorical cross-entropy, Hinge Loss, Kullback-Liebler Divergence LOSS (KL-Divergence), Huber Loss. Choosing a cost function will be influenced by the type of problem, the output activation function and network architecture.

Hyperparameters of Neural Networks

- Learning Rate is a crucial parameter that controls to which extend the model needs to be modified. It is determines the step size at each training iteration. It is usually a value between 0 and 1. We can determine the direction of a loss function's optimum by computing the loss function's gradient. The step size that in that direction is determined by the learning rate parameter.

- Batch size tells us how much data is changed during 1 cycle (one epoch) of training. Larger batch sizes could lead to faster training, but with a trade off for lower accuracy or overfitting. The best size will depend on a number of variables, such as the size of the training dataset, the complexity of the model, and the available computational resources.

- Epochs define the number times that the learning algorithm will work through the entire training dataset. Every sample in the training dataset has had a chance to update the internal model parameters after one epoch. One or more batches make up an epoch.

Optimalizators of Neural Networks

- Gradient Descent is a basic optimalization algorithm in classification problems and linear regression. It uses first derivates of cost function to find its minimum. The batch gradient descent averages and modifies the weights after gathering all the cost function values from every row in a single epoch. However, it can sometimes get stuck on local minimum and be unable to determine the global minimum.

- Stochastic Gradient Descent is similar to adaptive learning in boosting methods of ML. The network uses the value from each iteration and gradually updates the weights according to how much weight's are responsible for error. As it reaches global minima, it is more effective than Gradient Descent.

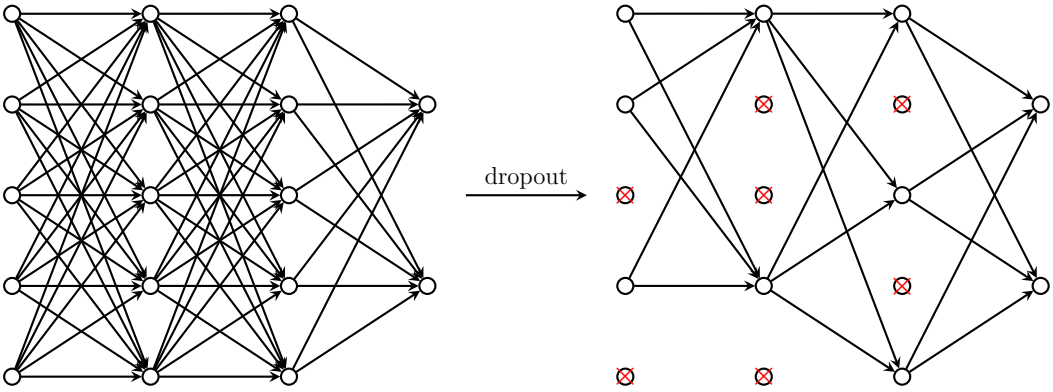
- Adam is convenient for most convex optimization problems with large datasets. It combines adaptive methods and momentum methods, where it takes into account the moving average of the gradient's first and second-order moments of the gradient. This allows it to effectively adapt the learning rates for each parameter. It is an extension to stochastic gradient descent.

Regulations of Neural Network

When talking about regulation of neural networks we can mention techniques like dropout or early stopping.

Dropout is a regularization strategy for neural networks that, with a given probability p, drops a unit (along with connections) during training. The goal is to stop co-adaptation, which occurs when a neural network becomes overly dependent on a single connection and may be an indication of overfitting. It makes intuitive sense to think of dropout as the formation of an implicit neural network ensemble.

An intuitive technique for training just enough, is early stopping. It prevents the model from learning on 'noised' dataset. When training, after every epoch, the model is evaluated using a validation dataset. The training process is terminated if the model's performance on the validaion dataset begins to drop (for example, if loss starts to rise or accuracy starts to fall).



2.3 Techniques used in Computer Vision

In Computer Vision (CV), ML plays an important role in extracting important information from images. CV successfully contributes to various domains, surveillance system, optical character recognition, robotics, suspect detection, and many more. The direction of CV research is going towards healthcare domain, medical imaging (MI) is the emerging technology, play a vital role to improve image quality and recognized critical features of binary medical image, covert original image into grayscale and set the threshold for segmentation. Within computer vision, three key tasks stand out: segmentation, detection, and classification.

Image Classifiaction

It's a known fact that the image we see as a whole is made up of hundreds to thousands of tiny pixels. Before computer vision can determine and label the image as a whole, it needs to analyze the individual components of the image. That is why image classification techniques analyze a given image in the form of pixels and accomplish this by treating the picture as an array of matrices, the size of which is determined by the image resolution. The pixels of the digital image are taken and grouped into what we know as "classes." From this point on, the procedure will vary depending on the algorithm. To guarantee that it is not left entirely on the final classifier, the selected algorithm will convert the image into a series of key attributes. These characteristics aid the classifier in identifying the subject matter and class to which the image belongs. We can say that the image classification pipeline looks like this: image pre-processing -> feature extraction -> object classification

Based on the nature of the problem, there are different types of image classification

methodologies. They are binary, multiclass, multilabel, and hierarchical.

Binary classification divides unknown data points into two groups using an either-or logic for labeling images. Binary classification is used to handle many different yes/no problems, such as analyzing product quality to determine whether a product has faults, and many more tasks requiring judgment calls.

As the name implies, multiclass divides objects into three or more classes, whereas binary classification divides objects into two classes. It's highly helpful in a variety of fields, including medical diagnostics (disease categorization), NLP (sentiment analysis in situations involving several emotions), etc.

The multilabel method permits an object to be allocated to more than one label, in contrast to multiclass classification, which assigns an image to a single class. For instance, you could have to categorize multiple colors in an image. Taking a picture of a salad, a picture of one will feature red, orange, yellow, purple, and other colors. Consequently, several colors will be used as labels on a single image.

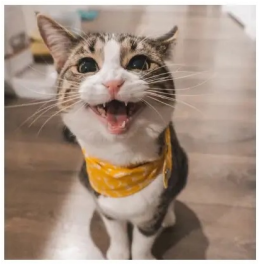
The process of classifying classes into a hierarchical structure based on their similarities is known as hierarchical classification. A lower-level class is more definite and detailed, while a higher-level class represents larger categories. If we're classifying a dog, our first model would recognize dog vs other animal. If a dog is correctly predicted, another model will be used to classify the breed of the dog into border collie, golden retriever, poodle. All features of higher-class attributes will be hierarchically contained in the latter ones. Hierarchy allows for effective information transfer between related classes as well as a flexible and interpretable framework for organizing and representing complex visual concepts.

Image Classification correlates one class from the training data with a whole image or video frame regardless of the amount of information present. It involves assigning labels, is suitable when fine-grained information is not necessary. During the model training stage, publically accessible datasets are frequently employed to enable correct data labeling.

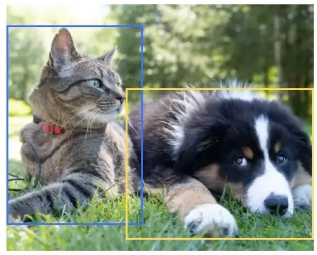
Object Detection

The classification is advanced in object detection. It not only classifies many objects in an image, but also provides annotations for the bounding boxes that correspond to each entity's position. The CV model's extra benefits enable its implementation in a number of practical contexts as these models are only a small portion of detecting, labeling,

and annotating objects in images or videos. Faster R-CNN, YOLO (You Only Look Once), and SSD (Single Shot MultiBox Detector) are a few popular object detection algorithms. These algorithms are tailored to specific application requirements and differ in terms of speed, accuracy, and trade-offs. An example of an object detection model is a model for human detection. Model would return a bounding box around each person with a label



Cat



Cat, Dog

Figure 2.5: Comparison of classification (left) with output: 'Cat' and object detection (right) with output: 'Cat, Dog' and their localization with bounding boxes

Image Segmantation

Since they both serve the same purpose, object detection and image segmentation are comparable. Both techniques identify items in pictures and provide coordinates for locating objects. However, segmentation algorithms produce accurate masks that cover objects at the pixel level, as opposed to creating whole boxes around them.

Annotations for image segmentation include the exact pixel locations of any instances that are present in the image. Image segmentation is better suitable for practical uses because of its accurate results. However, compared to object detection, picture segmentation models are computationally expensive due to algorithm complexity. The example use is in applying visual effects like background blurring and makup effects to the image. The functionality identifies specific textures, colors and segments object features within image data. It is also widely used ub medical imaging, self-driving cars or satelite imaging.

In the figure figxx a U-Net, an architecture introduced by Olaf Ronneberger, Philipp Fischer and Thomas Brox in 2015 is used. It's a convolutional neural network that

was specifically designed to be used in Biomedical Imaging. A pretrained model MobileNetV2 is used as a lightweight, efficient feature extractor for the input image. It would analyze the photo of the dog and create a set of feature maps highlighting important visual attributes like edges, textures, and shapes relevant for the segmentation task. Then, with the feature maps provided by MobileNetV2, U-Net would take on the task of image segmentation. Its architecture, designed to work with fewer data points and to be precise in delineating object boundaries, would use the feature maps to generate a predicted mask. It would attempt to closely replicate the true mask by classifying each pixel as belonging to the dog or the background. A Pix2Pix model is then used to generate a segmented image that tries to match the true mask, learning the mapping from the input image to the segmentation mask during training.

For this task it is also possible to use interactive segmantation, which devides image into two regions: selected object and everything else.It receives a location in an image, calculates the object's boundaries at that location , and returns image data that defines the object's area.

Figure 2.6: Landmarks for pose estimation model by MediaPipe and U-Net segmentation on an image from Oxford-IIIT Pet Dataset (Parkhi et al, 2012).

2.4 Mediapipe

MediaPipe is a framework used to create machine learning pipelines for time-series data like video and audio. Google initially developed it to process real-time video and audio analysis on YouTube. In 2019, the public release allowed researchers and developers to incorporate MediaPipe into their projects. Unlike other machine learning frameworks that require high computing power, MediaPipe can run efficiently on devices with low power, such as Android and IoT devices. It consists of the MediaPipe framework and MediaPipe solutions. The MediaPipe framework is developed using C++, Java, and

Objective C programming. MediaPipe solutions include 16 pre-trained TensorFlow and TensorFlow Lite models built on top of the MediaPipe framework for specific use cases.

MediaPipe Hands

The problem of detecting hands is somewhat intricat. The model must be able to recognize hands and it must function across a wide range of hand sizes with a big scale span so it must be robust. It is rather difficult to identify hands based just on their visual traits since hands lack the high contrast patterns.

MediaPipe Hands is a solution that tracks hands in real-time. MediaPipe Hands utilizes a combination of object detection, classification and regression to recognize and track hands within a given image or video frame. The pipeline consists of two models: palm detector and a hand landmark model.

The palm detector is a single-shot detector model that uses a orientated hand bounding box to locate palms on a whole input image. This is done before landmark detection because it is easier to estimate bounding boxes of rigid objects like palms than detecting hands with articulated fingers. It ignores many aspect ratios and so the bounding boxes are only squared. The model uses both classification (hands, no hands), and object detection (predicting a bounding box around the detected hand).

The Hand Landmark Model runs subsequently with the palm detection model and uses regression to precisely localize 21 3D coordinates within the hand region. Each landmark has a specific location on the hand that can vary within the pixel grid of an image. To capture the exact position, continuous values are used, allowing the model to indicate precisely where each landmark is located on the hand in terms of its x and y (and possibly z) coordinates within the image. As the hands move and rotate freely, leading to a wide range of possible positions and orientations for each landmark, Continuous outputs enable the model to track these changes smoothly and accurately across frames in real-time. The model is resilient to self-occlusions and partially visible hands, and it learns a consistent internal hand posture representation. The results form the model contain 21 hand landmarks consisting of x, y, and z, a hand flag that indicates the likelihood that a hand is present in the input image and a binary system of handedness, e.g. left or right hand. The coordinates x and y for each landmark are normalize to [0, 1] by image width and height. See more in Data Normalization.

The model was trained both on real-world and synthetic datasets, noting that the wrist point being learned only from synthetic images. More than 30,000 manually

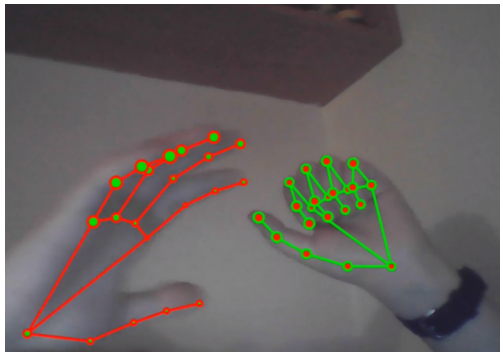


Figure 2.7: Hand landmarks model output

annotated real-world images were used and the model is very robust, allowing it to detect and map hand landmark points accurately, even on partially visible hands in most cases. In 2.7 my hands are not facing upfront but the model still finds all the landmarks for both hands. If tracking 2 hands, it also shows which one is left or right with different colors.

For encountering a tracking failure, There was also another model output created for tracking failures. It generates the likelihood that the given crop actually contains a hand that is reasonably aligned. The detector is triggered to reset tracking if the score falls below a predetermined threshold.

Data Normalization

The MediaPipe hand landmarks model provides coordinates for hand landmark points based on the position of pixels containing those points in an image. As a result, the coordinates of two images of the same hand sign with different placements in the frame can have significantly different distances between them. This makes it more challenging to train the model.

To solve this problem, the wrist's landmark point has been considered with coordinates [0,0], and the coordinates of all other landmark points were adjusted accordingly. First, the coordinates' values of the wrist's landmark point are subtracted from all coordinates' values.

Then, the coordinates were normalized to be between 0 and 1 by dividing them by

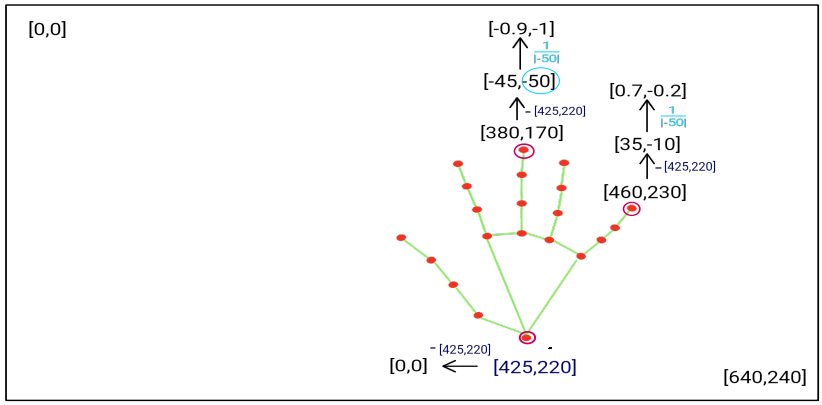


Figure 2.8: Process of normalization of landmark coordinates

the largest absolute value of the difference. Finally, the normalized coordinates were collected in the landmarks list. The coordinate normalization procedure is shown in Fig 2.8.

MediaPipe Face

With support for multiple faces and six landmarks (left eye, right eye, nose tip, mouth, left eye tragion, and right eye tragion), MediaPipe Face Detection a quick solution for face detection. BlazeFace, a compact and efficient face detector designed for mobile GPU inference, serves as its foundation. Because of its real-time performance, the detector can be used with any live viewfinder experience that needs a precise facial region of interest as an input for other task-specific models, like face region segmentation, facial feature or expression classification, and 3D facial keypoint estimation (like MediaPipe Face Mesh). BlazeFace leverages an enhanced tie resolution technique as an alternative to a GPU-friendly anchor mechanism adapted from Single Shot MultiBox Detector (SSD), and a lightweight feature extraction network that is similar to but different from MobileNetV1/V2.

The Hand company, availability, types of models, what they offer, devices options, what we use

Figure 2.9: Outputs of face detection model with different lightings

2.5 Gesture Recognition

The process of recognizing hand gestures as input, mapping them to a representation, and converting them into a purposeful command for devices is known as gesture recognition. The goal is to identify and process explicit hand gestures for output. Dynamic gestures refer to a sequence of poses that change over time, while static gestures remain constant. Static hand gestures can be classified based on their temporal relationship and recognized by analyzing features such as spatial position, orientation, contour, and texture. We will focus on recognizing static gestures using a vision-based method.

2.6 Drone Control

Chapter 3

Practical Part

3.0.1 Data Collection

Actually, many datasets that contain images of hand gestures are publicly available to use. In this project, we recorded our data. We did so with the MediaPipe hand-tracking model. We achieved this by capturing camera footage of each gesture being held for a certain period of time and moving around for variety. During the capture process, we pressed a key to label each gesture. Our dataset consisted of over 5000 samples. The different hand conditions for dataset samples include images from both the right and left hand, palms facing forward or backward towards the camera, and various degrees of hand position captured by the camera.

The model outputs x, y, and z coordinates of hand landmark points from images, with only x and y coordinates necessary and sufficient for training the final model. As a result, the z coordinates were eliminated, and the remaining x and y coordinates of hand landmarks for various hand signs are stored in a csv file for each hand landmark. The coordinates in Figure 3.1 represent the data points used to generate the final dataset and train the final model. The first column represents the label of a gesture, the next two columns are coordinates of landmark 0 at the wrist, and so on. Csv contains many combinations of landmark coordinates for each gesture.

3.0.2 Model

Training the Model

2 modely na rozonavanie gest - aplm a landmark points

After creating the dataset, the next task is to train a feedforward neural network with

Gesture	lm0.x	lm0.y	lm1.x	lm1.y	...	lm20.x	lm20.y
0	0	0	0.19444	-0.07639	...	-0.19444	-0.77083
1	0	0	-0.42857	0.08929	...	0.12500	-0.44643
3	0	0	0.23077	-0.14530	...	0.28205	-0.85470
4	0	0	0.16667	-0.10526	...	-0.02632	-0.30702
5	0	0	0.21698	-0.26415	...	0.26415	-0.05660
6	0	0	-0.12791	-0.12209	...	0.13953	-0.16279
7	0	0	-0.22973	0.11486	...	0.04054	0.41216

Figure 3.1: A selection of the saved csv file

the data. We used an available model by MediaPipe. The neural network (NN) has one input layer and one output layer, along with five hidden layers. These hidden layers include three dense layers and two dropout layers. The dense layers perform matrix-vector multiplication in the background, while the dropout layers reduce overfitting by randomly modifying the outgoing edges of hidden layer neurons and resetting them to 0 at each iteration during the training process. The output layer of the NN has the same number of neurons as the possible hand gestures it can recognize. Each neuron represents a specific hand gesture and produces an output value indicating the probability of that gesture being present in the input. Figure 3.2 shows the model's configuration. The model was built using an Adam optimizer, which is efficient and suitable for training models with large data and parameters. The loss function used was Sparse Categorical Crossentropy, which measures the loss between actual and predicted labels. The accuracy metric was used to evaluate the model, indicating how often the prediction matches the actual label. The training model was initially set to have only 4 hidden layers, but the training accuracy didn't exceed 90%. After updating to 5, without any other change, we got satisfactory results. The training accuracy obtained was 99.48% and the calculated loss was 0.031.

Face Recognition Model

Results and analysis of the model

The classification_report and confusion_matrix libraries from scikit-learn were used for quantitative analysis of the test dataset. The classification_report library produces an evaluation report of our model with accuracy, precision, recall, and F1 score matrices. Additionally, the 'support' matrix represents the model's real-time recognition

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dropout_18 (Dropout)        (None, 42)                0

 dense_29 (Dense)            (None, 256)               11008

 dropout_19 (Dropout)        (None, 256)               0

 dense_30 (Dense)            (None, 128)               32896

 dense_31 (Dense)            (None, 64)                8256

 dense_32 (Dense)            (None, 8)                 520

=================================================================
Total params: 52,680
Trainable params: 52,680
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.2: Configuration of the used feedforward neural network

performance.

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

The accuracy matrix calculates the number of correctly predicted labels by the model from the entire dataset (Eq.3.1). The precision matrix, on the other hand, measures the model's accuracy out of the predicted positives. It calculates the number of actual positives in the predicted positives and is an excellent measure to consider when the False Positive (FP) cost is high. Equation 3.2 depicts the mathematical formula of the precision matrix.

The recall matrix measures the number of predictions our model correctly labels as positives. It is a measure considered when false negatives have high costs. The mathematical formulation of the recall matrix is Equation 3.3.

The F1 score is calculated by combining both precision and recall, as shown in Equation 3.4. It is their harmonic mean.

Support is the number of samples in each class. The macro average of precision, recall and F1-score shows the average performance of the system across all classes, while

Classification Report

	precision	recall	f1-score	support
0	0.99	1.00	0.99	622
1	1.00	0.99	1.00	508
2	1.00	0.98	0.99	556
3	1.00	0.99	1.00	759
4	1.00	1.00	1.00	621
5	1.00	1.00	1.00	673
6	0.98	1.00	0.99	666
7	1.00	1.00	1.00	603
accuracy			0.99	5008
macro avg	1.00	0.99	0.99	5008
weighted avg	0.99	0.99	0.99	5008

Figure 3.3: Classification report of the model

the weighted average takes into account the class imbalance by weighting the metrics based on the number of samples in each class. Additionally, the loss value of 0.0310 (not shown) indicates how well the model is minimizing its errors during training.

The classification report of the implemented model in detail is shown in Figure 3.3.

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

$$\text{precision} = \frac{TP}{TP + FP} \tag{3.2}$$

$$\text{recall} = \frac{TP}{TP + FN} \tag{3.3}$$

$$\text{F1 score} = \frac{2PR}{P + R} \tag{3.4}$$

In the equations (3.1), (3.2), (3.3), and (3.4), TP, TN, FP, and FN represent True Positive, True Negative, False Positive, and False Negative, respectively.

The confusion matrix is a performance measure used in machine learning, particularly in classification tasks. In Python, the scikit-learn library can be used to create a

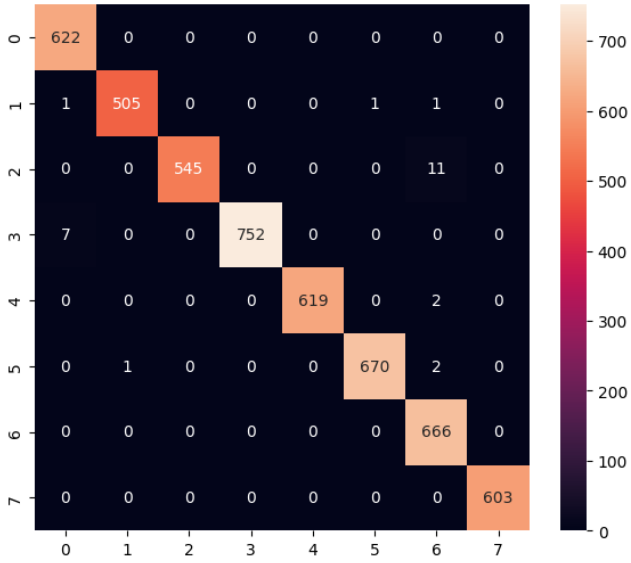


Figure 3.4: Confusion matrix of the model

confusion matrix. To obtain the datasets for the experiment, we collected and pre-processed the data before using it to predict hand gestures. The confusion matrix was used to observe the accuracy achieved by the model. The confusion matrix of the used model is shown in Fig 3.4. On the x-axis are predicted labels versus on the y-axis are actual labels.

Through the confusion matrix, we can determine which gestures are being recognized wrongly. For example, we can see that gesture 2 was 11 times misinterpreted as a gesture number 6. The recall of gesture 2 can then be calculated with Equation 3.5:

$$\text{recall}(2) = \frac{545}{545 + 11} \tag{3.5}$$

The system's performance is satisfactory, with almost perfect precision, recall, and F1-score in most classes, as well as high accuracy and weighted average.

3.0.3 Drone Implementation

The Tello drone is a small quadcopter with a vision positioning system and an onboard camera. It is controlled by a laptop computer using Wi-Fi or a smartphone using

2.4 GHz. Using Python, it is easy to utilize the functions of the Tello library. This library has built-in functions for interacting with the drone, managing communication, handling state changes, and controlling the drone through a series of events.

In our program, Tello starts taking commands after a key press and recognizes a gesture every 5 seconds. This way it can comfortably execute a command before recognizing it again. After adjusting speeds and sleep times, the Tello drone executed the received commands practically immediately and consistently.

In table 3.1 are displayed commands used in the project. We assigned them to the labels of our gestures. The gestures we use are shown in Fig **??**

Gesture Label	Command
0 - Palm	Move along y-axis (forward velocity)
1 - Fist	Move along -y-axis (backward velocity)
2 - Rock	Flip forward (upward velocity)
3 - Ok	Take-off or land (depending on whether in flight or landed)
4 - Peace	Takes a picture (saves in png)
5 - Like	Rotate 360°
6 - Up	Move up (ascending velocity)
7 - Down	Move down (descending velocity)

Table 3.1: Gesture commands for drone control

In addition, if we are not recognizing any gesture, the drone is ready to land on the palm. Because the drone can execute most of the commands only while flying, it takes off at the start of the program. It can also take off with gesture 4, so the program doesn't have to be reset.

Chapter 4

Conclusions

In this project, we have achieved touchless interaction between the drone and our hands. We implemented a machine learning model, specifically MediaPipe, for hand tracking. Our gesture recognition model underwent training and testing, and the confusion matrix presents its accuracy. The process is composed of 5 steps:

I. Drone capturing an image

II. Image processing via OpenCV

III. Gesture recognition

IV. Gesture - command conversion

V. Command execution

Gesture recognition is used in various sectors, including smart home automation and the medical field. It mainly focuses on human-machine interaction. The system receives input images from a drone camera connected to the device. The main goal of this project is to propose a real-time system with high accuracy. In summary, the project demonstrates how we can control a drone in an entertaining and educational way.

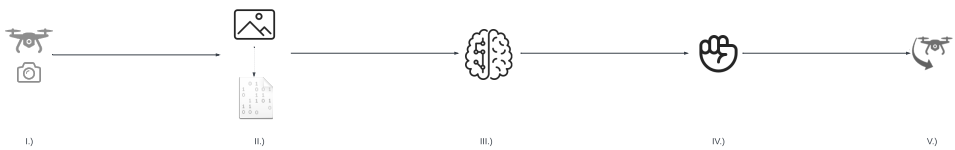


Figure 4.1: Steps of the process

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Google AI Blog. Mediapipe: On-device, real time hand tracking, 2019. Blog Post Accessed: December, 2023.

[3] Ahmed Eid and Friedhelm Schwenker. Visual static hand gesture recognition using convolutional neural network. *Algorithms*, 16(8):361, 2023. Journal Article DOI: 10.3390/a16080361.

[4] D. F. Escoté. djitellopy, 2021. GitHub Repository.

[5] MediaPipe. Mediapipehands, 2023. MediaPipeHands Source Code Accessed: November 2023.

[6] Joe Minichino. Opencv mediapipe hand gesture recognition, 2022. Project Repository on GitHub Accessed: December 2023.

[7] RYZE. Tello specs, 2023. TELLO Specifications Accessed: December 2023.

[8] F. Zhao, V. Bazarevsky, A. Vakunov, et al. Mediapipe hands: On-device real-time hand tracking, 2020. Research Paper Google Research, USA.