SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

Touchless Drone Control

SEMESTRAL PROJECT

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

2024 **Ivana Dukayová**

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.

Contents

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 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.

2.1.1 MediaPipe and hand landmark detection

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. This work used the MediaPipe solution to identify 21 3D landmark points on a hand from a single frame of a hand sign image. To achieve this, two dependent models are used simultaneously. First, a Palm Detection Model detects the palms of hands in the images since it is easier to detect rigid objects like palms and fists rather than a complete hand. Cropped palm images from this model are passed onto the next model, which is the Hand Landmark Model. The model accurately detects 21 3D

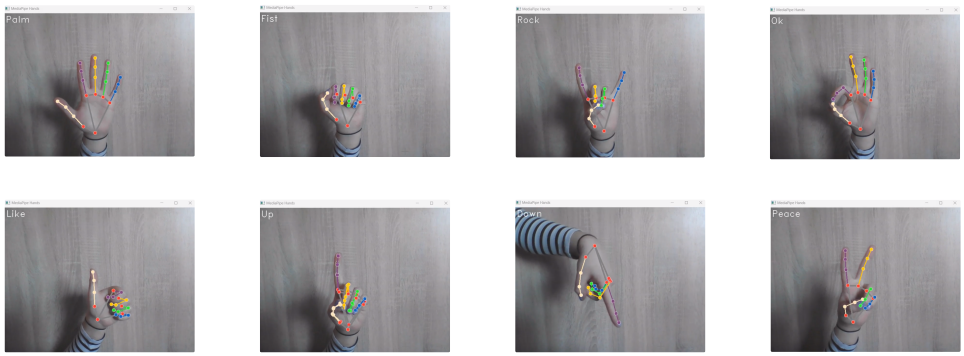

Figure 2.1: Gestures used in the project

hand landmark points in the detected hand region using regression. It is trained on approximately 30,000 manually annotated real-world images and is very well-trained and robust, allowing it to detect and map hand landmark points accurately, even on partially visible hands in most cases.

2.1.2 Creating Data

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 2.2 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.

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 2.2: A selection of the saved csv file

2.1.3 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 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 2.3.

2.1.4 Model

Training the Model

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

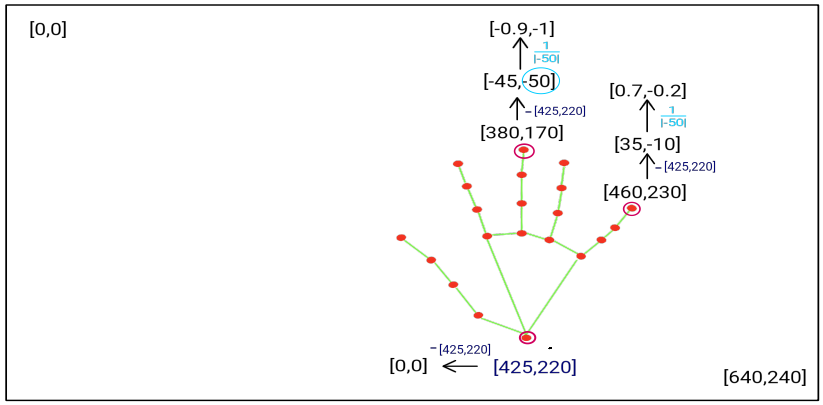

Figure 2.3: Process of normalization of landmark coordinates

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 2.4 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.

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 performance.

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

```
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 2.4: Configuration of the used feedforward neural network

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 2.5: Classification report of the model

The accuracy matrix calculates the number of correctly predicted labels by the model from the entire dataset (Eq.2.1). 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 2.2 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 2.3.

The F1 score is calculated by combining both precision and recall, as shown in Equation 2.4. 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 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 2.5.

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

$$\text{precision} = \frac{TP}{TP + FP} \tag{2.2}$$

$$\text{recall} = \frac{TP}{TP + FN} \tag{2.3}$$

$$\text{F1 score} = \frac{2PR}{P + R} \tag{2.4}$$

In the equations (2.1), (2.2), (2.3), and (2.4), 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

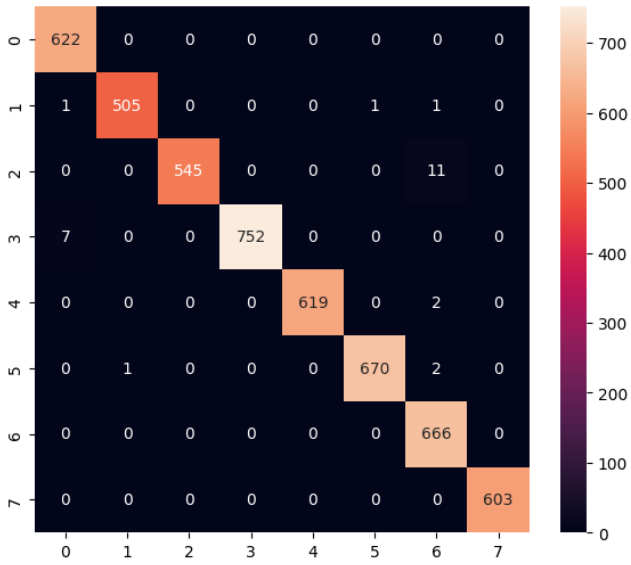

Figure 2.6: Confusion matrix of the model

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 2.6. 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 2.5:

$$\text{recall}(2) = \frac{545}{545 + 11} \tag{2.5}$$

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.

2.1.5 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 2.1 are displayed commands used in the project. We assigned them to the labels of our gestures. The gestures we use are shown in Fig 2.1

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 - Peace	Takes a picture (save in png)
4 - Ok	Take-off or land (depending on whether in flight or landed)
5 - Like	Rotate 360°
6 - Up	Move up (ascending velocity)
7 - Down	Move down (descending velocity)

Table 2.1: 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 3

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.

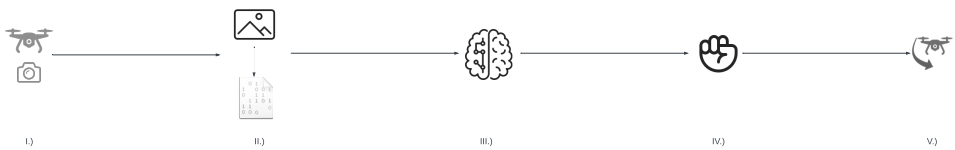

Figure 3.1: Steps of the process

Bibliography

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

[2] 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.

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

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

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

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

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