energies

MDPI

Article

Gesture recognition based on deep learning for UAV flight control

Volodymyr Samotyy [1,2], Nikita Kiselov [3] and Ulyana Dzelendzyak [2]

[1] Department of Automatic Control and Information Technology, Faculty of Electrical and Computer Engineering, Cracow University of Technology, 31155 Cracow, Poland, vsamotyy@pk.edu.pl;
[2] Department of Computerized Automatic Systems, Institute of Computer Technologies, Automation, and Metrology, Lviv Polytechnic National University, Lviv 79013, Ukraine; uliana.y.dzelendziak@lpnu.ua
[3] Paris-Saclay Faculty of Sciences, Université Paris-Saclay, Orsay 91400, France; nikita.kiselov@universite-paris-saclay.fr

Abstract: The article presents a system for controlling UAVs with gestures recognized by a model based on neural networks. A method based on a combined deep learning model is proposed that provides real-time recognition with minimal computing power consumption. An implementation with the possibility of controlling the quadrotor in two ways, namely gestures and the keyboard, is presented. The functionality of adding new gestures for recognition using interactive code in the Jupyter Lab web application is rationalized. The work of the control and recognition module is demonstrated on the example of controlling the DJI Tello Edu drone. The results of tests in real conditions are presented.

Keywords: gesture control; deep neural networks; computer vision; UAVs; Python; convolution neural network; artificial intelligence; quadcopter; TensorFlow; Tensorboard; DJI

1. Introduction

Gesture control has always been a popular research topic. Still, with the advent of neural networks in computer vision systems, the implementation of these systems in various devices has begun to grow rapidly. Thus, we are talking not only about specialized devices for people with hearing impairments but also about smartphones and other devices that have built-in cameras. Controlling UAVs using hand gestures is one such example of use that has gained popularity not only in the consumer sector but has also attracted interest from the military industry. This article presents a system based on a self-created neural network and models of the MediaPipe platform for controlling UAV gestures. The developed system combines such tasks as gesture recognition, gesture classification, processing, and execution of commands transmitted to the UAV. In practice, a model of visual gesture control of UAVs was built using modern developments in the field of artificial intelligence and the MediaPipe neural network platform from Google.

Gesture recognition is a field of computer and language technologies for interpreting human gestures using mathematical algorithms [1]. It is sometimes considered a sub-discipline of computer vision. Currently, users can use simple gestures to control and interact with devices without physically touching them. Most approaches are implemented using cameras and computer vision algorithms. Gesture control is essentially a natural interaction without any mechanical devices. The main application areas of gesture recognition are currently the automotive sector, consumer electronics, gaming, military [2, 3], home automation (IoT), and automated sign language translation.

The ability to track human hand movements and determine what gestures they can perform can be achieved using various tools. Let's consider computer vision-based models, which are based on the visual perception of an image by a computer and its subsequent interpretation. The most popular method of classical computer vision for gesture

and motion recognition is segmentation. The method of gesture recognition based on neural networks is based on the architecture of a convolution neural network and a different set of systems for its use [4, 5, 6].

The latest developments in this area include the so-called combination models. They are a combination of different types of deep neural networks. Foar example, a combination neural model based on several convolution deep neural networks and a simple multilayer neural model can be used for gesture recognition systems [7]. The convolution neural networks will be responsible for finding the hand image in the input data and identifying key points, while the conventional NN will classify these points as points in space that are specific to a particular gesture. Thus, this model can be easily adapted to be retrained for new gestures, as well as optimized for running on mobile devices. This model architecture was used in the developed system.

2. System Architecture

The development of the system architecture was based on the following main principles: fault tolerance; flexibility of component replacement (the drone can be replaced with another model or new commands can be added to the neural network); image processing does not take place on the drone. It only executes commands. Gesture control will be performed over a UAV, or rather a quadcopter, which is a subspecies of a multicopter. We used a DJI Tello Edu quadcopter, a special STEM version of a quadcopter from the Chinese company Ryze Robotics in cooperation with DJI, another Chinese com- pany that is the market leader in drones. Tello Edu's image processing is supported by the Intel Movidius Myriad processor, which allows executing instructions written in Python and additionally supports low-level image processing and recognition tools.

The most important component of the gesture management system is, of course, the gesture recognition module. It consists of two parts: a comprehensive keypoint recognition model called MediaPipe Hands and a neural network classifier that recognizes gestures based on the keypoints found. The output is the ordinal index of the recognized gesture. This index is transmitted to the drone control module, where, depending on the set parameters, the control command is transmitted to the drone.

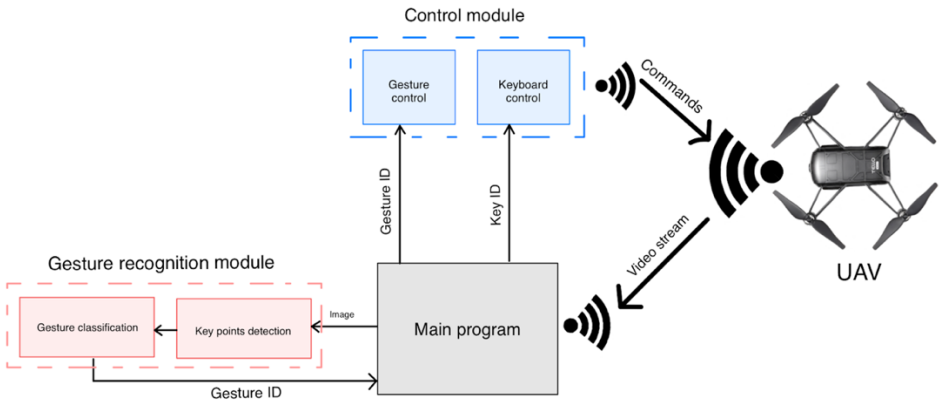


Fig. 1: Schematic representation of the system architecture

The drone can also be controlled via the computer keyboard. This part of the control module is independent of the recognition module and interacts only with the main program. The main program is a code that initializes all modules, establishes communication with the drone, and is responsible for data transfer between modules. For example, this module transmits the image from the drone's video stream to the gesture recognition

module and sends the index of the recognized gesture to the drone control module. This architecture provides several options for controlling the drone, which increases flight safety. At the same time, the modularity of the architecture allows you to change and customize drone control parameters (such as speed or command type) and add new gestures without changing the main execution program. It is enough to make the necessary changes to the solution modules.

3. Implementation

The following technologies were used to implement the UAV gesture control system:

- Python SDK djitellopy;
- a module for recognizing hand keypoints;
- the backend of the project to connect and control the drone;
- code for training a neural network written using the TensorFlow framework;
- code for NN hyperparameters optimization using Tensorboard platform.

Additionally, for graphical visualization of gesture recognition, code was written to display the image from the drone's camera in a program window using the OpenCV Python library. This library is used to visualize the results of neural networks in real time, process the image before feeding it to the neural network input, and display additional information on the video stream from the drone. The drone's battery status is displayed in the lower left corner, and the frame rate per second in the upper left. Key points of the hand are drawn on top of the image as a "white skeleton"

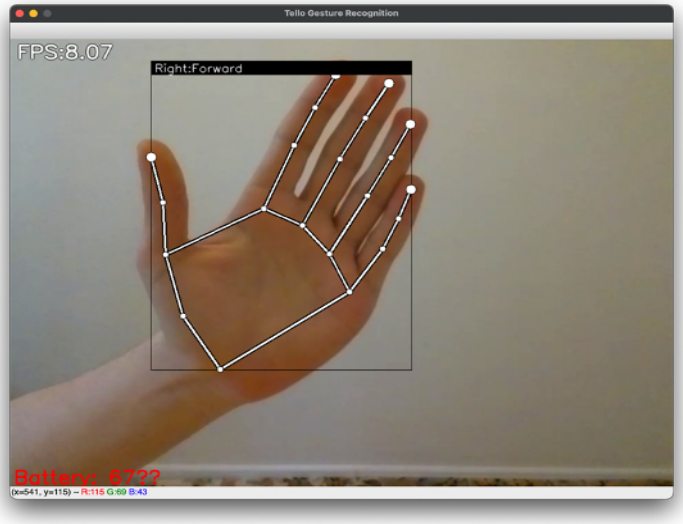


Fig. 2: Graphical visualization of gesture recognition of interconnected key points. The name of the classified gesture is displayed in the corner of the black frame that surrounds the hand in the image.

The code can be divided into three parts: the hand recognition module, gesture classifier module and drone control module. However, all components of the project have a slightly more complex organizational structure. Detailed information can be found in Appendix.

3.1. Hand recognition module

As already mentioned, the code consists of three parts . The gesture recognition module was developed first. For gesture recognition, a combined architecture is used , which

includes the MediaPipe Hands model to recognize key points of the hand and a self-developed neural network to classify gestures based on these points. Then, depending on the classified gesture, a certain command is transmitted to the quadcopter.

The solution is implemented using MediaPipe, a framework for building cross-platform machine learning solutions. The proposed model and architecture demonstrate real-time inference speed on mobile GPUs with high prediction quality. A single-pass deep learning-based detector model optimized for a real-time mobile application similar to BlazeFace, which is also available in MediaPipe, is used to detect initial hand positions. Hand detection is an extremely challenging task because the model must work on different hand sizes with a large zoom range (~20x) and be able to detect even intertwined hands. The palm detector solves these problems. The Hand Landmark model is activated after the palm has been successfully detected by the detector in the image. After running palm detection over the entire image, a subsequent hand landmark model performs precise localization of 21 keypoint coordinates in 3D within the detected hand regions using regression. The model learns a constant representation of the internal pose of the hand and is resistant even to partially visible hands and self-occlusion. The model has three outputs:

- 21 hand marks, x, y coordinates and relative depth;
- a hand flag indicating the probability of the presence of a hand in the input image;
- binary classification of hand (left or right).

To recover from a tracking failure, there is another model result, analogous to calculating the probability of an event, that detects whether a hand is actually present in the frame. If the score is below the threshold, the detector is triggered and resets the tracking. Since the MediaPipe Hands model is already ready to use, the process of connecting it in a Python script is quite simple. It is enough to import the hands class from the mediapipe library and transfer the image as a vector. At the output, we get 3D coordinates of the key points of the hand, namely 21 three-dimensional coordinates (x, y, z) of each point. Two coordinates of points (x, y) in the 2D plane will be used for gesture recognition. In order to process the results with a neural network-classifier, it is necessary to transform the data into a vector and normalize the data, which will allow the coordinates to be one range and thereby speed up model training and increase accuracy.

The step-by-step process of converting and processing the results into a vector looks like this:

- point coordinates are converted from absolute to relative. That is, instead of indicating the coordinates within the entire image, they will determine the position of the hand relative to the hand. The reference point is the base point with 0 index (coordinates (0,00));
- the list of arrays of points is combined into one consecutive array with a length of 42 elements;
- this concatenated array is normalized by the maximum number, according to the formula:

$$z = \frac{x - min(x)}{[max(x) - min(x)]}$$

At the output, we get a 1x42 array , which is ready for use by a classifier module.

(Landmark coordinates)

ID：0	ID：1	ID：2	ID：3	……	ID：17	ID：18	ID：19	ID：20
[551, 465]	[485, 428]	[439, 362]	[408, 307]	……	[633, 315]	[668, 261]	[687, 225]	[702, 188]

(Convert to relative coordinates from ID:0)

ID：0	ID：1	ID：2	ID：3	……	ID：17	ID：18	ID：19	ID：20
[0, 0]	[-66, -37]	[-112, -103]	[-143, -158]	……	[82, -150]	[117, -204]	[136, -240]	[151, -277]

(Flatten to a one-dimensional array)

ID：0		ID：1		ID：2		ID：3		……	ID：17		ID：18		ID：19		ID：20	
0	0	-66	-37	-112	-103	-143	-158	……	82	-150	117	-204	136	-240	151	-277

(Normalized to the maximum value(absolute value))

ID：0		ID：1		ID：2		ID：3		……	ID：17		ID：18		ID：19		ID：20	
0	0	-0.24	-0.13	-0.4	-0.37	-0.52	-0.57	……	0.296	-0.54	0.422	-0.74	0.491	-0.87	0.545	-1

Fig. 3: A step-by-step processing process based on the example of real data

3.2. Gesture classifier module

The model of the gesture classifier is a multiperceptron with four fully connected layers , the input of which is supplied with a pre-processed vector of key points. A multi-layer perceptron (MLP) is a class of artificial neural network (ANN). MLP consists of at least three layers of nodes: an input layer, a hidden layer, and an output layer. Three of the four hidden layers have the activation function ReLU, and the last one is Softmax. To implement this neural network , the Tensorflow library was used together with the Keras application software tool. Structure report is shown on Fig.4. Below is the mathematical definition of the utilised Neural Network:

Model: Sequential

Input: $x \in R^{42}$

Layer 1: Dropout

Dropout rate: $p_1 = 0.5$

$h_1 = \text{Dropout}(x, p_1)$

Layer 2: Dense

$W_2 \in R^{42\times32}$

$b_2 \in R^{32}$

$h_2 = \sigma(W_2 h_1 + b_2)$

Layer 3: Dropout

Dropout rate: $p_2 = 0.5$

$h_3 = \text{Dropout}(h_2, p_2)$

Layer 4: Dense

$W_3 \in R^{32\times32}$

$b_3 \in R^{32}$

$h_4 = \sigma(W_3 h_3 + b_3)$

Layer 5: Dropout

Dropout rate: $p_3 = 0.5$

$h_5 = \text{Dropout}(h_4, p_3)$

Layer 6: Dense

$W_4 \in R^{32\times16}$

$b_4 \in R^{16}$

$h_6 = \sigma(W_4 h_5 + b_4)$

Layer 7: Dense

$W_5 \in R^{16\times8}$

$b_5 \in R^{8}$

Output: $y = \sigma(W_5 h_6 + b_5)$

Here, σ denotes the activation function (ReLU). ReLU activation functions promote faster training and mitigate the vanishing gradient problem due to their linear, non-linear

nature and computational simplicity. The Rectified Linear Unit (ReLU) activation function can be defined mathematically

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Dropout is a regularization technique that randomly sets some elements of the input to zero during training with probability p_i . Dropout is an effective regularization technique that helps prevent overfitting and improves generalization in deep neural networks by randomly dropping units during training, thereby encouraging more robust feature learning.

$$\text{Dropout}(x, p) = \frac{1}{1-p}\,\mathcal{M}(x) \odot x$$

In this equation, x is the input vector, p is the dropout probability, $\odot$ denotes element-wise multiplication, and $M(x)$ is a binary mask vector generated by sampling each element independently from a Bernoulli distribution with probability p. The scaling factor of $\frac{1}{1-p}$ is applied to maintain the expected value of the input during training.

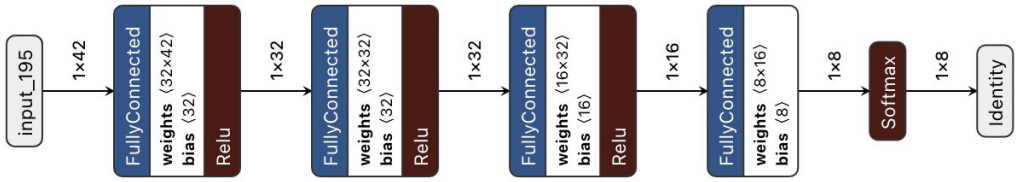


Fig. 4: Classifier model structure

The Adam optimizer and cross-entropy loss function are used in the developed model . More than 1,500 pairs of data for 8 classes of gestures were collected to train the model. The Google Colab platform was used for training. For greater accuracy, code was created using the Tensorboard data visualization platform to select the best hyperparameters for model training. In our case, it helps determine the most optimal model parameters to use.

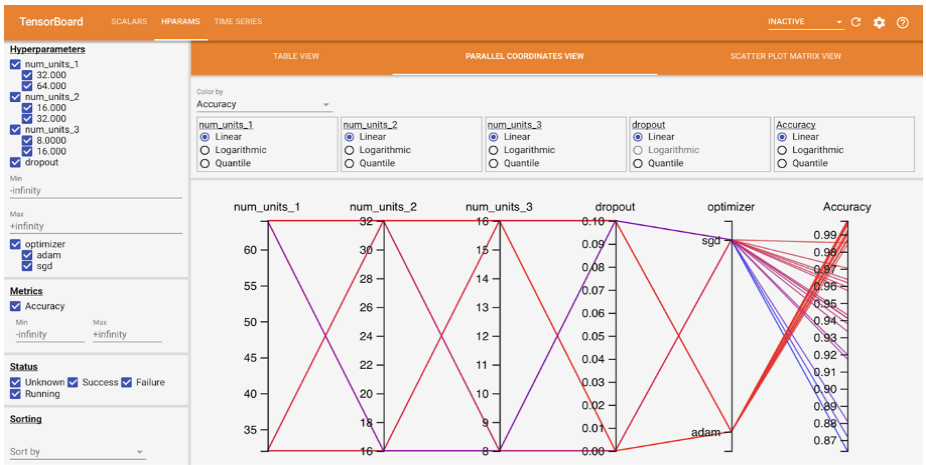
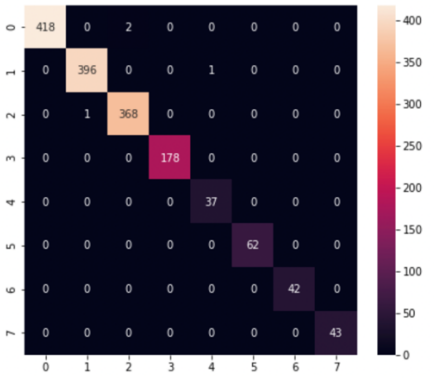


Fig. 5: View of the Tensorboar interface and the results of finding the most optimal hyperparameters

On Fig.6 can be seen that the accuracy of the model on the test data set (this is 30% of all data) is greater than 97% (precison >97%) for any class. Due to the simple structure of the model, it is possible to obtain high accuracy, having a small number of examples for training each class . We don't need to retrain the model for each gesture with different lighting, because MediaPipe takes care of all the keypoint detection.



Classification Report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	420
1	1.00	1.00	1.00	397
2	0.99	1.00	1.00	369
3	1.00	1.00	1.00	178
4	0.97	1.00	0.99	37
5	1.00	1.00	1.00	62
6	1.00	1.00	1.00	42
7	1.00	1.00	1.00	43
accuracy			1.00	1548
macro avg	1.00	1.00	1.00	1548
weighted avg	1.00	1.00	1.00	1548

Fig. 6: Matrix of inconsistencies (left) and algorithm accuracy assessment results (right)

3.3. UAV control module

After the gesture recognition part is ready, the next step is the implementation of the drone control system based on the recognized gestures , as well as receiving the image from the drone's camera. Because, as a UAV , we used the DJI Tello Edu quadcopter, which has an open SDK for its programming and control, and the djitellopy library was used for the convenience and speed of development.

Thus, this library is an ideal tool for our tasks, because the most difficult part of working with a drone is getting the image from the camera itself. Dj Tello is controlled from a computer and phone via the WiFi protocol (IEEE 802) at a frequency of 2.4 Ghz . The Tello SDK connects to the UAV via a UDP Wi-Fi port , allowing users to control the UAV using text commands. Streaming applications often use UDP because dropping packets is better than waiting for packets to be delayed due to retransmissions, which is not possible in a real-time system. That is why this protocol is used to control the drone and transmit the image. Due to the fact that the communication with the drone at a distance may not be stable, and the acquisition of real-time data is critical to the control of the drone, the use of this protocol is completely reasonable.

djitellopy library takes care of all the work with the protocol, receiving and sending data. After the successful execution of the program in the output console, we will receive the status that the connection is established and the data stream from the camera is received. After the image is processed by the gesture recognition module, the class of the gesture, or its absence, is transmitted as an index. For each gesture and corresponding index, a certain command of the drone is set. Commands and corresponding gestures are shown on the Fig.7

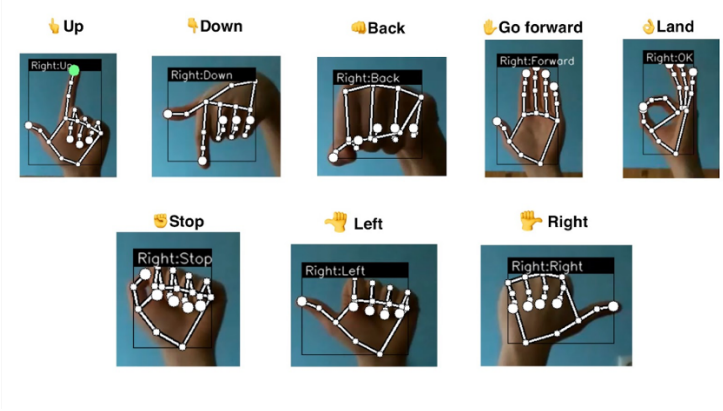


Fig. 7: Gestures and corresponding UAV control commands

Since there are possible noises and other problems when transmitting the image, the commands are recorded in a special buffer. If the buffer is filled with commands of the same type , the speed of the drone is set in the given direction. This makes it possible to increase fault tolerance , as well as to make the movements of the drone smoother. Due to the fact that gesture recognition occurs in real time, the buffer does not create problems with latency , because it fills up very quickly.

3.4. Adding new gestures

The last stage of implementation is the addition of the functionality of recording new gestures. Since the solution itself is modular, to add a new gesture you only need to retrain the neural network - the gesture classifier. To simplify data collection, a special mode was implemented that delivers create a dataset for a new gesture directly from the quadcopter's camera.

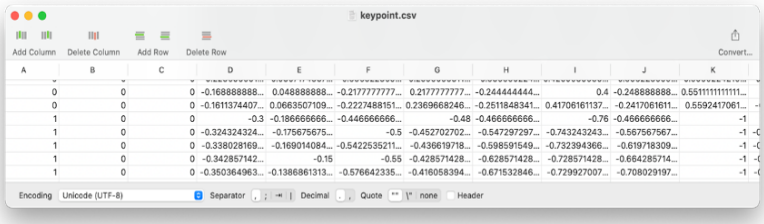


Fig. 8: File with key points in the form of a normalized vector

In this mode, when pressing a number key from "0" to "9", the key points that were recognized by the MediaPipe model Hands are recorded in a tabular data file from the indexes according to the pressed key. The coordinates of the points are already recorded in the pre-processed form of a normalized vector . In addition, in this mode, additional data can be collected to improve the recognition of already insinuating gesture An interactive code in Jupyter format was created to retrain the neural network-classifier on new data. More information is provided in the Appendix.

4. Results analysis

After the implementation of all components, the system was tested and demonstrated its full functionality in UAV flight control. The recognition module performed gesture classification with ultra-high accuracy both for the initially programmed gestures and

after training on new ones. The key point recognition model on the Mediapipe platform showed fast performance even on a weak laptop with a dual-core Intel i5 processor with integrated graphics. It is worth noting here that until recently, such models were only possible to run on a multi-core PC with discrete graphics. That is, the recognition module has fully justified the chosen architecture. The classifier neural network was quickly retrained to add new gestures. On average, it was enough to collect 30-70 examples to get accurate results for recognizing a new gesture.

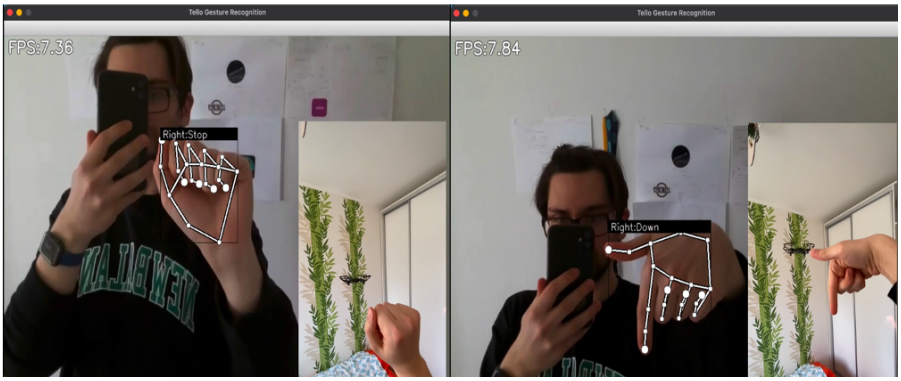


Fig. 9: Demonstration of the UAV gesture control. In the lower right corner, you can see the overlaid video from the smartphone camera recording the drone's flight.

The control module did its job properly - the drone smoothly executed commands and its flight was stable. Additionally, we tested indoor emergencies (such as a collision with a wall or loss of visual contact). In this case, the drone quickly recognized the "STOP" command or quickly switched to the keyboard control mode, which allowed it to avoid emergencies.

The main potential drawbacks that could be improved are that the recognition system does not work satisfactorily in very dark lighting or at a large distance from the hand. The first situation can be solved by using infrared cameras instead of conventional ones (such cameras can work with the light emitted by the heat of a person's palms). Since the silhouette of the hand will not change, the keypoint detection model will be sufficiently retrained on the new data without changing the architecture.

The problem of long distance is not so acute, but it can also be solved by using the Holistic model. This model allows you to recognize first the human body (key points of the skeleton), and then the key points of the hand on enlarged images from the area where the hands are located on the skeleton. This approach theoretically allows recognizing gestures at a long distance, when the image of the hand does not occupy most of the frame.

Adding the recognition of moving gestures may also be a future direction in the development. For example, writing a letter in the air with a finger or other moving hand gestures. Since movements are an integral part of human cognitive perception, the ability to recognize such gestures would add more intuitiveness and convenience to the control system. For this purpose, it is possible to use a neural network classifier based on the short-term memory (LSTM) architecture. The input is the stored history of key points for the last n frames, and the output is the gesture class. Since the buffer will store only a vector of point coordinates, the memory consumption will be negligible. Another disadvantage is the recognition of gestures of only one hand. Since the hand tracking model can work with multiple hands in the frame and recognize each one, multi-gestures based on two hands could be added in the future. This would give more control for potentially difficult situations in flight.

5. Conclusions

A gesture recognition system for UAV control was developed using the MediaPipe neural network platform. The software for gesture recognition with high accuracy using artificial intelligence and the subsequent use of classified gesture commands to control UAVs was developed. The software was based on platforms and tools from Google and DJI. This software has the following technical characteristics: real-time gesture recognition and control of UAVs, modularity of the solution for easy modification of recognition and control modules, the ability to add your own gestures for recognition, the ability to control the quadcopter in two ways (gestures and keyboard), a graphical interface for visualizing the gesture recognition process and displaying additional useful information. Since there is a rapid trend toward the development of artificial intelligence in the field of computer vision and autonomous vehicles, the developed software can serve as a basis for future gesture control programs for quadcopters both in the user area and in the field of industrial UAVs. The potential of this program in the military sector is also considerable.

Author Contributions: Conceptualization, N.K.; Formal analysis, V.S.; Investigation, U.D. and V.S.; Methodology, N.K. and U.D.; Visualization, N.K.; Software, N.K.; Writing—original draft, N.K. and U.D.; Writing—review and editing, V.S. and U.D.

All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Jhonatan Kobylarz, Jordan J. Bird, Diego R. Faria, Eduardo Parents Ribeiro, and Aniko Ek ´ art, "Thumbs ´ thumbs up down: non-verbal human-robot interaction through real-time emg classification via inductive and supervised transductive transfer learning," Journal of Ambient Intelligence and Humanized Computing, Mar 2020.
2. An Intelligent, K Verma, A Singh, K Singh, and V Singh, "An intelligent human - unmanned aerial vehicle interaction approach in real time based he machine learning using wearable gloves," Journal of Intelligent & Robotic Systems, vol. 102, no. 1, pp. 1–12, 2021.
3. PatSeer, "Patent landscape report hand gesture recognition patseer pro," Retrieved 2017-11-02, 2017.
4. V. Pavlovic, R. Sharma, and T. Huang, "Visual hand gesture interpretation for human-computer interaction: A review," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 7, pp. 677–695, July 1997.
5. V.G. Abakumov, O.Yu. Lomakina, and O.B. Yarovenko, "The use of hand gestures in the human-machine interface," Electronics and communication, vol. 16, no. 6, pp. 86–91, 2011.
6. M.Z. Islam, MS Hossain, R. ul Islam, and K. Andersson, "Static hand gesture recognition using convolutional neural network with data augmentation," in 2019 Joint 8th International Conference he Informatics, Electronics Vision (ICIEV) and 2019 3rd International Conference he Imaging, Vision Pattern Recognition (icIVPR), 2019, pp. 324–329.
7. H. Lahiani, M. Kherallah, and M. Neji, "Hand pos estimation system based he viola-jones algorithm for android devices," in 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA), 2016, pp. 1–6.
8. F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, CL Chang, and M. Grundmann, "Mediapipe hands: On-device real-time hand tracking," arXiv preprint arXiv:2006.10214, 2020.
9. Thomas G. Zimmerman, Jaron Lanier, Chuck Blanchard, Steve Bryson, and Young Harvill, "A hand gesture interface device," 1995.
10. Yang Liu and Yunde Jia, "A robust hand tracking and gesture recognition method for wearable visual interfaces and its applications," in Proceedings of the Third International Conference he Image and Graphics (ICIG'04), 2004.
11. Kue-Bum Lee, Jung-Hyun Kim, and Kwang-Seok Hong, "An implementation of multi-modal game interface based he pdas," in Fifth International Conference he Software Engineering Research, Management and Applications, 2007.
12. V. Pallotta, P. Bruegger, and B. Hirsbrunner, "Kinetic user interfaces: Physical embodied interaction with mobile pervasive computing systems," in Advances in Ubiquitous Computing: Future Paradigms and Directions. IGI Publishing, Feb 2008.
13. Gestigon, "Gestigon gesture tracking - techcrunch disrupt," Retrieved 11 October 2016, 2016.

14. C. Manresa, "Hand tracking and gesture recognition for human-computer interaction," Electronic Letters he Computer Vision and Image Analysis, vol. 5, no. 3, pp. 96–104, 2005.