SMART GLOVE AND HAND GESTURE-BASED CONTROL INTERFACE FOR MULTI-ROTOR AERIAL VEHICLES

KIANOUSH HARATIANNEJADI

A THESIS

IN

THE DEPARTMENT

OF

ELECTRICAL AND COMPUTER ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF APPLIED SCIENCE (ELECTRICAL AND

COMPUTER ENGINEERING)

AT

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

DECEMBER 2020

Abstract

Smart Glove and Hand Gesture-Based Control Interface for Multi-Rotor Aerial Vehicles

Kianoush Haratiannejadi

This thesis introduces two types of adaptable human–computer interaction methods in single-subject and multi-subject unsupervised environments.

In a single-subject environment, a single shot multi-box detector (SSD) is used to detect the hand's regions of interest (RoI) in the input frame. The RoI detected by the SSD model is fed to a convolutional neural network (CNN) to classify the right-hand gesture. In a crowded environment, a region-based convolutional neural network (R-CNN) detects subjects in the frame and the RoI of their faces. These regions of interest are then fed to a facial recognition module to find the main subject in the frame. After the main subject is found, the R-CNN model feeds the right-hand RoI of the main subject to the CNN to classify the right-hand gesture. In both single-subject and multi-subject environments, a fixed number of gestures is used to describe specific commands from the right-hand gestures to the vehicle (take off, land, hover, etc.).

A smart glove on the left hand is used for more precise vehicular control. A motion-processing unit (MPU) and a set of four flex sensors are used in the smart glove to produce discrete and continuous signals based on the bending value of each finger and the roll angle of the left hand. The generated discrete signals with the flex sensors are fed to a support vector machine (SVM) to classify the left-hand gesture, and the generated continuous signals with the flex sensors and the MPU module are used to determine some characteristics of the gesture command such as throttle and angle of the vehicle.

Three simultaneous validation layers have been implemented, including (1) Human-Based Validation, where the main subject can reject the robot's behavior by sending a command through the smart glove, (2) Classification Validation, where the main subject can retrain the classifier classes at will, and (3) System Validation, which is responsible for finding the error's source when the main subject performs a specific command through the smart glove.

The proposed algorithm is applied to groups consisting of one, two, three, and four subjects including the main subject to validate and investigate the behavior of the algorithm in various desirable and undesirable situations. The proposed algorithm is implemented on an Nvidia Jetson AGX Xavier GPU.

Acknowledgments

I would like to thank my supervisor Prof. Rastko R. Selmic, for his dedicated support and guidance. Prof. Selmic continuously provided encouragement and was always willing and enthusiastic to assist in any way he could throughout the research project.

I cannot forget to thank my family for all the unconditional support in these very intense academic years and send me endless messages of encouragement throughout the process.

I would also like to thank my great friend Mr. Kiarash Ariyankia for always being there for me and support me in any circumstances. Finally, many thanks to all participants that took part in the study and enabled this research to be possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Reaserch Topic

This thesis proposes a method for capturing and classifying the right- and left-hand gestures of a main subject and transforming them into commands that control a robot such as an unmanned aerial vehicle (UAV). This method uses various neural networks, including a convolutional neural network (CNN), a region-based convolutional neural network (R-CNN), a single shot multi-box detector (SSD), and a support vector machine (SVM). A new self-validating interface was developed for controlling multi-rotor aerial vehicles in single-subject and multi-subject environments. In recent years, multi-rotor aerial vehicles have used in various applications such as monitoring, mapping, person-following [1] and transportation [2].

For a single-subject environment, the proposed method uses right-hand gesture recognition, where a hand in each image frame is detected using an SSD, and a CNN model then performs a classification. For a multi-subject environment, the proposed method first uses an R-CNN to detect subjects in the input frame, and a facial recognition module to find the subject of interest. Then, the R-CNN model detects the main subject's right-hand region, and a CNN model classifies the gestures of the detected right hand into one of the predefined classes. The proposed method differs in the process according to whether the environment is single- or multi-subject.

Some aspects of the proposed method in this thesis are the same for both environments. In single- or multi-subject environments, the left-hand controls the aerial vehicle, which occurs simultaneously with the right-hand control, uses a smart glove with four flex sensors and a motion-processing unit (MPU). These sensors feed data into an SVM for left-hand gesture recognition. An advantage of this combined interface is that wearable devices, such as smart gloves, do not depend on environmental conditions such as lighting or the user's physical features. The gesture recognition interface should be sufficiently robust and adaptable to be able to handle various hand shapes and sizes.

1.2 Motivation

Gesture recognition system recognizes human gestures and classifies them such that they can be used as control commands. In modern gesture recognition systems, human gestures are transmitted via special gadgets (such as gloves) equipped with various sensors or read by a camera. The captured data are then interpreted as commands and used as inputs to control applications or robotic devices.

Currently, various techniques involving wearable gadgets and gesture recognition have been used to advance methods of human–robot interaction [3, 4, 5, 6]. Applications of these techniques include a wearable gesture recognition glove that controls robotic devices in disaster management [7] and wearable motion sensors that drive a virtual car [8]. Other examples include robotic devices that maintain a clean environment [9] and wearable motion sensors for the impaired hearing persons [10].

Hand gesture recognition systems can be categorized into glove-based and vision-based systems. A glove-based gesture recognition system consists of a glove and sensors that captures data. The glove extracts the configuration and motion of the user's hand. Authors in [10] translated a sign language gestures into text, where their method consists of a series of accelerometers on each finger of a glove and other sensors on the shoulders that send electrical signals to a microcontroller. The microcontroller then interprets the action associated with the gesture. In addition to the glove-based part of the proposed method where enables the system to recognize the hand gestures using wearable sensors, the proposed method also has a vision-based part which uses visual inputs such as images from a camera to extract the features to be used in gesture recognition.

1.3 Contribution

We propose an adaptable, closed-loop system that can recognize and classify right-hand and left-hand gestures simultaneously in both single- and multi-subject environments. The flex sensors and the MPU on the smart glove enable the user to produce continuous-time control commands (e.g., the throttle of a drone or its roll angle). We demonstrate how the proposed method classifies gestures and how it handles various undesirable situations.

The proposed approach has the following novelties and advantages in comparison with existing methods in [3, 4, 5, 6, 7, 10, 11]:

(1) The proposed smart glove produces 16 different gestures (commands) including two control signals that are used to control of an unmanned vehicle.

(2) A hand recognition module and background removal method are used before gesture classification to improve classifier performance.

(3) Detection of the main subject's right-hand region and classification of the main subject's right-hand gestures have been implemented.

(4) The system is adaptable to new users due to online learning.

(5) Self-validation of the classifier is proposed which improves the reliability.

(6) System validation and a human-in-the-loop validation increase the overall system reliability.

Furthermore, the smart glove, with 16 different commands is appended to the image processing system (creates four commands) to provide more control commands in comparison to gesture classification alone. For instance, angle variation in turn right and turn left commands is possible by changing the flex sensors' bending value on the left hand fingers. Similarly, increasing and decreasing the altitude of the drone is feasible by altering the angle of the smart glove. The control precision of this combined method outperforms the smart glove and image processing techniques on their own.

Chapter 2

Background and Related Works

2.1 Introduction

This chapter presents an introduction and an overview of the utilized neural networks such as CNN and R-CNN and classification methods such as SSD and SVM in this thesis. Additionally, recent related works on gesture recognition with cameras and wearable gadgets are discussed and compared with the proposed method.

2.2 Fundamentals

In the proposed algorithm, we use a series of neural networks for various purposes. For instance, an SSD model is used to detect a hand in a single-subject environment, and an R-CNN model is used to find the subjects in each input frame. A brief introduction to each model is presented in the following subsections.

2.2.1 Single Shot Multibox Detector

An SSD is an object detection model with a two-component structure [12]. The model takes an image as an input and categorizes the various objects in this image into different classes. Figure 2.1 shows that for image classification, the model applies a base network,

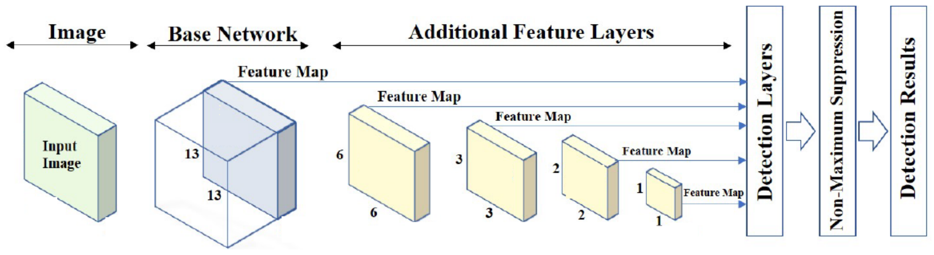


Figure 2.1: Single shot multi-box model architecture.

including a standard architecture, while for object detection, it uses a structure of additional feature layers. This auxiliary structure contains convolutional feature layers sorted by size in descending order, placed at the end of the base network to perform object detection at different scales.

The proposed method in this thesis uses the SSD object detection model for hand tracking in the single-subject scenario [13]. Subsequently, a non-maximum suppression (NMS) algorithm [14] is used to merge all the detected regions related to similar hand gestures. The NMS output is used as an input to the Hand Gesture Classification module. Object detection is performed at different scales and uses boxes at various aspect ratios similar to the faster R-CNN [12]. Since each convolutional layer operates at a different scale, the model can detect objects of various sizes. This model does not require region proposals (based on sets of rectangles) for the convolution layers. Instead, it uses various bounding boxes and then adjusts each bounding box as part of a prediction in a single network, which accelerates calculations.

2.2.2 Convolutional Neural Network

The CNN model consists of several layers of artificial neural networks (ANNs) stacked in a cascading structure [15]. For the CNN hidden layers we choose three layers of max-pooling, two layers of dropout, five rectified activation functions in different layers, one sigmoid activation function, one flattening layer, and two fully connected layers.

In the convolutional layer, the CNN extracts the feature maps from the input image using feature detectors. When feature maps are connected, the convolutional layer is formed.

Since the convolutional operation is linear, the output of the convolution is passed through an activation function to make the output nonlinear, and a rectified linear unit (ReLU) and a sigmoid function are used as activation functions to enhance the nonlinearity of the feature maps.

Pooling Layer:

The purpose of pooling layers is to decrease the feature maps' resolution and summarize the input distortions in a smaller feature map. We use max-pooling method in the CNN design to extract the highlights from the input frame. In max-pooling, we select the largest element within each feature map such that [16]:

$$y_{i,j} = \underset{(p,q)\in R_{i,j}}{max}\ x_{p,q}\ , \tag{1}$$

where $x_{p,q}$ is defined as

$$x_{p,q} = \sum_{i=1}^{m} \omega_i x_{p,q_i}\ , \tag{2}$$

in (1), $y_{i,j}$ is the output of the pooling function associated with the feature map region $R_{i,j}$, and $x_{p,q}$ is the input element at location (p, q) which embodies an area around the position (i, j) in the input frame. In (2), ω_i is the i-th weight associated with x_{p,q_i} which is i-th element of $x_{p,q}$, and m is the number of element in input $x_{p,q}$.

By using a max-pooling technique, unnecessary information is removed from the generated feature maps in the convolutional layer and reduces the map's dimensions. The maximum value in each sub-region of the feature maps is stored, and the remaining values are deleted as unnecessary information. The pooled feature maps are generated to form the max-pooling layer.

Dropout:

The dropout technique is used to prevent overfitting during the training process. The dropout technique is applied only during the training time and at each iteration, which temporarily disables a neuron with a probability of p. All the inputs and outputs to the disabled neuron are frozen at the current iteration. The dropped-out neurons are tested with probability p at every iteration of training, so a dropped-out neuron at one step can be activated at the next iteration [17].

The dropout-rate p is corresponding to $p\%$ of the neurons being dropped-out during the whole training process. The dropout controls the participation of each neuron x_j with an independent Bernoulli random variable α_j for each step [18]:

$$y_i = \frac{1}{p}\sum_{j=1}^{N}\omega_{j,i}(\alpha_j \cdot x_j),\ \alpha_j \sim B(1,p), \tag{3}$$

where $B(1,p)$ is the Bernoulli function which generates the independent random variable α_j with probability p. The scaling factor $\frac{1}{p}$ scales the output y_i to keep the expected value of the output during the training process. The weight from input x_j to input x_i is denoted by $\omega_{j,i}$, and N is the number of neurons connected to the neuron x_i.

Activation Function:

Activation functions are used to determine the neural network's output in classification problems. Furthermore, an activation function limits the amplitude of the output of a neuron and maps the output values between 0 to 1 or -1 to 1 depending upon the function's type [17, 19].

Comparison in [19] between most commonly used activation functions reveals that the suitable activation functions for training an image processing-based algorithm with the proposed architecture in this thesis are ReLU and sigmoid activation functions. Moreover, the training process requires fewer iterations with ReLU and sigmoid activation function than the other activation functions to solve nonlinear problems. Additionally, we find that our CNN's accuracy is 5% higher with these two functions than with other activation functions such as radial basis function and hyperbolic tangent function.

ReLU has output 0 if the input is less than 0, and has the same value as the input if the input is greater than 0:

$$f(x) = max(x, 0), \tag{4}$$

where $f(x)$ is the output of the function and x is the input neuron.

Sigmoid functions are used in machine learning for logistic regression and neural network implementations. Sigmoid function and its derivative reduces the time required for training and building the model [19]. The sigmoid activation function is mostly used in neural networks trained to classify the input into classes.

We use the sigmoid function as it transforms the neuron's output $g(x_i)$ to values between 0 and 1, and is defined as:

$$g(x_i) = \frac{1}{1 + e^{-x_i}}, \tag{5}$$

where input neuron x_i is defined as:

$$x_i = \sum_{j=1}^{N} \omega_j x_{ij}, \tag{6}$$

and ω_j denotes the weight associated to the j-th element of input x_i and N is the number of element in input x_i. Here, we apply the sigmoid function to our CNN design to obtain each hand-gesture class's probability.

Optimizer:

The main purpose of using optimizer is to reduce the losses, increase the learning rate, and provide the most accurate results [20]. Comparisons [20] and [21] between adaptive moment estimation (Adam), gradient descent, adagrad, and adadelta show that the suitable optimizer based on the proposed neural network structure is Adam optimizer because of its algorithm speed, computation capacity and obtained results with the proposed algorithm. The weight update law for Adam optimizer [22] is given by

$$\omega_{t+1} = \omega_t - \frac{\eta \cdot \hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}, \tag{7}$$

where ω_t and ω_{t+1} are the current model weight and the next iteration model weight, respectively. The learning rate (step size) in Adam equation is denoted by η, and to avoid division by zero, a small positive term ε is added in the denominator. The bias correction of the moving averages $\widehat{m}_t$ and $\widehat{v}_t$ in the Adam optimizer are as follows:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t}, \end{aligned} \tag{8}$$

where m_t and v_t are moving averages as follows:

$$\begin{aligned} m_t &= (1-\beta_1)g_t^2 + \beta_1 m_{t-1}, \\ v_t &= (1-\beta_2)g_t^2 + \beta_2 v_{t-1}, \end{aligned} \tag{9}$$

and g_t is gradient on current mini-batch, and β_1 and β_2 are the exponential decay rates.

Flattening Layer:

The flattening layer converts two-dimensional max-pooling outputs to a one-dimensional array. Finally, the fully connected layer is created by connecting each neuron to all neurons in the previous layer to classify the extracted features.

The number of layers in the CNN model in the proposed method is shown in Table 2.1, and the CNN model structure is illustrated in Figure 2.2.

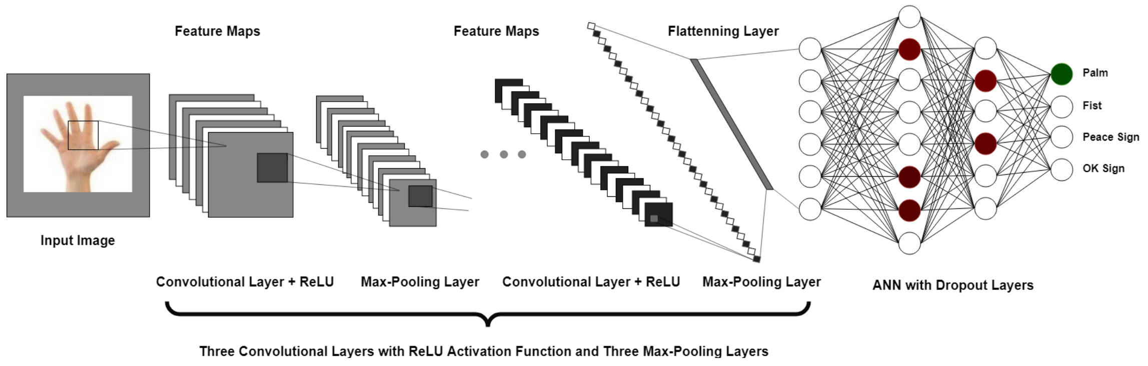

Feature Maps
Feature Maps
Flattenning Layer
Input Image
Palm
Fist
Peace Sign
OK Sign
Convolutional Layer + ReLU
Max-Pooling Layer
Convolutional Layer + ReLU
Max-Pooling Layer
ANN with Dropout Layers
Three Convolutional Layers with ReLU Activation Function and Three Max-Pooling Layers

Figure 2.2: Convolutional neural network model architecture in the gesture recognition model.

Table 2.1: Number of layers used to design the CNN model in a single-subject environment [13].

Layers Level	Layers	Number of Layers
1	Convolutional Layer	3
2	Max-pooling Layer	3
3	Flattening Layer	1
4	Fully Connected Layer	2
5	Rectifier Activation Function	5
6	Dropout Layer	2
7	Sigmoid Activation Function	1

2.2.3 Regional Convolutional Neural Network

An R-CNN, consists of two steps [23]. In the first step, it uses selective search, which extracts nearly $2,000$ region suggestions from the input image into boxes. Hence, instead of attempting to classify a very large number of regions, the network can work with only $2,000$ regions. In the second step, it extracts all the features from those $2,000$ regions independently using a large CNN for classification. The R-CNN classifies each region by feeding data from the last layer into SVMs for each suggested box in the input image.

2.2.4 Support Vector Machine

Support vector machine (SVM) is an universal learning machine, which is applied to both regression [24] and pattern recognition [25]. An SVM uses a kernel to map the data from input space to a high-dimensional feature space in which the problem becomes linearly separable [26]. The decision function of an SVM is related not only to the number of support vectors and their weights but also to the a priorly chosen kernel that is called the support vector kernel [24]. There are three advantages of using SVM model in the classification problems: (i) conversion of the classification problem from a complex classification problem to the computation of a linear decision function, (ii) absence of local minimum in the SVM optimization problem, and (iii) a sparse solution which causes a computationally efficient decision function.

SVM uses Cover's theorem, which guarantees that any data set becomes arbitrarily separable as the data dimension grows, and a class of functions called kernels is used in SVM to achieve this goal. Many kinds of kernels can be used, such as the gaussian kernel or gaussian radial basis function (RBF) when there is no prior knowledge about the data, polynomial kernel for image processing, ANOVA radial basis kernel in regression problems, and linear kernel when dealing with large sparse data vectors. In the proposed method, the smart glove's gathered data has four dimensions (four flex sensors), and there is a clear margin of separation between classes. Since the SVM model works relatively well in these two conditions, we choose the SVM as the left-hand classifier.

Since the proposed algorithm is designed to control a drone in a real-time scenario, we need a kernel in the SVM model that is fast and has a real-time response time of less than a second to generate the output and send it to the next module. Response time and training an SVM with a dataset with four dimensions with a linear kernel are faster than any other kernel. In the next chapters in this thesis, we will use the linear kernel to classify the smart glove data into pre-defined hand gestures.

2.3 Related Works

Human-robot interaction has broad applications such as gaming [27] and controlling and navigation of a quadrotor in GPS-denied environments [28]. In several previous approaches, [8, 28, 29, 30], the user must stand alone and show only the requested hand.

Moreover, the use of only right-hand gestures to interact with a drone is presented in [31]. In contrast, the proposed algorithm can distinguish and receive commands from the gestures of both hands simultaneously, and the proposed system can interact with and recognize the main subject in a multi-subject environment.

An effective human–robot interaction requires a gesture vocabulary [31, 32]. While [32] used only right-hand gestures to interact with and command waiters based on six gestures, the proposed method in this thesis uses an image processing and self-validation technique with four gestures and a smart glove with 16 distinct gestures.

Authors in [33] proposed a gesture recognition method using depth information from pixels for the human skeleton where depth information is used to segment the body into joints. In this thesis, we use the Skeleton Detection module on 2D images to determine all the users presented in each frame. In addition, we use the Skeleton Detection module to detect the heads of the subjects and their pairs of hands.

While [34] used an initial keypoint detector to produce noisy labels in multiple views of the hand to detect parts of the hand, we use an initial keypoint detector to detect the joints in the body. Unlike [34], which used multiple cameras and a controlled environment, the proposed method in this thesis uses one camera, and it does not require a controlled environment.

The advantage of the approach in [33] and the proposed approach in this thesis in comparison to [29] and [35] is that the user does not require to stand while performing the gesture, thus the proposed approach is suitable for real-time implementation.

We have combined two approaches to human–robot interaction: a smart glove [31] and an image processing classification method [36]. The proposed system in this thesis recognizes the gestures from the smart glove using an SVM. The SVM uses data from flex sensors and an MPU and simultaneously classifies right-hand gestures in each frame with three validation modules to validate and improve the performance of the Hand Gesture Classification module. The SSD model is used for hand detection due to its simple training process, suitable speed, and accuracy, while the CNN model is used for gesture recognition. For the multi-subject environment, we add one more module to the algorithm that detects all present subjects in each frame and labels their bodies to identify their heads and hands. Afterward, a facial recognition model identifies the main subject and then an R-CNN determines the main subject's pair of hands. Finally, the CNN classification module classifies the user's right- or left-hand gestures (in the proposed implementation, we classify the right-hand gestures).

The R-CNN for object detection consists of several steps [37]. The category-independent region proposals are extracted from the captured input images. The fixed-length features are then calculated and obtained from region proposals using a large CNN model. Next, an R-CNN classifies each feature by feeding the data from the last layer into an SVM.

In [11], a Haar feature-based AdaBoost classifier has been used to categorize five different gestures for controlling a drone. While the accuracy of their framework is highest when the drone's operator posed within a distance of 3 ft, which is the sensing range of the camera used in the study, the proposed method works at its highest accuracy when the operator performs a gesture within the camera's sensing range, which is 8.5 ft.

Wearable gadgets that measure body movements are part of a rapidly growing field. The data collected from hand movements and gestures can be converted into voices or commands. For instance, those with speech impairments have difficulty communicating in a society in which most people do not understand sign language, similar to the glove in [38], the proposed glove converts sign language gestures into predefined commands. Their glove maps sign language to speech output by recognizing the gesture made by the hand. The proposed smart glove in this thesis uses exact parameter values to determine a command instead of a threshold to determine a gesture.

There are several gadgets that provide human–machine interaction, such as joysticks, buttons, keypads, keyboards, and gloves. The most significant advantages of smart gloves are their compatibility with natural human movement and their smooth fluctuation with three degrees of freedom. Therefore, using finger curvature and hand angles from smart gloves is an elegant method for human–machine interaction in controlling robot movements. Authors in [39] made a connection between wearable gadgets and non-wearable methods, such as joysticks, trackballs, mouse position, and mouse velocity, in random applications. The results show that compared to non-wearable gadgets, the robotic glove is more consistent with natural hand movements, has better interaction, and is closer to what the user expects from the behavior of a robot.

Furthermore, Table 2.2 provides a comparison between the proposed method and similar methods based on specific criteria such as the number of gestures, classifier accuracy, validation modules, ability to increase the number of gestures, adaptability with a new user, type of classifiers, and the source of interaction between the user and the system. As shown in Table 2.2, while no other method includes validations and the user must be alone in front of the camera, in our method there are three layers of validation and the method allows for multi-subjects where the algorithm can detect the main subject.

Table 2.2: Comparison between the proposed method and similar approaches.

Other Approaches / Criteria	Method [3]	Method [4]	Method [5]	Method [6]	Proposed Method
Number of Gestures	13 Continuous Commands	7 Continuous Commands	9 Continuous Commands	13 Continuous Commands	20 Continuous and Discrete Commands
Increasing the Gestures	✓	✗	✓	✗	✓
Classifier	P-CNN	No Classifier Just Filtering	CNN	MLP	Right Hand: SSD + CNN + R-CNN Left Hand: SVM
Classifier Accuracy	91.9%	82%	81.5%	92%	94.28%
Adaptability with New Users	✓	✗	✗	✗	✓
Real-Time Processing	✓	✓	✓	✓	✓
Output Validation	✗	✗	✗	✗	✓
Commands Sources	Camera	Camera	Wearable IMU and MMG Sensors	Wearable Accelerometer and Gyroscope Sensors	Camera and Wearable MPU and Flex Sensors on a Glove
Independent of the Surrounding Environment	✓	✗	✓	✓	✓

Chapter 3

System Components

3.1 Introduction

The programming languages C++ and Python are used to program the proposed method, and CoppeliaSim software is used as the simulation environment based on real-world characteristics. Furthermore, for hardware, we use an Arduino UNO as the smart-glove processing unit. We also use flex sensors and an MPU 6050 as sensors that measure hand fluctuations. We use a Blue tooth module as the signal transmitter between the smart glove and the main processor in the robot, which is an NVIDIA graphics processing unit (GPU). This chapter presents an overview of the system components that we use in this project.

In the Chapter 4, the structure of the smart glove, the hand recognition module, and the validation modules are explained.

3.2 System Components

3.2.1 Arduino

To recognize left-hand gestures with a wearable gadget, smart glove, we require a programmable MEMS such as an Arduino UNO (Figure 3.1) which contains analog pins that read the flex sensor signals. Additionally, the Arduino UNO microprocessor is Bluetooth compatible, that is used for wireless transmission of data and commands from the smart glove to the main processor, an NVIDIA GPU, in the robot.

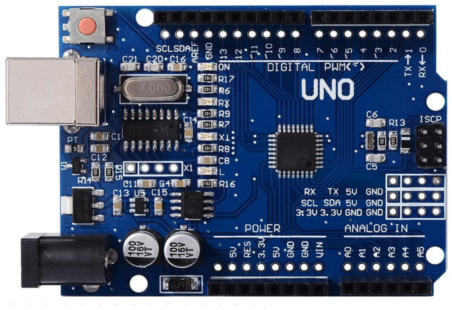


Figure 3.1: Arduino UNO board.

Arduino UNO is an open-source electronic board which is empowered by easy-to-use hardware and software. The programming language for Arduino UNO is C++, and we program the Arduino to calibrate the flex sensors, read the flex sensors and MPU signals, and transmit the data to the main processor. This model of Arduino has an ATmega 328P microprocessor, which has 14 digital input/output pins, six analog inputs, a $16MHz$ quartz crystal, and a reset button.

3.2.2 Flex Sensors

The tool that we use to measure the finger bending value of the left hand consists of four flex sensors. Flex sensors produce various voltages when they bend, and we can measure the curvature in every finger by collecting the signals produced from this bending. Figure 3.2 shows that the flex sensors which are used in this thesis have two pins. In the proposed circuit, as shown in Figures 3.3 and 3.4, the positive pins are connected to each other and then connected to one of Arduino analog pins, and the negative pins are connected to the Arduino analog pins separately. In the proposed implementation, four flex sensors are implemented on the glove to measure the bending value of four fingers: the thumb, the index, the middle, and the ring fingers.

To shift the output voltage of the flex sensors to a range that the Arduino can manage with minimum fault and without damaging the board, we use a voltage divider, which is

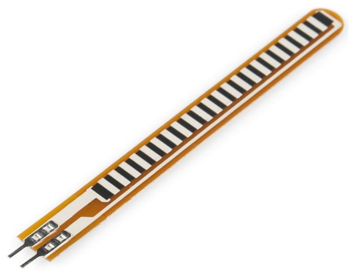
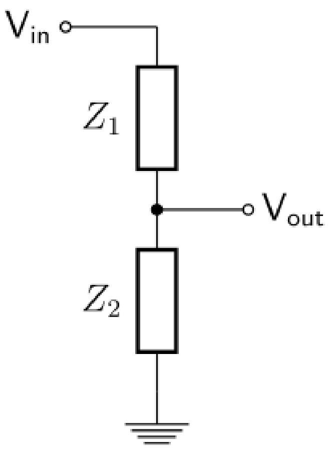
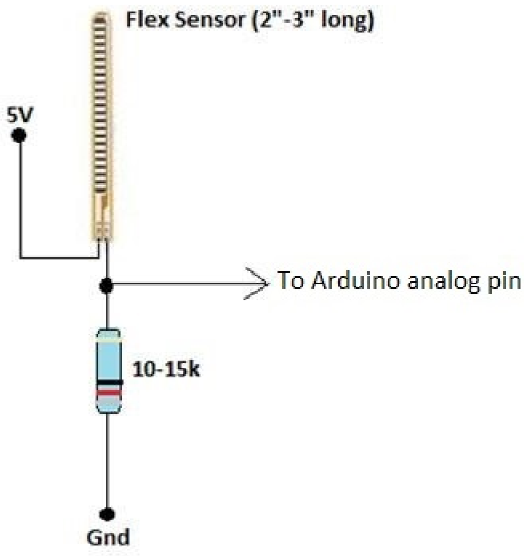


Figure 3.2: Flex sensor with two pins for producing voltage on them.

illustrated in Figure 3.3a.



(a) Voltage divider circuit.



(b) Circuit of a flex sensor connection.

Figure 3.3: Flex sensors calibration diagram.

The voltage divider and the flex sensor are connected as shown in Figure 3.3b. In the proposed smart glove, all the flex sensors' positive pins are connected to one of the Arduino 5 volt pins. Furthermore, the rest of the analog pins of the flex sensors are coupled with $10K\Omega$ resistors. For all the flex sensors, the other resistor pins are connected to the Arduino ground (GND) pin. Figure 3.4 shows that the Arduino analog pins are connected to the common connection between the flex sensors and resistors, and the other flex sensor pins are connected to the Arduino 5 volt pin.

The flex sensors are calibrated through the Arduino program. Since these sensors are

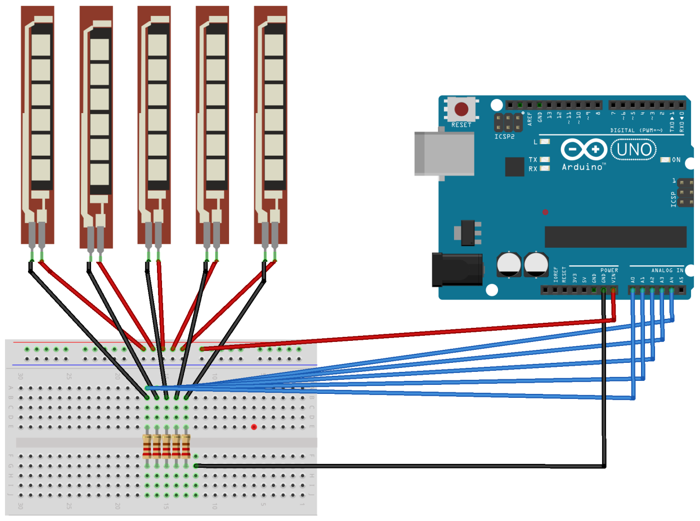


Figure 3.4: Connection between all flex sensors and Arduino pins.

used to measure the bend in every finger of the left hand to recognize its gesture, the maximum value is set when the fingers are bent, and the minimum value is set when the fingers are completely straight.

3.2.3 Arduino Integrated Development Environment

The Arduino integrated development environment (IDE) makes it easy to write C++ code and upload it to the board. The code is written in two parts. The first part is run once at the beginning of the process and calibrates the flex sensors and the MPU. The second part is run after the first part and continues running for the rest of the process.

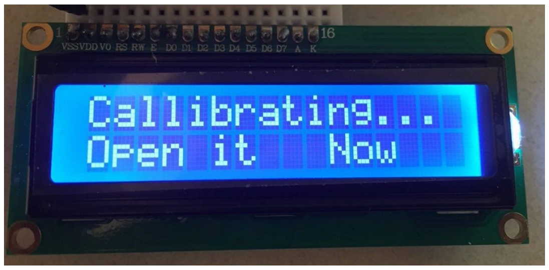
I) **First Part, Calibration**

- *Flex sensors calibration:* At the beginning of the process, this part of the program calibrates the flex sensors in two separate steps. Figures 3.5c and 3.5d show that the first step is opening the fingers to set the minimum values to zero, and the second step is forming a fist with the fingers to set the maximum value of each sensor signal to 100%.
- *MPU calibration:* To define the the direction of the user's hand and to establish

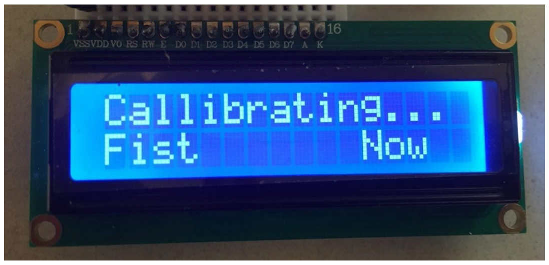
the zero angles of the hand, the user must hold the hand steady and perpendicular to the body and parallel to the ground.

(a) Calibrating start.

(b) Motion-processing unit calibration.



(c) Flex sensor calibration: setting the zeros.

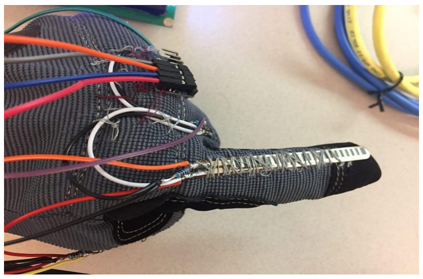


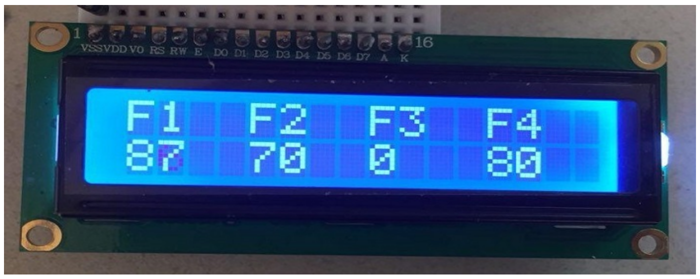
(d) Flex sensor calibration: fixing the maximums.

Figure 3.5: Calibration process: MPU module and the flex sensors.

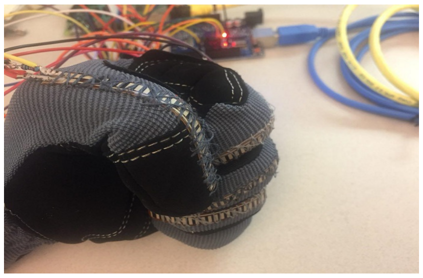
II) **Second Part, Reading and Transmitting the Generated Signals**

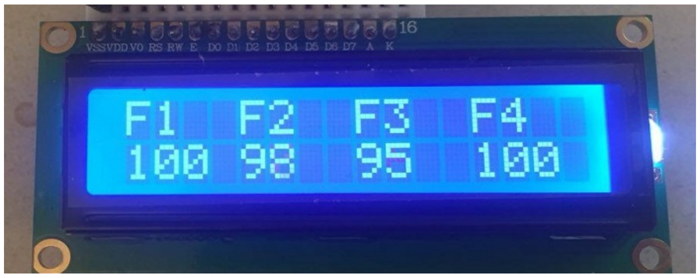
- *Reading the flex sensor signals:* After the calibration step, with any bend in the fingers, the flex sensors begin generating signals. Subsequently, the Arduino reads the produced signals through its analog inputs and converts them into the calculated minimum and maximum ranges as shown in Figure 3.6.
- *Reading the MPU values:* With any fluctuation in the left hand at various angles, the MPU produces various signals, and the Arduino reads them through its analog pins, passes them through a filter, and then transmits them.
- *Transmitting data:* After reading the signals produced through the analog pins, the Arduino transmits the data through a Bluetooth module to the main processor.


(a) Pointing gesture.



(b) Sensors output values for pointing gesture.


(c) Fist gesture.



(d) Sensors output values for fist gesture.

Figure 3.6: Gestures with the smart glove. Bending values for four flex sensors on the glove are shown as F1, F2, F3, and F4 for four fingers of the left hand.

3.2.4 MPU 6050

The MPU 6050 sensor (Figure 3.7) contains a micro-electromechanical system accelerometer and a MEMS gyro in a single board. It is highly accurate, as it contains 16-bit analog-to-digital conversion hardware for each channel. Therefore, it captures the *x*-, *y*-, and *z*- axis simultaneously. The sensor also uses a I2C-bus to interface with the Arduino UNO.

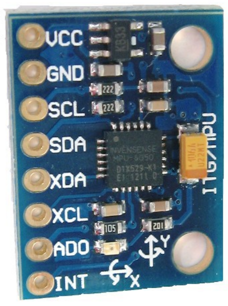


Figure 3.7: MPU 6050.

This sensor helps to measure the angle of the left hand in the *x*-, *y*-, and *z*- axis. By using the MPU pins which are SDA, SCL, VCC, and GND pins, we can read the MPU signals. The signals thought these pins are analog, so we connect them to the Arduino analog pins as well as the flex sensors.

3.2.5 Bluetooth Module

We use a bluetooth module as the data transmitter to transmit the collected data from the flex sensors and the MPU to the main processor, which is located in the robot, Figure 3.8.

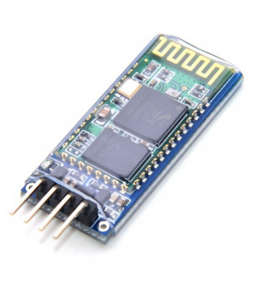


Figure 3.8: Arduino UNO Bluetooth module.

Additionally, the smart glove can be extended for use with both hands. We can implement all the equipment and circuits individually for each hand and then transfer data from one hand to another through a Bluetooth module and from the second hand, we can transmit the collected data from both hands to the main processor.

The deployed smart glove's equipment are demonstrated in Figure 3.9.

3.2.6 Webcam

This thesis proposes an adaptable human–robot interface that uses two types of human–computer interactions, one of which is an image processing technique for right-hand gesture recognition. The image processing technique requires a webcam as its input device to capture the frames.

The cameras that we use for this project are the AUSDOM AW335 Full HD1080p webcam (Figure 3.10a) for single-subject scenarios and a LOGI HD 1080p webcam (Figure

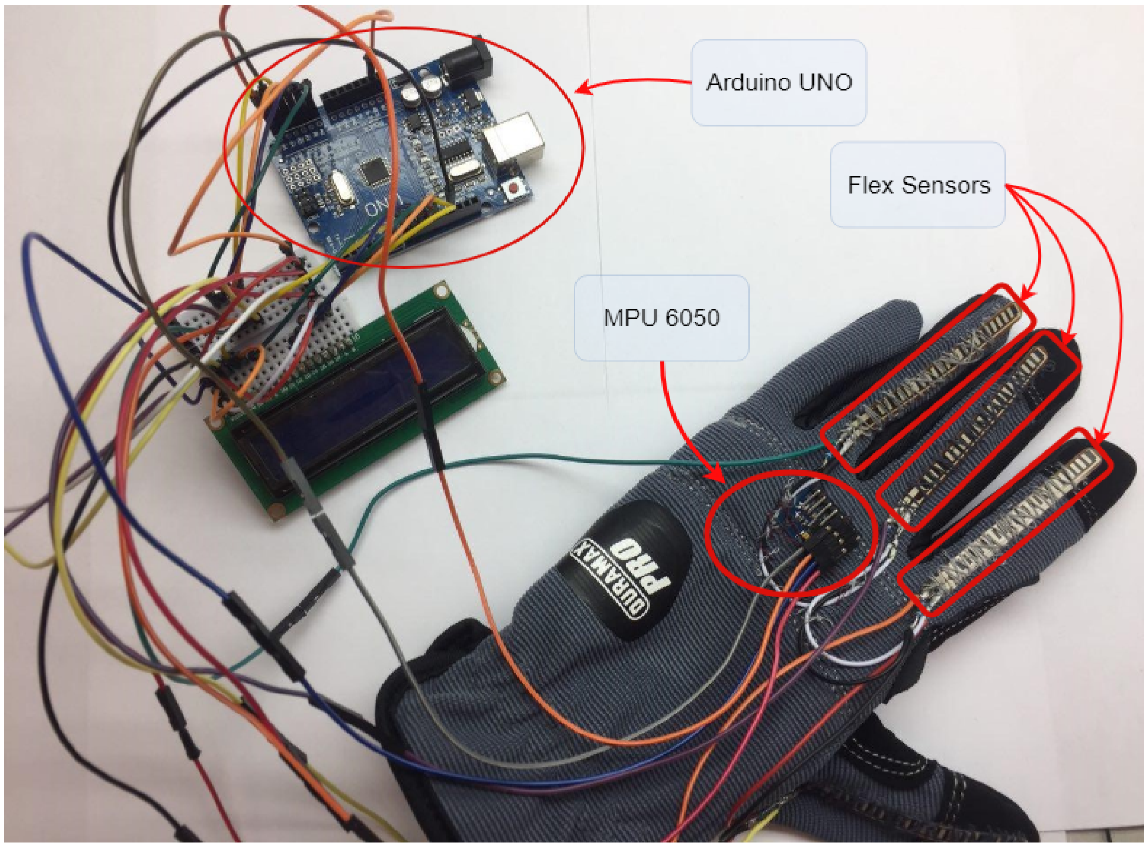


Figure 3.9: Smart glove.

3.10b) for multi-subject environments. The input frames are taken through these webcams and sent to the neural networks to capture a hand and its gestures.

(a) AUSDOM AW335 Full HD1080p webcam.

(b) LOGI HD 1080p webcam.

Figure 3.10: Cameras used in the project.

3.2.7 NVIDIA GPU

The proposed image-based human–robot interaction model includes an SSD network to recognize a user's hand. It also includes a CNN to classify the gestures of the detected hand in a single-subject environment and an R-CNN to detect the subjects in the input frame in a multi-user environment. Further, it has a facial recognition module to recognize the main subject and a CNN classifier to classify the main subject's right-hand gestures. Thus, a powerful GPU is required to run machine-learning (ML) and deep-learning (DL) models such as CNN and R-CNN concurrently. For this purpose, we chose the NVIDIA Jetson AGX Xavier GPU (Figure 3.11).

Figure 3.11: The main processor on the robot, the NVIDIA GPU.

This Linux-based GPU with 512 cores and 32GB memory is capable to implement ML and DL models and build an adaptable human–robot interface that uses two types of human–computer interactions concurrently. Utilization of this GPU enables the system to reach a reasonable speed in the proposed algorithm run-time without any timeout and decreasing in accuracy.

Chapter 4

System Implementation

4.1 Introduction

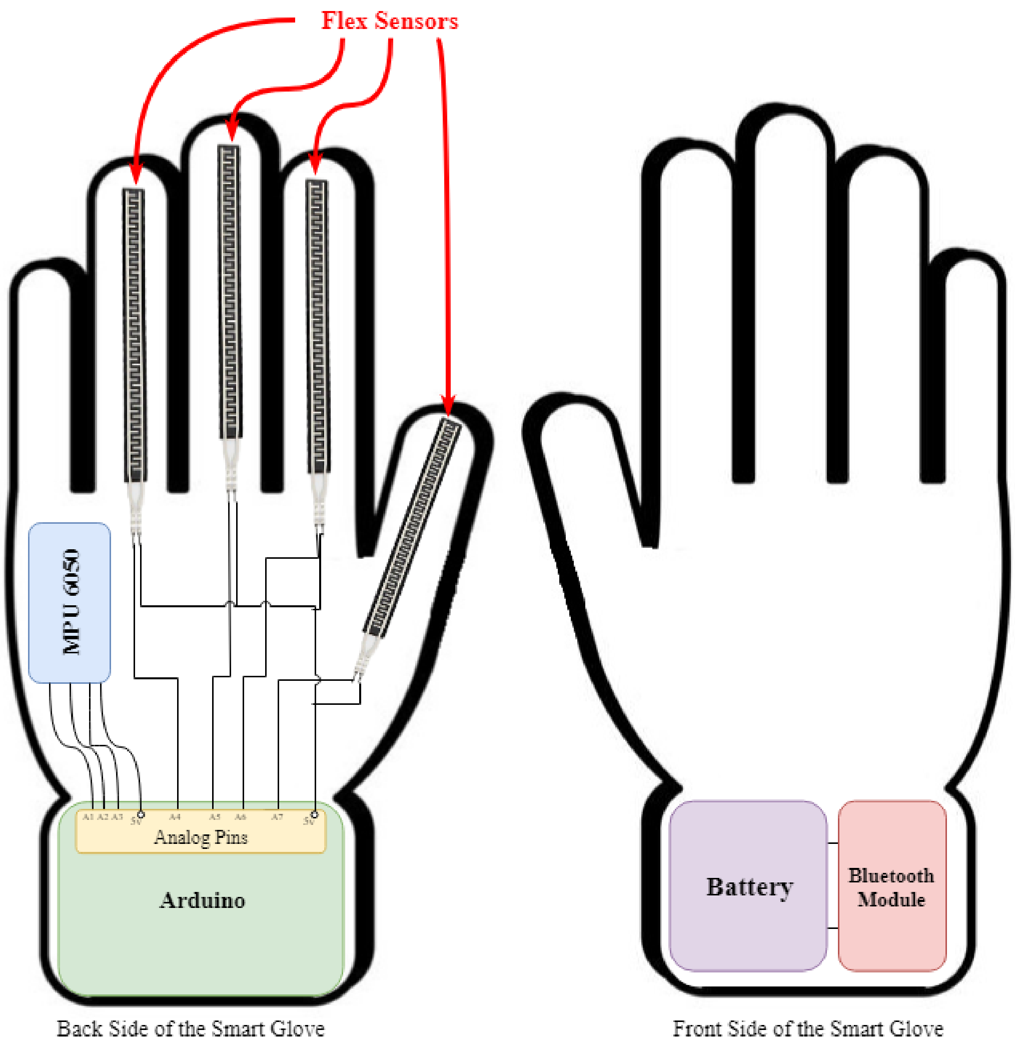
This chapter addresses the structure of the smart glove, the hand recognition module, and the validation modules. We show block diagrams for the system and the recognition sections, pseudo-code for validation process, and the configuration of the CNN.

4.2 Smart Glove

The proposed smart glove converts hand gestures into a set of predefined commands. The microprocessor gathers all the raw data to analyze them in separate units. Figure 4.1 shows the component's locations on the smart glove. The flex sensors are placed on ring, middle, index, and thumb fingers, and the MPU module is placed on the back of the glove. The Arduino is place on the wrist, and the bluetooth Module and the battery are placed on front side of the smart glove.

4.2.1 Flex Sensors

Flex sensors generate voltage signals as a function of applied strain. First, the flex sensors and the MPU must be calibrated. To calibrate the flex sensors, the fingers must be



Flex Sensors
MPU 6050
A1 A2 A3 5V A4 A5 A6 A7 5V
Analog Pins
Arduino
Battery
Bluetooth Module
Back Side of the Smart Glove
Front Side of the Smart Glove

Figure 4.1: Smart Glove designed in this project.

kept fully open, and the algorithm then assigns an output value of zero to each of the flex sensors. The fingers must then be closed, and the algorithm assigns a value of 100%. To calibrate the MPU, the user must keep the smart glove in a stable position and parallel to the ground. After reading the data, the microprocessor broadcasts the data via a transmitter to the Flight Controller and the System Validation module.

The smart glove operates in two different modes simultaneously. The first mode considers each finger as an on/off switch, and the second mode uses the exact bending values of each flex sensor.

Flex Sensors as On/Off Switches:

The smart glove supports 16 different gestures. To analyze these gestures, an SVM model with a linear kernel is used. The output of the SVM model determines whether each flex sensor is on or off and the command that four of them represent. The algorithm then uses the exact bending value of each flex sensor in the second mode of the smart glove. These commands are separated into two class types to communicate with the System Validation module and the Flight Controller, which are going to be explained in the next section.

Bending Value:

This mode determines the first coefficient for the chosen command. For instance, if Gesture $\#2$ represents turn left, the finger-bending value determines the angular velocity. The system converts each gesture into binary code. Each digit represents a flex sensor, and the SVM output determines whether each digit is on or off. For example, Gesture $\#2$, or the pointing gesture as shown in Figure 3.6(a), is coded in binary as 0010. After determining each digit, the system considers the changes of $rs_j(t)$ for the corresponding flex sensor j from the moment that flex sensor is activated t_0 (those sensors with value 1) in each sampling time t as

$$rs_j(t_n) = f_j(t_n) - f_j(t_0), \tag{10}$$

where $rs_j(t_n)$ is the change between time step t_n and t_0. The bending values for flex sensor j between time steps t_n and activation time t_0 are $f_j(t_n)$ and $f_j(t_0)$, respectively. We determine the reference velocity of the drone, $v_G(t_n)$, as

$$v_G(t_n) = \frac{rs_j(t_n)}{max\{f_j\} - min\{f_j\}} v_0 + v_0, \tag{11}$$

where v_0 is the drone's velocity before each classification and which we consider an initial velocity. Maximum and minimum bending values, $max\{f_j\}$ and $min\{f_j\}$, are calculated in the calibration step for each flex sensor. In the following sections, we show the effect of an activated flex sensor that corresponds to f_j value.

For instance, in Figure 3.6, when the user makes a pointing gesture, the microprocessor gathers the bending value of each flex sensor and then sends the data to the computation unit. An SVM model then classifies the collected data into one of the 16 possible left-hand gestures. Figure 4.2 shows the difference in value for each gesture. Descriptions of the generated gestures with the smart glove are provided in Table 4.1. Table 4.2 provides flex sensors features and specifications.

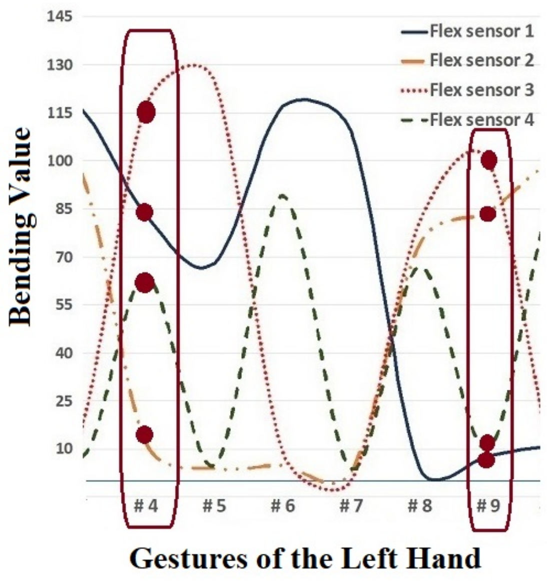


Figure 4.2: Finger's bending values of four flex sensors. As indicated on the horizontal axis, a combination of four bending values from the flex sensors represents a specific gesture.

Table 4.1: Gestures for the smart glove and their corresponding functions.

Gestures	Description
Gesture # 1	*land*
Gesture # 2	*turn left*
Gesture # 3	*turn right*
Gesture # 4	*increase the height*
Gesture # 5	*decrease the height*
Gesture # 6	*turn* 180°
Gesture # 7	*move straight to forward*
Gesture # 8	*move straight to backward*
Gesture # 9	*reserved*
Gesture # 10	*reserved*
Gesture # 11	*reserved*
Gesture # 12	*reserved*
Gesture # 13	*reserved*
Gesture # 14	*reserved*
Gesture # 15	*penalty for the output*
Gesture # 16	*hover*

Table 4.2: Flex sensors features and specifications.

Features	**Specifications**
Operating Voltage	$0 - 5V$
Operating Temperature	$-45°$ C to $+80°$ C
Flat Resistance	$25K\Omega$
Resistance Tolerance	$\pm 30\%$
Bend Resistance Range	$45K\Omega$ to $125K\Omega$

4.2.2 MPU 6050

The MPU module, with its gyroscopic sensor, divides the space into three angle ranges: $0°$ to $60°$, $60°$ to $120°$, and $120°$ to $180°$, as shown in Figure 4.3. The MPU generates different values by holding the hand in different angles along the x-axis, y-axis, and z-axis. These values are used as the second coefficients for the 16 hand gestures or as separate coefficients for a certain parameter. For instance, we can define the pitch axis (z-axis) of the hand as a coefficient of the drone altitude. The Flight Controller uses the MPU correspond value to the z-axis as a coefficient of the drone altitude unless a command is

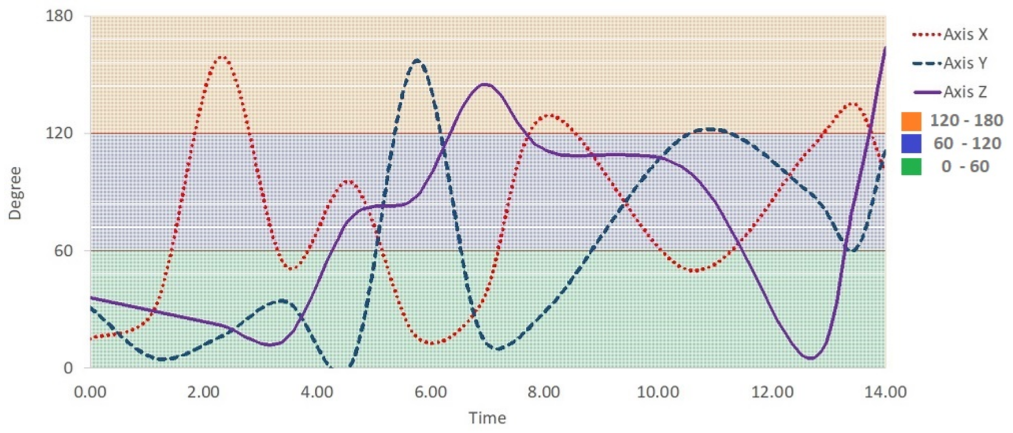


Figure 4.3: MPU response in three zones by fluctuating the smart glove.

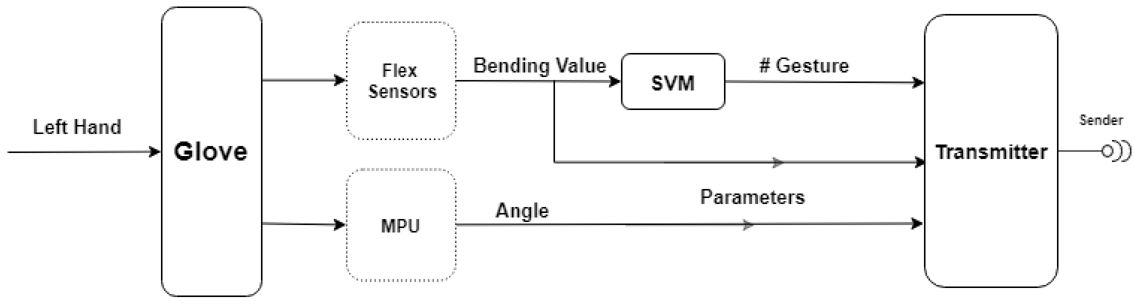
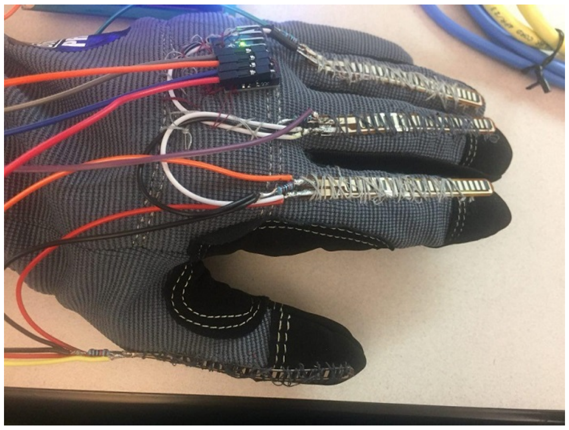


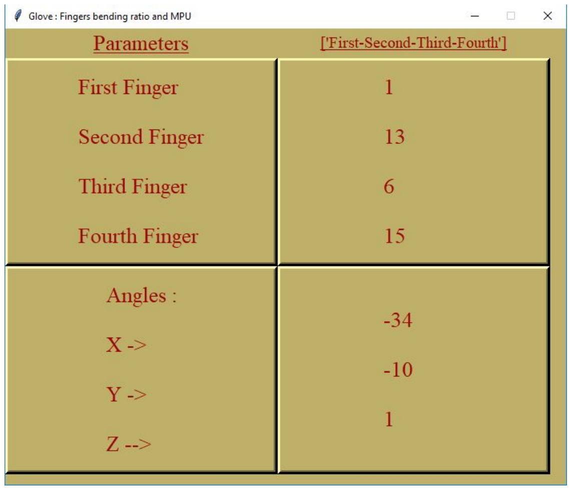
Figure 4.4: Gesture recognition module with the smart glove. This unit by using flex sensors calculates the bending value for each finger and then an SVM classifier classifies the gesture based on those bending value. An MPU module is used to calculate the angle of the left hand.

given to change the state of the drone. If the user holds the smart glove pitch angle in the third range 120° to 180°, the altitude of the drone rises. If the user holds the smart glove pitch angle in the second range 60° to 120°, the altitude does not change and if the pitch angle in the z-axis drops to the range 0° to 60°, the altitude decreases.

Figure 4.4 shows the smart glove block diagram with all of its components. Figure 4.5 shows a simple GUI that gathers all the signals produced by the flex sensors and the MPU and shifts the gathered signals into a calibrated range. The GUI contains three sections: the SVM output, the flex sensor output, and the MPU output.



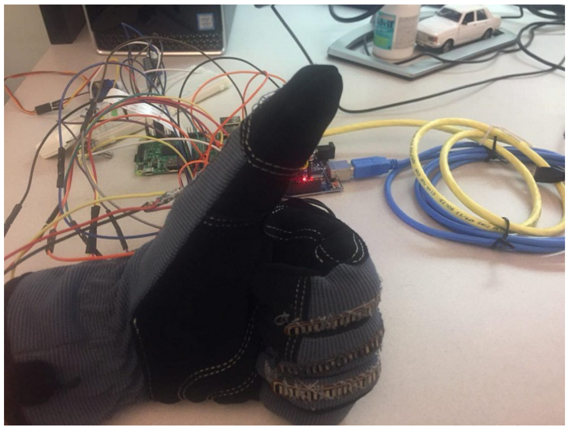
(a) Open palm gesture.



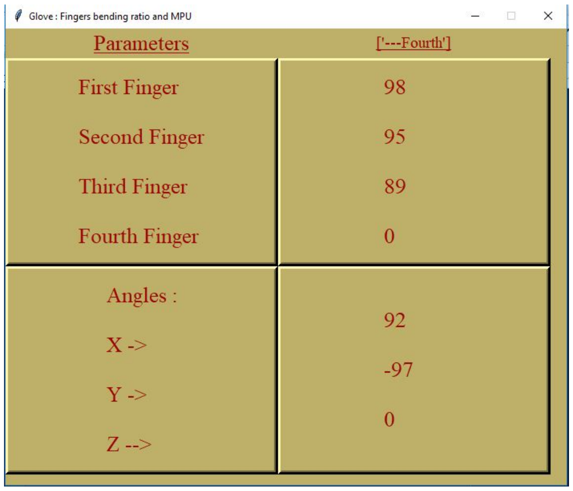
Glove : Fingers bending ratio and MPU

Parameters	['First-Second-Third-Fourth']
First Finger	1
Second Finger	13
Third Finger	6
Fourth Finger	15
Angles :	
X ->	-34
Y ->	-10
Z -->	1

(b) Outputs for open palm gesture.



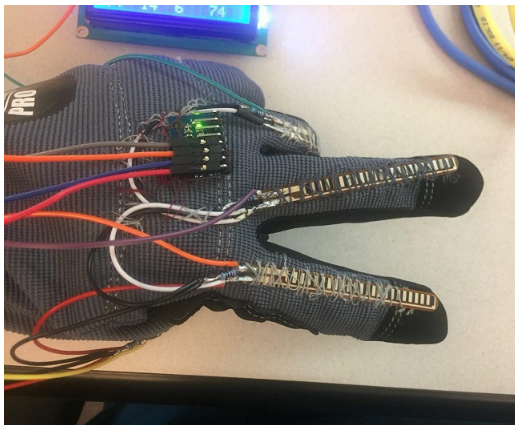
(c) Thumb up gesture.



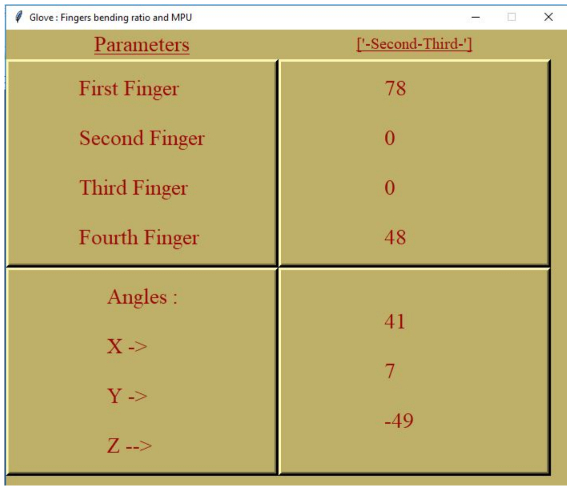
Glove : Fingers bending ratio and MPU

Parameters	['---Fourth']
First Finger	98
Second Finger	95
Third Finger	89
Fourth Finger	0
Angles :	
X ->	92
Y ->	-97
Z -->	0

(d) Outputs for thumb up gesture.



(e) Victory sign gesture.

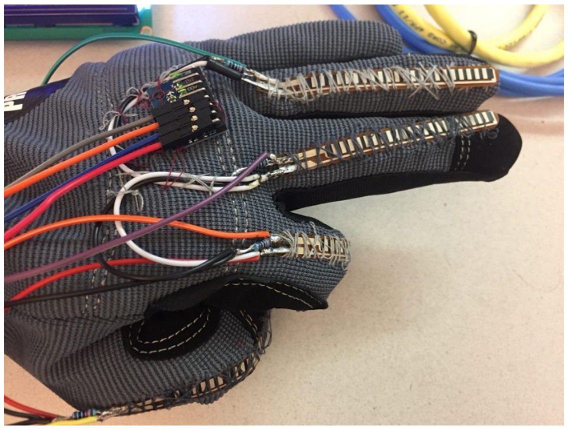


Glove : Fingers bending ratio and MPU

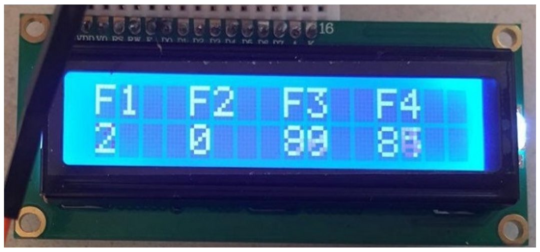
Parameters	['-Second-Third-']
First Finger	78
Second Finger	0
Third Finger	0
Fourth Finger	48
Angles :	
X ->	41
Y ->	7
Z -->	-49

(f) Outputs for Victory sign gesture.

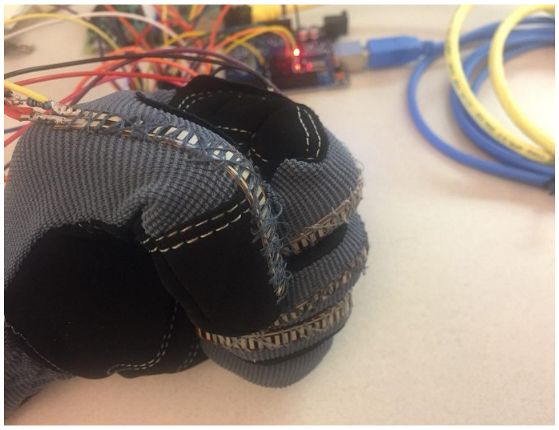
Figure 4.5: Smart glove gestures with the GUI.



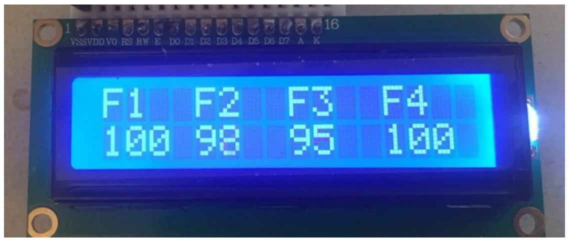
(a) First and second fingers are open.



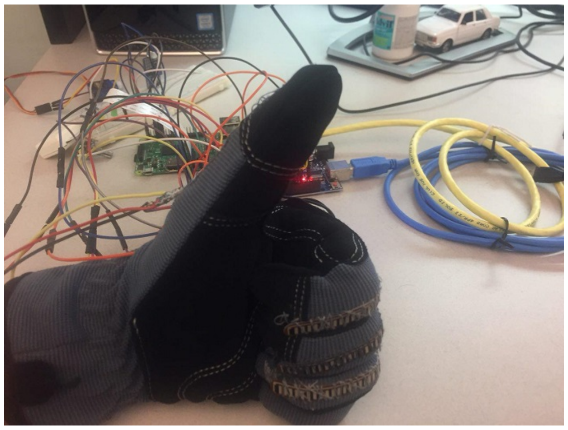
(b) Flex sensors outputs.



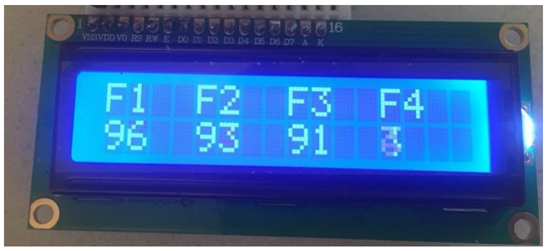
(c) Fist gesture.



(d) All flex sensors outputs are maximum.



(e) Thumbs up gesture.



(f) Thumbs up flex sensors outputs.

Figure 4.6: Smart glove gestures and their flex sensors outputs.

4.3 Hand and Gesture Recognition in a Single-Subject Environment

Block diagram in Figure 4.7 illustrates the gesture-based control system with four main units: (i) Human-in-the-Loop Validation, (ii) Hand Detection and Gesture Classification Unit, (iii) Flight Controller, (iv) and System Validation module. In the single-subject environment we use the hand detector to find the hand and the detected hand is then classified. Also, there is a System Validation module which activates a Retrainer module if the classifier has a low confidence in its results. Then, the Retrainer retrains the classifier until the classifier reaches a better confidence which can be reached even in one step.

Two types of gesture recognizers are used in this thesis to interact with an unmaned aerial vehicle: (i) a smart-glove method and (ii) an image processing method. Note that the classifier in the proposed method does not create time-series predictions; also, the proposed method in this thesis focuses mainly on static gestures (or postures) and not dynamic gestures.

The Hand Detection and Gesture Classification Unit in the single-subject case has three

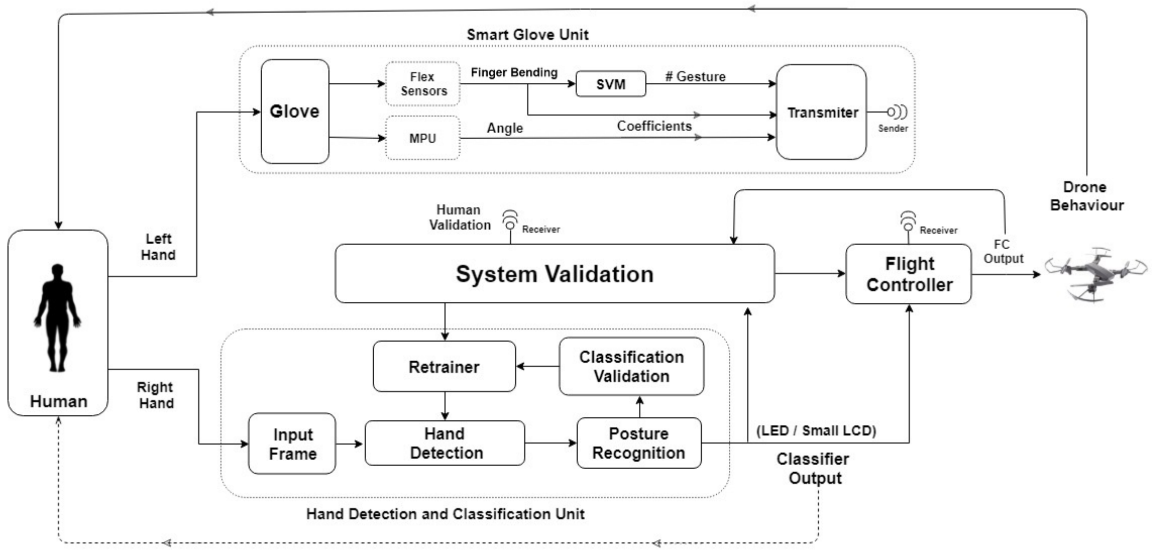


Figure 4.7: System block diagram for a single-subject environment, including three separate validation modules: (i) Classification Validation, (ii) System Validation, and (iii) human-in-the-loop validation.

main modules: (i) the Hand Detection and Classifier module, (ii) the Classification Validation module, and (iii) the Retrainer module (Figure 4.8). The primary input to this unit is a gesture frame, and the main output is the classified gesture represented as a class. The classifier has an internal validation module, which checks and improves the classifier's performance.

We performed hand detection and gesture recognition in a single-subject scenario using the AUSDOM AW335 Full HD 1080p webcam. The inputs in the first stage were images from the camera. First, we converted input images from RGB to grayscale to reduced image dimensionality and thus accelerated the process. Later, these grayscale images were fed into the Hand Detection module to find and capture the hand, if there is one, in each image.

4.3.1 Hand Detector

The first step of gesture recognition is detecting a hand in the input frame, Figure 4.8. The algorithm uses the SSD model to detect the hand and the CNN model to classify the detected hand into one of the predefined classes. The classifier output maps to one of the predetermined commands such as take off, land, turn right, or turn left. The output command is then sent to the Flight Controller, which is used to execute the command. The command is then sent to the System Validation module and the Classification Validation module.

We chose the SSD model for hand detection because of its simple training and appropriate speed and accuracy [12]. We trained the SSD model with N images of size of $n \times m$ pixels from various users who are in front of a white background showing one of the four hand gestures. After the hand is tracked, the SSD select a rectangular RoI of the detected hand. This RoI and the input frame from which it is extracted (input frame of the SSD) are flattened and sent into the next model for classification. Thus, to recognize various hand postures in each frame, the classifier does not check the entire frame; it checks only the position in the input frame that is related to the extracted RoI. This method is used to increase the accuracy and speed of the system.

4.3.2 Gesture Classifier

We select the CNN model for the proposed gesture recognition step. Compared to its predecessors, the main advantage of CNN is that it automatically detects the essential features in each sample for each class without any human supervision. Furthermore, the proposed algorithm is designed to retrain and accept new classes, in which this feature is possible with CNN's retraining and pre-training features. Another advantage of using a CNN model as the classifier is the weight sharing feature of CNN, which makes CNN more efficient in terms of memory and complexity. The CNN model consists of several layers of ANNs stacked in a cascading manner, [15]. To train the CNN model, we use an extensive database, as described in detail in Section 4.4.4. To train the classifier, we use the Keras library [40], which is a powerful library to build layers for each neural network model. By using the Keras library and CNN model, each detected RoI from the input frame is classified into one of the four available classes as shown in Figure 4.9.

The CNN model also uses a dropout technique and a sigmoid activation function to prevent overfitting [17]. The number of layers in the CNN model that we use in this thesis and the description of each layer are shown in Table 4.4, and the CNN model structure has

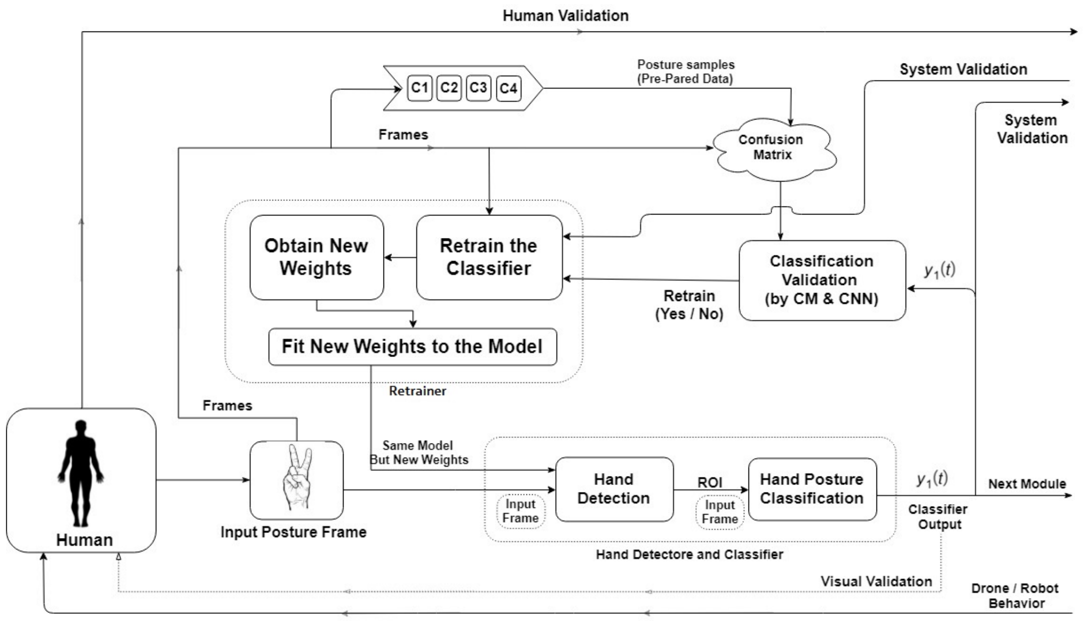
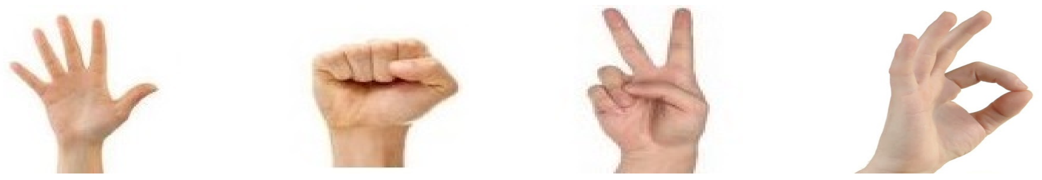


Figure 4.8: Hand Detection and Classification Unit block diagram in a single-subject environment.



(a) Palm **(b)** Fist **(c)** Victory sign **(d)** OK sign

Figure 4.9: Four specified right-hand gestures as the webcam inputs, for classification. Each gesture represents a particular command that is given to the multi-rotor aerial vehicle as a control signal. The possible classes are (a) Palm (Class $\#1$): take off, (b) Fist (Class $\#2$): land, (c) Victory sign (Class $\#3$): turn left, and (d) OK sign (Class $\#4$): turn right.

been illustrated in Figure 2.2. Considering the dataset and the problem statement, during the experimentation phase, we chose the number of hidden layers in the CNN model. We increased the number of each type of layer until the classifier achieved our desired performance. The CNN's performance decreased as the number of layers increased; therefore, we reduce the number of that specific layer by one unit and use the result in the CNN architecture.

4.3.3 Retrainer Module

Figure 4.8 illustrates that the Retrainer module is activated in two cases: (i) when the Classification Validation module detects low confidence in the confusion matrix and (ii) when the System Validation module sends the retraining penalty signal. After the Retrainer module is enabled, as described in [13] for the single-subject environment, the Retrainer receives S new images ($S = 5$ in our implementation) from the user as its input. The Retrainer module then retrains the running CNN model with the newly gathered data to obtain new weights. The module then fits the new weights to the CNN model and sends the CNN model to the classifier. Algorithm 1 illustrates the steps for the validation process in a single-subject environment.

Algorithm 1 Pseudo-code for the validation process in a single-subject environment.

```
begin
    User poses in front of the camera.
    Hand Detector ← Input frame
    Classifier ← Hand RoI
    Flight Controller ← Classification unit output
    Drone ← Flight Controller output parameters
    User observes the drone behavior.

    if User disapproves the drone behavior then
        User sends a command to the System Validation module through the smart glove.
        System Validation module checks the Flight Controller output parameters changes.
        if The changes of the Flight Controller output parameter are not as expected then
            System Validation module adjusts the Flight Controller.
        else
            System Validation modules activates the Retrainer module.
            while Confidence factors of the gestures are less than predefined threshold do
                The Retrainer captures $S$ new images ($S = 5$ in our implementation) from
                the low confidence gesture.
                Retrainer retrains the running CNN model with the newly gathered data in
                order to obtain new weights.
            end
            Retrainer fits the new weights to the CNN model, and sends the CNN model to
            the classifier.
        end
    end
end
```

4.4 Hand and Gesture Recognition in a Multi-Subject Environment

We use the CNN model in both single- and multi-subject environments to classify a right-hand gesture into one of the classes illustrated in Figure 4.9. Block diagram in Figure 4.10 illustrates that the gesture-based closed-loop control system for multi-subject environments contains five main units and modules: (i) the Smart Glove unit, (ii) a Skeleton Detection module and a Face Recognition, (iii) a Gesture Classification unit with a Retrainer and a Classification Validation module, (iv) a Flight Controller, and (v) a System

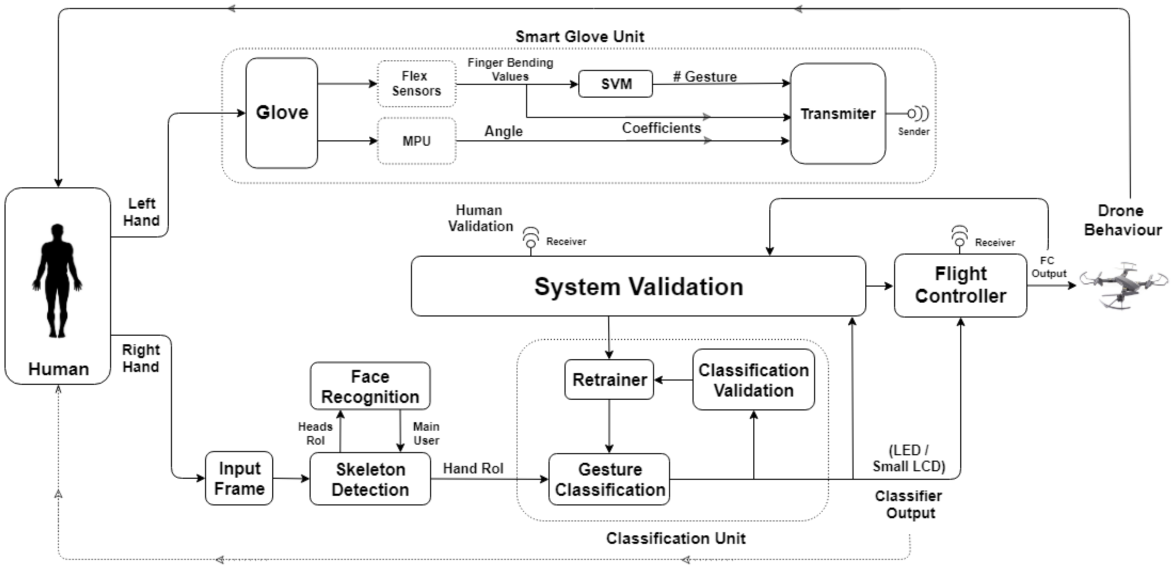


Figure 4.10: System block diagram for a multi-subject environment including three separate validation modules: (i) Classification Validation, (ii) System Validation, and (iii) human-in-the-loop validation.

Validation. In both single- and multi-subject environments, two types of gesture recognition are used to interact with an aerial vehicle: the smart glove-based method and the image processing-based method.

The Hand Gesture Classification unit in a multi-subject environment contains all the modules in the single-subject environment except for the Hand Detection module, as shown in Figures 4.11 and 4.12. The primary input to the Hand Gesture Classification unit is a

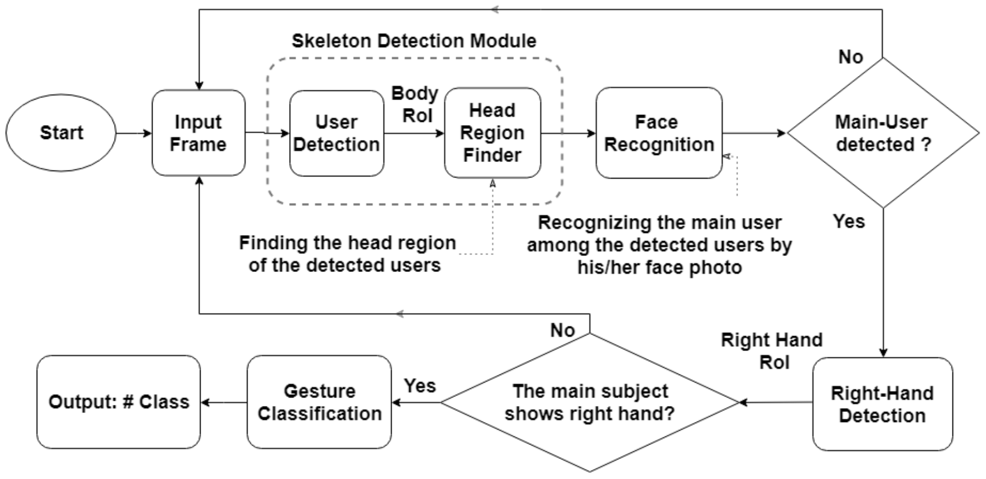

Skeleton Detection Module
Start
Input Frame
User Detection
Body RoI
Head Region Finder
Face Recognition
Main-User detected ?
No
Yes
Finding the head region of the detected users
Recognizing the main user among the detected users by his/her face photo
Right Hand RoI
Right-Hand Detection
The main subject shows right hand?
No
Yes
Gesture Classification
Output: # Class

Figure 4.11: Flowchart of hand gesture recognition in a multi-subject environment.

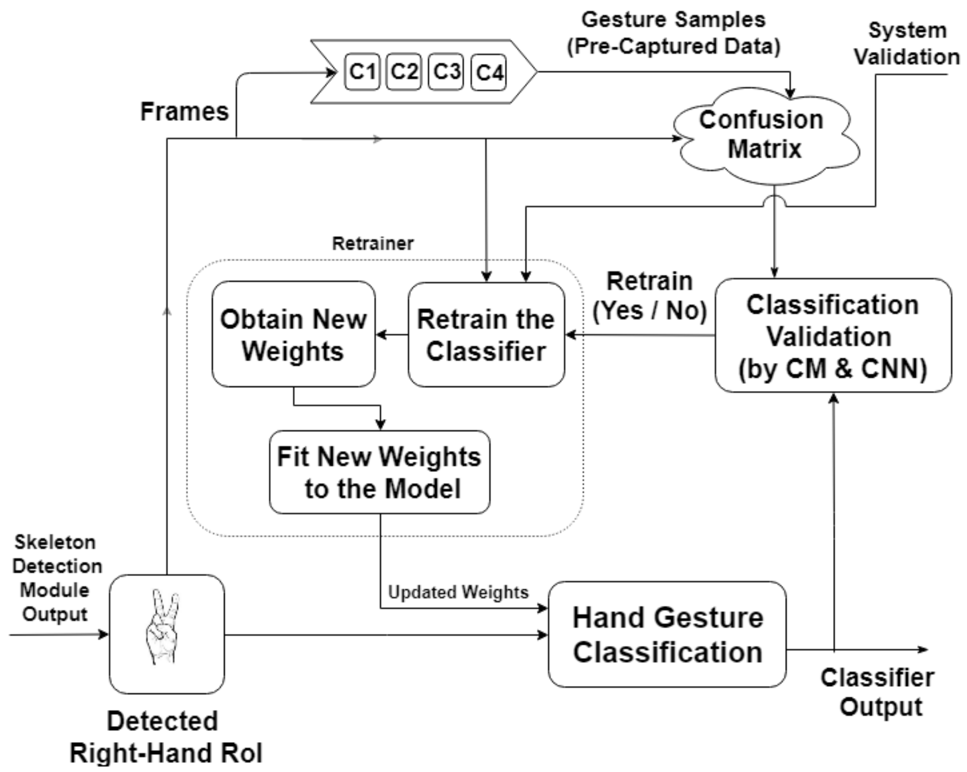


Figure 4.12: Gesture Classification block diagram in a multi-subject environment.

right-hand RoI, and the main output is the classified gesture represented as a class. In both single- and multi-subject environment cases, the classifier has an internal validation module that checks and improves the classifier's performance.

4.4.1 Skeleton Detection Module

In a multi-subject-environment scenario, the Skeleton Detection module recognizes the joints or each subject's skeleton in each input frame from the camera [41]. In order for the Skeleton Detection module to recognize joints of each subject, we assume that there is enough separation between the subjects. The proposed method is a top-down approach to detecting joints. The preliminary step to find each subject in every frame is to find a nose. The Skeleton Detection module begins from the nose, and then it searches for eyes, ears, a neck, and then the rest of the body as long as it is visible in the frame. If the Skeleton Detection module finds the nose and ears, the module counts that as a subject. Afterwards,

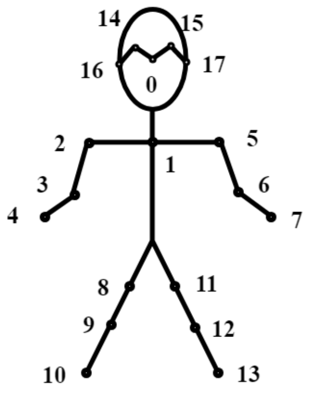


Figure 4.13: Numbered joints for the skeleton algorithm. The initial keypoint is the nose.
Numbered joints for the skeleton algorithm [42]. The initial keypoint is the nose.

the module connects the detected joints and forms a representation of the body of each subject.

If the skeleton detector fails to detect all the joints of a subject (e.g., the subject is too close to the camera), the module saves the related coordinates of each joint and creates a numbered list of detected joints and their coordinates from the input frame. Figure 4.13 shows the pattern used to number the joints [42].

4.4.2 Facial Recognition Module

After the Skeleton Detection module detects subjects in each frame, the next step is to determine which subject is the main subject. Head regions detected by the Skeleton Detection module are fed to the facial recognition model to identify the main subject. If the facial recognition model confirms the existence of the main subject, the algorithm uses the joint coordinates obtained from the Skeleton Detection module to find the right hand of the main subject. For the facial recognition model, we use a condition-free (meaning no illumination changes or occlusions) facial recognition library with 99.38% accuracy that requires just one sample image to be trained [43]. This facial recognition model uses the histogram of oriented gradients (HOG) method to find the face patterns in the detected head region and the face landmark detection model to recognize the main subject's face [44, 45].

The goal of the HOG method is to find a pattern in the image that has a face. The HOG

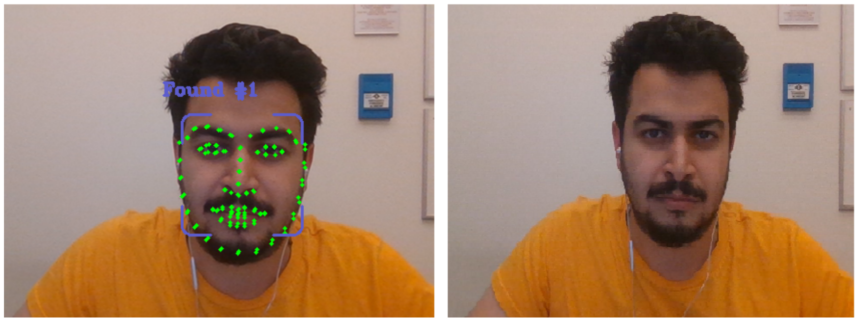


Figure 4.14: Face landmark detection.

finds the basic structure of the image based on the lighting of each pixel. The next step is to find the part of the image that is similar to a known HOG pattern extracted from a dataset of the main face [44].

The HOG model is pre-trained, while the face landmark detection model is unique for each person [45]. Figure 4.14 shows that the facial recognition model creates a face landmark for the main subject before the Skeleton Detection module begins processing. The facial recognition model then creates the landmark based on the single image of the main subject's face and uses it to detect this face among the detected head regions from the Skeleton Detection module. The proposed method requires a face image of the main subject to train the facial recognition model during the initialization phase.

4.4.3 Right-Hand Detection Module

The right-hand detection module, which is part of the Skeleton Detection module, searches through the detected joints of the main subject for the wrist of the right hand. The right-hand detection module then sends the right-hand region to the posture classification module (Figure 4.11). Chapter 4.5 describes how we evaluate the behavior of the algorithm in various multi-subject-environment scenarios, including with a hidden right hand and no gesture performed by the main subject.

4.4.4 Right-Hand Classifier

Upon receiving the right-hand RoI as an input image, a CNN model classifies the detected hand into one of the predefined classes (Figure 4.9). In both cases, a single-subject and a multi-subject environment, the classifier output maps to one of the predetermined commands: take Off, land, turn right, or turn left. Subsequently, as illustrated in Figures 4.10 and 4.12, the output command is sent to the Classification Validation module to check the classification confidences and then to the next module to execute the command.

4.4.5 Retrainer Module

Figure 4.12 illustrates that the Retrainer module in both single- and multi-subject environments is activated in two cases: (i) when the Classification Validation module detects low confidence in the confusion matrix and (ii) when the System Validation module sends the retraining penalty signal. After the Retrainer module is enabled, as described in the Section 4.3.3 and [13], the Retrainer receives S new images ($S = 5$ in our implementation) from the user as its input. The Retrainer module then retrains the running CNN model with the newly gathered data to obtain new weights. The module then fits the new weights to the CNN model and sends the CNN model to the classifier. The validation processes for a crowded environment are explained in Algorithm 2.

4.5 Validation Modules

Figures 4.7 and 4.10 show that the proposed algorithm for both single- and multi-subject environments has three validation modules that work together to improve system performance and reliability. In this section first we describe the human-in-the-loop validation, where the user is the first to validate the system output by observing the robot's behavior, second, we describe the outer layer of the system where there is a validation module that finds the source of a problem with either the classifier or the Flight Controller, and finally,

Algorithm 2 Pseudo-code for the retraining and validation process in a multi-subject environment.

```
begin
    Subjects pose in front of the camera.
    Skeleton Detection module ← Input frame
    Face Recognition module ← Heads RoI
    Skeleton Detection module ← Main subject ID
    Classifier ← Hand RoI
    Flight Controller ← Classification unit output
    Drone ← Flight Controller output parameters
    User observes the drone behavior.

    if User disapproves the drone behavior then
        User sends a command to the System Validation module through the smart glove.
        System Validation module checks the Flight Controller output changes.
        if The changes of the Flight Controller output are not as expected then
            System Validation module adjusts the Flight Controller.
        else
            System Validation module activates the Retrainer module.
            while Gestures confidence values are less than predefined threshold do
                The Retrainer captures $S$ new images from the low confident gesture.
                Retrainer retrains the running CNN model with the newly gathered data in
                order to obtain new weights.
            end
            Retrainer fits the new weights to the CNN model, and sends the CNN model to
            the classifier.
        end
    end
end
```

we describe the Classification Validation module which validates the gesture classifier output.

4.5.1 Human-in-the-Loop Validation

The user is responsible for two tasks: (i) to feed the system with hand gestures and (ii) to validate the drone's performance, as illustrated in Figure 4.7. By observing the drone's behavior, the user decides whether the system output is correct or not. If the user decides the output is incorrect, the user can indicate disapproval through the smart glove by performing Gesture #15. The Gesture #15 is set to declare an observed misbehavior and activate System Validation. Then, the System Validation checks both the classifier output and the Flight Controller output signals, to find the source of the observed misbehavior.

4.5.2 Gesture Recognition Validation

The Classification Validation module is responsible for checking the classifier's accuracy and, if necessary, to send a retraining command to the Retrainer. The Classification Validation module has access to the CNN model. A trustworthy source is required to test the reliability of the classifier output. We use d images taken from the user during an initialization phase, and the system assigned them to the corresponding classes ($d = 10$ in our implementation and experimental results). The Classification Validation module assumes that such data are correctly classified. The Classification Validation module then uses the running CNN model to create the confusion matrix (CM), as shown in Table 4.3, and analyzes the CM to calculate running CNN classification errors.

Table 4.3: Confusion matrix for the CNN model. Class #1: take off, Class #2: land, Class #3: turn left, and Class #4: turn right.

-	Class #1	Class #2	Class #3	Class #4
Class #1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
Class #2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
Class #3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$
Class #4	$x_{4,4}$	$x_{4,4}$	$x_{4,4}$	$x_{4,4}$

Let us define a classifier output reliability R_{C_i} as

$$R_{C_i} = \frac{x_{i,i}}{\sum_{j=1}^{C} x_{i,j}}, \tag{12}$$

where $x_{i,j}$ is an element of the CM, with $i, j \in \{1, 2, 3, 4\}$, and $x_{i,i}$ is a diagonal element of the CM. If the classifier output reliability is less than a predefined threshold (in our implementation the threshold is set to 70% as a high accuracy that the classifier still classifies correctly), the Retrainer module retrains the CNN model with new images and updates the CNN weights.

When the classifier Retrainer is activated, $S = 5$ images from the gesture with the lowest confidence are taken by sending a specific command through the smart glove. After gathering the $S = 5$ images, with the mentioned system configuration, it takes up to 4.2 seconds to retrain the classifier and to fit the updated weights to the running CNN model. After each step (gathering images, retraining based on gathered images, and fitting the updated weights to the running CNN model), if the confidences are less than the threshold, the Classification Validation module reactivates the Retrainer. Repeatedly, the Retrainer takes the images produced by the user's command, retrains based on those images, and fits the CNN model. The Classification Validation module continues reactivating the Retrainer until all the calculated confidences become more than the threshold.

4.5.3 System Validation

Triggered by human validation, the System Validation module checks the performance of the Flight Controller and the classifier output. The System Validation initially assumes that the classifier output is correct and assumes the problem is related to the Flight Controller. If the Flight Controller check is passed, the initial assumption is proven wrong, so the System Validation module triggers the classification Retrainer module. After the System Validation module detects which output is incorrect, it transmits a command to the corresponding Retrainer to improve the performance.

To test the first assumption, the System Validation module compares the controller output parameters with their previous values for each command. For instance, if the classifier output command is landing, the System Validation module must receive a command to reduce the altitude parameter in the Flight Controller output; otherwise, it is a mismatch. If there is a mismatch, the System Validation module transmits a penalty command to the controller. Otherwise, the System Validation module sends the penalty command to the classification module to activate the Retrainer and update the CNN model.

Table 4.4: Configuration of the proposed CNN layers.

Layers	Config
Conv2D	activation: linear
	batch input shape: $70 \times 70 \times 1$
	kernel size : 3×3
	use bias: true
	filters: 32
	kernel initializer: class name: Variance Scaling distribution: uniform mode: fan avg scale: 1.0
Activation	ReLU
Max-Pooling 2D	pool size 2×2
Conv2D	activation: linear
	batch input shape: $70 \times 70 \times 1$
	kernel size : 3×3
	use bias: true
	filters: 64
	kernel initializer class name: Variance Scaling distribution: uniform mode: fan avg scale: 1.0
Activation	ReLU
Max-Pooling 2D	pool size 2×2
Conv2D	activation: linear
	batch input shape: $70 \times 70 \times 1$
	kernel size : 3×3
	use bias: true
	filters: 64
	kernel initializer class name: Variance Scaling distribution: uniform mode: fan avg scale: 1.0
Activation	ReLU
Max-Pooling 2D	pool size 2×2
Flatten	-
Dense	activation: linear
	kernel initializer: class name: Variance Scaling distribution: uniform mode: fan avg scale: 1.0
	units: 500
Activation	ReLU
Dropout	rate: 0.5
Dense	activation: linear
	kernel initializer class name: Variance Scaling distribution: uniform mode: fan avg scale: 1.0
	units: 500
Activation	ReLU
Dropout	rate: 0.2
Dense	activation: linear
	kernel initializer class name: Variance Scaling distribution: uniform mode: fan avg scale: 1.0
	units: 4
Activation	Sigmoid

Chapter 5

Validation and Verification

5.1 Introduction

This chapter presents the training and testing details of each model in this thesis. To validate the proposed algorithm's performance, we perform different scenarios with different group sizes in front of the camera. There is one main subject in each group that performs different gestures in front of the camera. We study the behavior of the proposed algorithm in different desired and undesired scenarios. Finally, this chapter presents comprehensive experimental results that validate the proposed algorithm for several desired and undesired cases.

Additionally, a multimedia file is attached to this thesis. We use CoppeliaSim software to test the proposed algorithm and create this multimedia file. This multimedia file shows a drone that receives commands from two sources: (i) a camera for right-hand gesture classification and (ii) a smart glove for left-hand gesture classification.

5.2 CoppeliaSim

We use a CoppeliaSim simulation environment to emulate a real-life scenario of drone control, and the drone behavior in this environment is recorded in the attached multimedia file. We demonstrated a drone's behavior based on four commands from the image-based

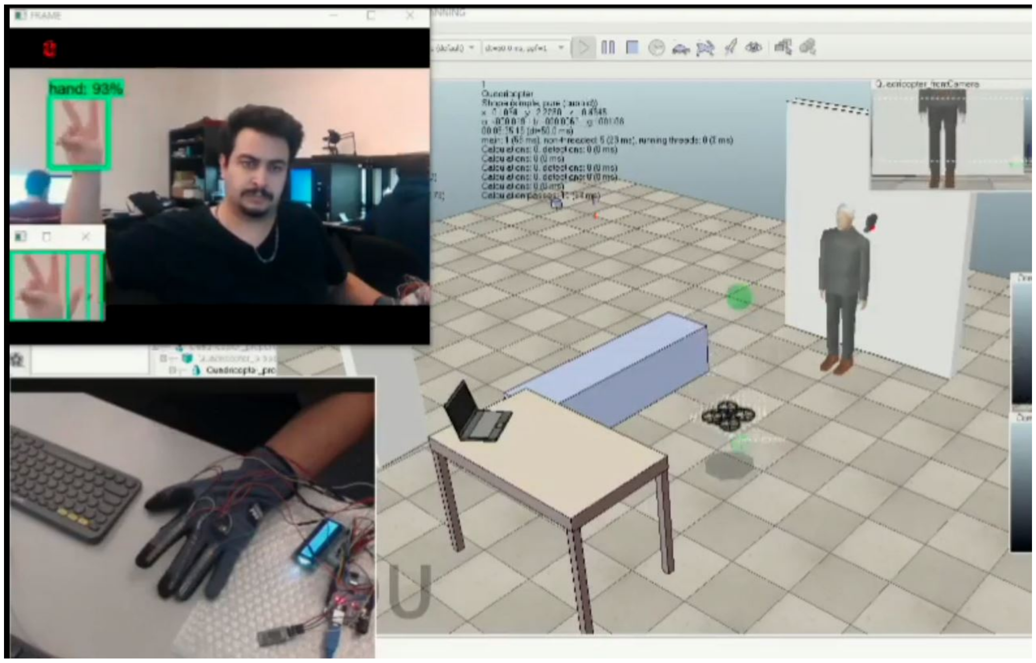


Figure 5.1: Multi-rotor aerial vehicle control using hand gestures and smart glove with self-validation.

input of the right-hand gesture and five commands from the smart glove-based input of the left-hand gesture (see Figure 5.1 and the multimedia file).

The multimedia file shows a drone that receives commands from two sources – a camera for the right-hand gesture classification, and a smart glove for left-hand gesture classification. A PID controller is applied as a Flight Controller for the drone. After the main subject performed the gestures through the smart glove and the camera, the proposed method creates and sends commands to the cloud-based environment. The Flight Controller reads the commands from the cloud-based environment and sets the drone controller parameters.

5.3 Gesture Recognition Using Smart Glove

The user calibrates the MPU by keeping his or her hand parallel to the ground and calibrates the flex sensors by fully closing and opening the fingers. The calibration steps and the smart glove's functionality are the same in both single- and multi-subject environments. Furthermore, the smart glove operates in two different modes simultaneously.

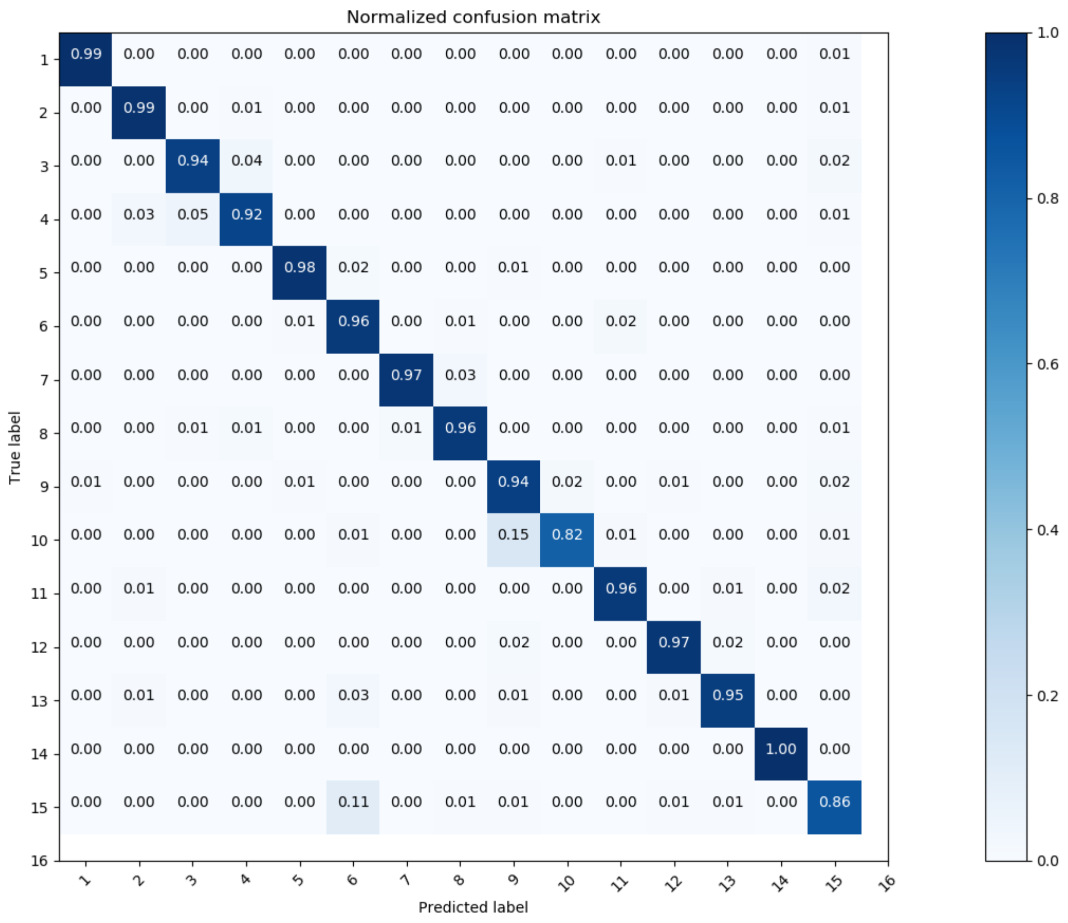


Figure 5.2: Support vector machine model confusion matrix for the left-hand gesture recognition.

The first mode considers each finger as an "on switch" when the finger is bent and an "off switch" when the finger is fully straight. The second mode uses the exact bending value from each flex sensor.

5.3.1 Gesture Classifier Using the SVM Model

The smart glove supports 16 different gestures and the SVM model with a linear kernel is used to analyze these gestures. This model is trained using 7216 samples on four fingers in 16 different classes (commands). Figure 5.2 demonstrates the CM of the SVM model after training.

Furthermore, Figure 5.3 shows the differences in value for each gesture. Descriptions of the gestures generated with the smart glove are provided in Table 4.1 and are the same in both single- and multi-subject environments.

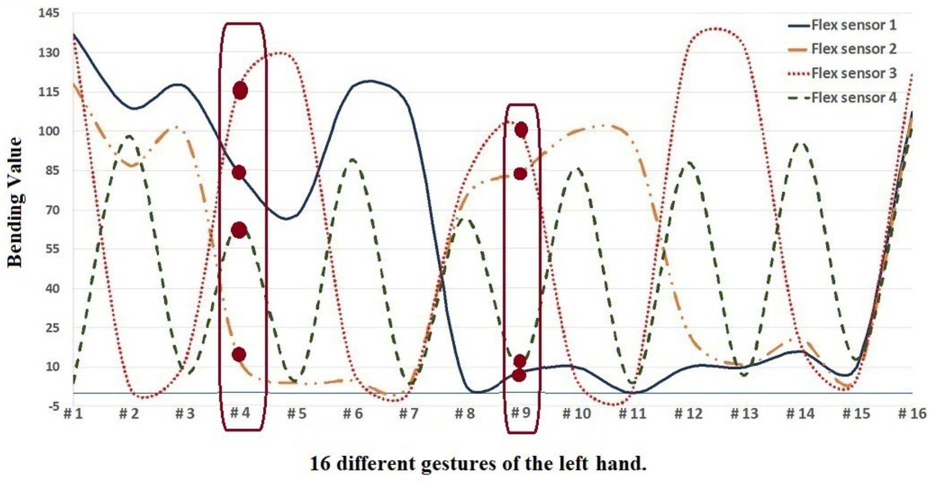


Figure 5.3: Flex sensors bending values through the 16 gestures of the smart glove. The generated signals with the flex sensors are used for left-hand gesture recognition and the generation of continuous control signals. As indicated on the horizontal axis, a combination of four bending values from the flex sensors represents a specific gesture.

5.4 Gesture Recognition in a Single-Subject Environment

Right-hand gesture recognition consists of a Hand Detection module and a Gesture Classification module. The algorithm detects the hand in each frame and then classifies the detected gesture. We implement the Hand Detection and the Gesture Recognition modules using the AUSDOM AW335 Full HD 1080p webcam. Moreover, we examine the performance of the validation system in various scenarios. A machine with 16 GB of RAM and NVIDIA Geforce 1060 Ti GPU uses to train and validate the modules.

5.4.1 Hand Detection Using the SSD Model

From 46 different subjects, we obtain 4600 images (100 images per subject located at various distances from the camera) that are separated into training and validation sets. The training and validation sets consist of 85% and 15% of the total dataset, respectively.

The loss function for the SSD model is a sum of the confidence loss L_{conf} and localization losses L_{loc}

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + L_{loc}(x, l, g)), \tag{13}$$

where N is the number of matched default boxes. The confidence loss L_{conf} is the softmax loss over multiple classes c confidences and the localization loss L_{loc} is the loss between the predicted box l and the ground truth box g parameters [12].

Softmax function in the L_{conf} is calculated as follows:

$$L_{conf}(x, c) = \frac{e^x}{\sum_{j=1}^{c} e^{x_j}}, \tag{14}$$

where the x is the input and the e^x gives a positive value above 0 but not fixed in the range (0, 1), which is what is required of a probability. The $\sum_{j=1}^{c} e^{x_j}$ is the normalization term, which guarantees that all the output values of the function will sum to 1, thus forming a valid probability distribution.

During training, the total loss between the training set and the validation set is expected to converge to the minimum. When the total loss converges to the minimum, an inference graph is generated, as shown in Figure 5.4. This graph compares the total loss to the training steps for the SSD. The tracking process is tested on a set of 635 new JPEG color images. The average accuracy is 94.28%, which is obtained in 1.43 seconds.

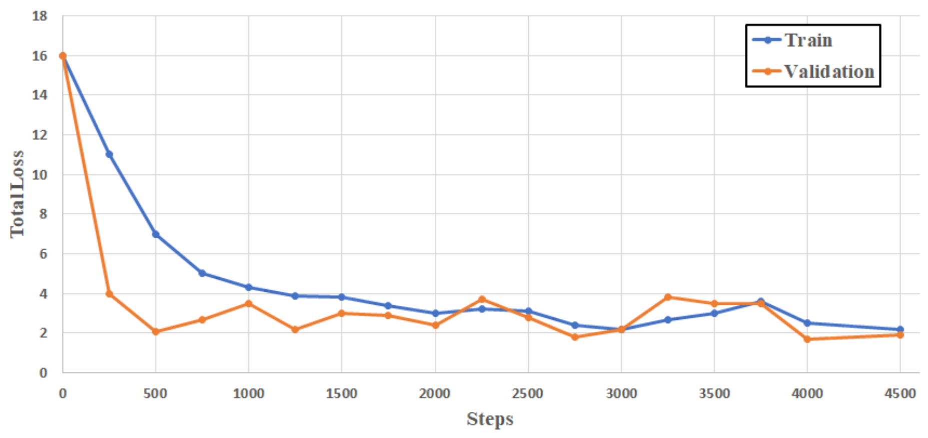


Figure 5.4: Total loss of the training process with respect to the steps.

5.4.2 Gesture Classifier Using the CNN Model

A dataset is gathered by capturing 125 color images of size 640×480 pixels from each subject for each distance from the camera. In total, 5000 images are collected from 40 subjects standing at various distances from the camera (i.e., 1 m, 1.5 m, 2 m, 2.5 m, and 3 m). Subjects stand in front of a white background with their right hand showing four predetermined static gestures as shown in Figure 4.9. After manually filtering the blurry and noisy images, a dataset of 4600 JPEG color images is obtained.

To train the model, we divide the dataset into a training dataset and a testing dataset. We allocate 70% and 30% of all refined images to the training and testing sets, respectively. Data augmentation is used on the training set to prevent overfitting. A sign of an overfitted model is when the training loss is low and the validation loss is high [17]. In Figure 5.5(b), the training loss is higher than the validation loss. Both validation and training losses diminish until they reach their lowest value after 14 epochs and overfitting is avoided at each epoch.

The model is trained for 15 minutes for 15 epochs with an accuracy of 96.94%. The batch size is 32, the Adam optimizer is chosen as the optimizer, and mean squared error is used as a loss function in the training process. After training, the system's accuracy on 120 images from each class (480 images the system had never seen before) is 98.12%. Figure 5.5 shows the training and test graphs from this process.

Table 5.1 shows the accuracy, loss values, and their related training data sizes. The proposed CNN model, with details is given in Table 4.4, which shows convergence for data sizes larger than 700 images for each class (gesture). In addition, the data augmentation technique, based on vertical flipping and setting the degree range for random rotation, is used to reduce the required data size and reach a higher accuracy.

Note that the system performance in a single-subject environment in [13] is evaluated with various scenarios and penalties on the controller and the classifier.

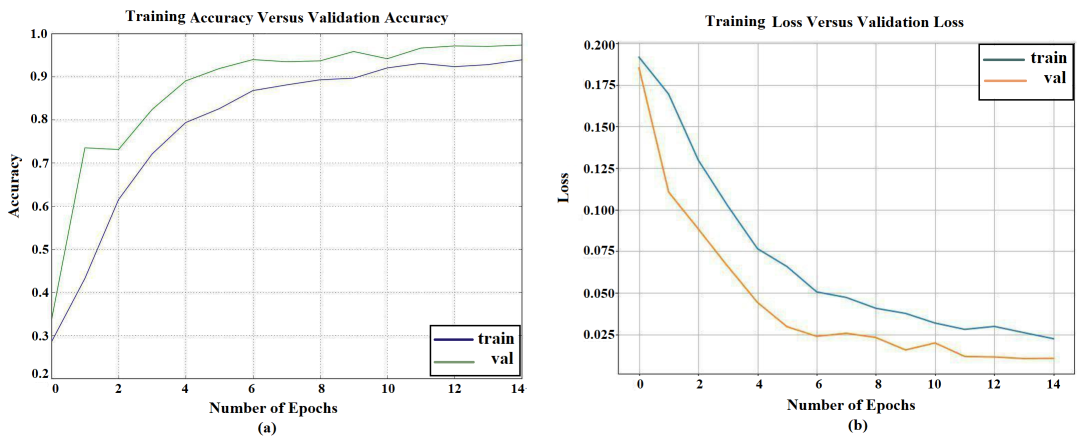


Figure 5.5: Training set versus validation set in a single-subject environment: (a) Accuracy; (b) Loss.

Table 5.1: Righ-hand gesture classifier training and validation results for different data sizes.

Data Size (number of images for each of four classes)	Training Accuracy %	Validation Accuracy %	Train Loss	Validation Loss
25	36.11	31.25	0.186	0.189
50	50	46.77	0.159	0.151
100	62.41	69.67	0.129	0.101
200	82.03	81.4	0.074	0.068
350	84.73	91.23	0.055	0.035
500	89.59	95.18	0.039	0.019
700	**94.29**	**96.8**	**0.023**	**0.01**
1150	91.36	93.94	0.039	0.012

5.5 Gesture Recognition in a Multi-Subject Environment

We demonstrate the system's performance for a multi-subject environment captured by a LOGI HD $1080p$ webcam. To implement the algorithm, we use an external Linux-based GPU and an NVIDIA Jetson AGX Xavier. The algorithm runtime for the multi-subject environment on the GPU for one and two subjects in the frame is 15 to 19 frames per second, and the runtime for three and four subjects in the frame is 10 to 13 frames per second.

To evaluate the accuracy and to study the behavior of the proposed algorithm in a multi-subject environment, we created a dataset of subjects showing different predefined right-hand gestures. We randomly selected n subjects with different physical features and placed them in different group sizes to be photographed in various situations making predefined gestures in front of the camera. We distributed n subjects in groups of m users as shown in Table 5.2. After filtering and excluding blurry and noisy images, we use the remaining images to test the proposed method. Furthermore, we add another class in case the main subject does not perform any gestures and keep his or her right hand down. With each image, we test the proposed algorithm with different users based on the number of subjects in each image.

5.5.1 Gesture Classifier Using the CNN Model

To classify the right-hand gestures, the same CNN model is used in both single- and multi-subject environments. A dataset is created by capturing 125 color images of size 640×480 pixels for each user with varying distances from the camera. A total of 8125 images are collected from 65 subjects standing at various distances from the camera (i.e., $1m$, $1.5m$, $2m$, $2.5m$, and $3m$). After manually filtering the blurry and noisy images, a dataset of 7600 JPEG color pictures is obtained. We use 70% of the refined images to train the CNN model for classification and use the rest as the test dataset.

To avoid overfitting at each epoch, the data augmentation technique is used in the training process, see Figure 5.6. The model is trained for 30 epochs with an accuracy of 97.33%. The batch size is 32, the Adam optimizer is chosen as the optimizer, and mean squared error is used as a loss function in the training process. After training, the system's accuracy on 300 images from each class (1200 images the system had never seen before) is 96.41%.

Four right-hand gestures are used in the proposed algorithm's implementation for both single- and multi-subject environments; however, the proposed algorithm can handle a variety of numbers of gestures. For a different number of gestures, only the CNN model needs to be changed. The CNN model is described in Table 4.4, and the parameter that requires tuning in this case is the dense unit of the last layer. Afterward, the CNN model can receive training on C gestures. In this thesis, four gestures ($C = 4$) are chosen, but there is no limitation on the number of gestures for the proposed algorithm.

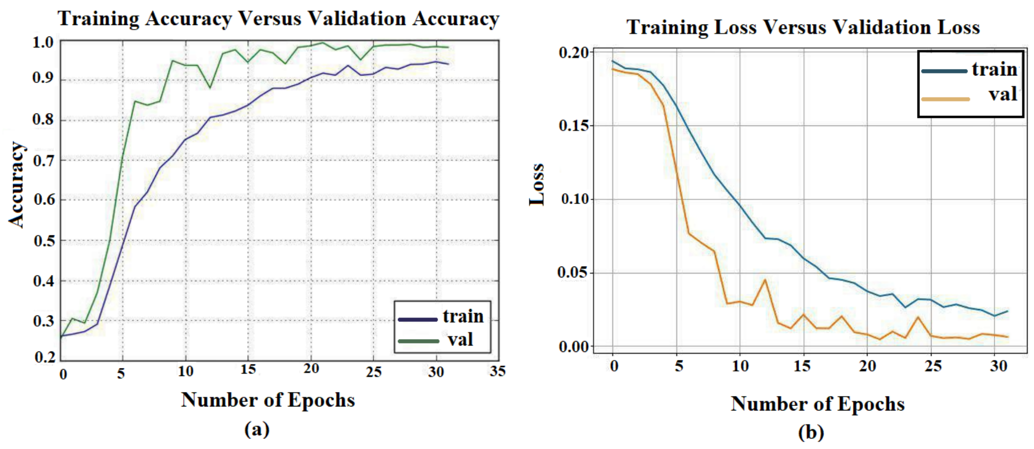


Figure 5.6: Training set versus validation set in a multi-subject environment: (a) Accuracy; (b) Loss.

5.5.2 Skeleton Detection Using R-CNN Model

To detect all subjects in the input frame, we use a joint-based detection module as the Skeleton Detection module [41]. We implement this joint-based detection module with an R-CNN network on the GPU with 15 frame rate per seccond (FPS). The proposed algorithm maps the joints, as shown in Figure 4.13.

The Skeleton Detection module has four primary responsibilities: (i) detecting the initial key points, which are the nose and ears of each subject in the input frame, (ii) detecting the joints of every subject in the input frame, (iii) obtaining the head RoI for every detected subject, and (iv) finding the main subject's right-hand RoI.

To test the proposed method in a multi-subject environment, we randomly organize subjects into 35 groups of a single subject, 190 groups of two subjects, 1120 groups of three subjects, and 1260 groups of four subjects. Each group is used to test all the main modules, such as Skeleton Detection, facial recognition, right-hand detection for the main subject, and right-hand gesture classification. Table 5.2 shows a summary of results for each of the groups of single, two, three, and four subjects. The proposed method is tested on 10420 images.

Table 5.2: Gesture recognition in a multi-subject environment: Experimental results for four different group populations. A group of m subjects refers to the population of subjects in the input frame which means there are m subjects in the frame.

Group Population	Number of Subjects	Total Images	User Detection		Facial Recognition		Right-Hand Detection		Gesture Classification		Algorithm Accuracy
			Correct	%	Correct	%	Correct	%	Correct	%	%
1	35	140	140	100	140	100	140	100	129	92.14	98.04
2	20	760	756	99.47	755	99.87	755	100	679	89.93	97.32
3	16	4,480	4,395	98.1	4,309	98.04	4,258	98.82	3,601	87.57	94.88
4	10	5,040	4,809	95.42	4612	95.9	4471	96.94	3137	70.16	89.61

5.6 Effect of Activating a Flex Sensor on Velocity of the Drone Movement

To study the performance of sensors, the third flex sensor f_3 is activated and the effect of this activated sensor is discussed. As presented in Table 5.3, the third flex sensor, f_3, senses bending after 1 second, and the output of the SVM module therefore changes from 0000 to 0100. Immediately after the change in the SVM module output, the proposed algorithm begins to calculate the change. The computed change for the third flex sensor, $rs_3(t)$, rises from 0 to 31.32, and consequently, the $V_G(t)$ increases by $0.74m/s$. The $V_G(t)$ changes until the SVM module output resets to 0000, and later, the $rs_3(t)$ decreases back to zero or its minimum. The $V_G(t)$ variation and bending fluctuation for the third flex sensor are demonstrated in Figure 5.7. In Figure 5.7, the left vertical axis presents the bending value of the flex sensors, and the right vertical axis shows the velocity.

Table 5.3: Effect of the activated flex sensor on the drone velocity. Note: f_n: bending value of the n-th flex sensor, $G \# m$: m-th gesture with the smart glove, $V_G(t)$: drone velocity.

Time	f_1	f_2	f_3	f_4	SVM output	$rs_3(t)$	$V_G(t)$
1	10.67	15.21	5.78	0.37	$G\#0:0000$	0	$V_0 = 1$
3	12.19	7.82	68.41	15.72	$G\#2:0\underline{1}00$	38.41	1.2671
5	15.82	11.65	56.98	7.98	$G\#2:0\underline{1}00$	26.98	1.1834
7	3.49	0.03	118.71	13.19	$G\#2:0\underline{1}00$	88.71	1.6032
9	9.34	2.42	101.82	11.73	$G\#2:0\underline{1}00$	71.82	1.4883
11	18.31	9.98	95.36	20.41	$G\#2:0\underline{1}00$	65.36	1.4444
13	15.19	9.06	31.93	11.88	$G\#0:0000$	0	$V_0 = 1$

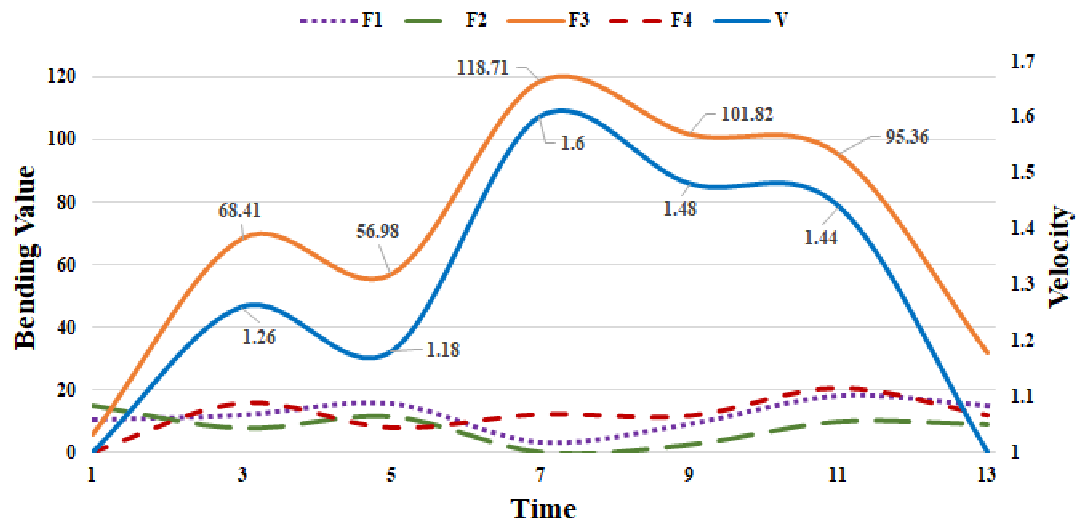


Figure 5.7: Activating the third flex sensor by bending the third flex sensor (f_3) and its effect on the velocity of the drone.

5.7 Test Scenarios

We tested various scenarios to study the system's behavior as follow:

- Testing validation modules in three different scenarios:
  - Validation system with no penalty.
  - Penalty on the Flight Controller.
  - Penalty on the classifier.
- Testing the system performance in different single- and multi-subject environments:

- Frame contains both hands.
- Frame without right hand.
- Frame contains right hand but there is no posture.
- Frame contains correct hand.

First, we tested the validation modules as presented in Figure 5.8. In all scenarios, the user shows Gesture #1 (a command to take off), but in each scenario, the user observes a different response.

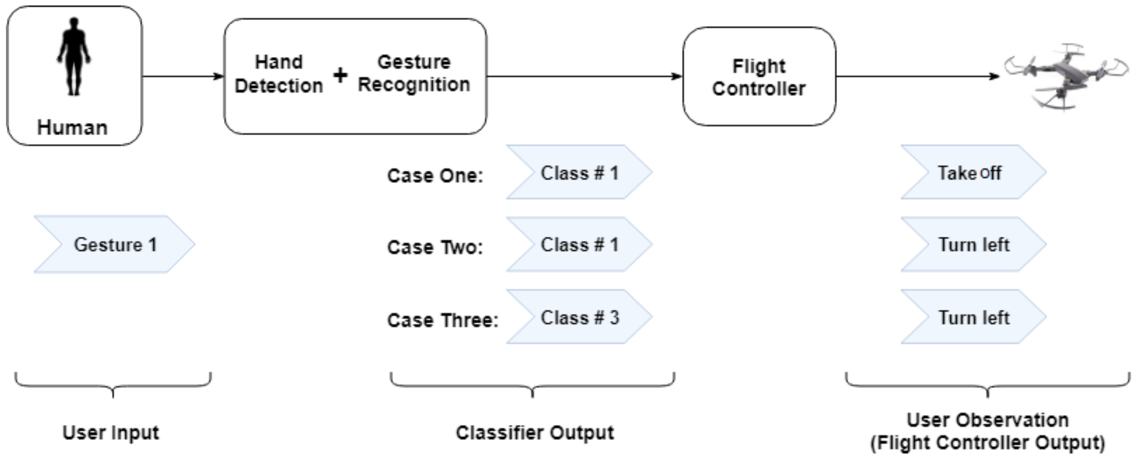


Figure 5.8: Three cases for the System Validation: 1) No penalty, 2) Penalty on the Flight Controller, and 3) Penalty on the classifier. The commands are: Class #1: take off, Class #2: land, Class #3: turn left, and Class #4: turn right.

5.7.1 Scenario 1: System Validation With No Penalty

In Scenario 1, the drone is on the ground, and the user makes Gesture #1, as shown in Figure 5.9. The Hand Detection module obtains the hand RoI and sends it to the Gesture Classification module. Hence, the classifier classifies Class #1. The classifier output is sent to the Flight Controller, and the Flight Controller generates a take off command. The user observes the drone taking off. Therefore, the user confirms the output (drone behavior). In this scenario, the system operates correctly.

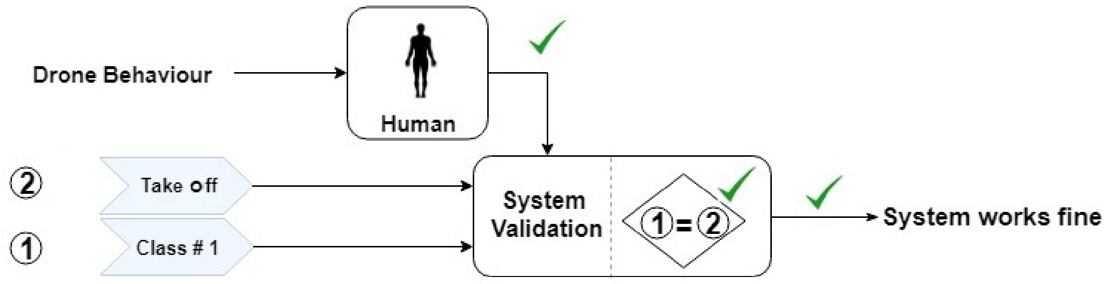


Figure 5.9: System without penalty. The commands are: Class #1: take off, Class #2: land, Class #3: turn left, and Class #4: turn right.

5.7.2 Scenario 2: Penalty on the Flight Controller

For the second scenario, the drone is on the ground again, and the user shows Gesture #1, as shown in Figure 5.10. As in the previous scenario, the classifier produces Class #1 as its output. The classifier output is sent to the Flight Controller, and this time, an error causes the Flight Controller to generate the turn left command instead of take off. Hence, the user rejects the drone behavior and sends a command to the System Validation module through his or her smart glove. When the System Validation module receives Class #1 from the classifier, expecting an increase in the Flight Controller output values, it then understands the mismatch and determines that the problem is with the Flight Controller. The System Validation module sends a penalty command to the Flight Controller, improving its future performance.

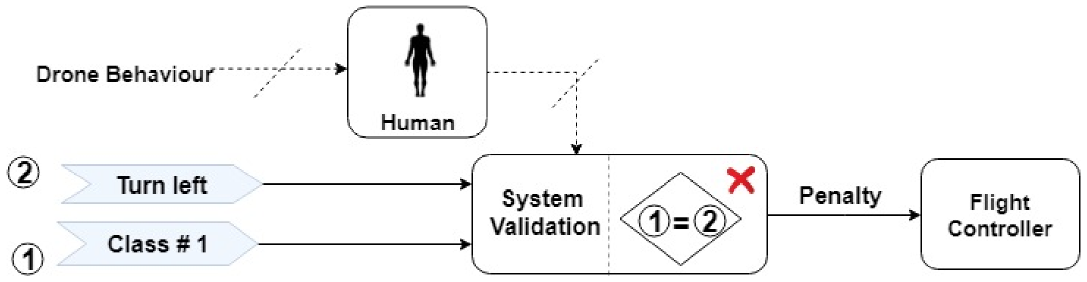


Figure 5.10: Penalty signal for the controller. The commands are: Class #1: take off, Class #2: land, Class #3: turn left, and Class #4: turn right.

5.7.3 Scenario 3: Penalty on the Classifier

The user shows Gesture #1, as shown in Figure 5.11. The hand detector chooses the hand RoI and transfers the RoI to the classifier. This time, the classifier provides Class #3 instead of Class #1 and then sends the classifier output to the Flight Controller. The Flight Controller presents the turn left command and forwards the command to the drone. The user observes the drone turning left but had expected the drone to take off. Therefore, the user rejects the drone behavior using the smart glove.

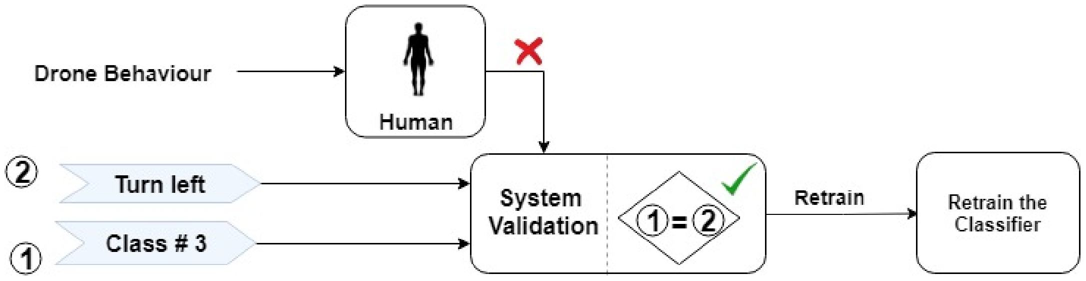


Figure 5.11: Penalty signal for the classifier. The commands are: Class #1: take off, Class #2: land, Class #3: turn left, and Class #4: turn right.

As indicated in Table 5.4 and Table 5.5, the System Validation module sends this penalty to the classifier. The classifier is then retrained in three steps. In the first step, the System Validation module checks the states of the classifier output and the controller output. The System Validation module discovers the controller operation is acceptable, so the problem is with the classifier. Next, the System Validation module sends the penalty

Table 5.4: Penalty on the classifier: Class #4. Classification Validation decides to retrain the model on Class #4 in three steps to improves the confidence from 54.55% to 79.19%. The four classes are: Class #1: take off, Class #2: land, Class #3: turn left, and Class #4: turn right.

Steps	Step #1				Step #2				Step #3			
Class	1	2	3	4	1	2	3	4	1	2	3	4
1	8	3	0	0	11	0	0	0	11	0	0	0
2	0	6	0	0	0	6	0	0	0	5	1	0
3	0	1	10	0	1	0	10	0	0	0	11	0
4	0	5	0	6	5	0	0	11	5	0	0	16
Confidant %	72.73	100	90.91	54.55	100	100	90.91	68.73	100	83.33	100	79.19

Table 5.5: Penalty on the classifier: Class #2. Classification Validation decides to retrain the model on Class #2 in three steps to improves the confidence from 16.67% to 100%. The four classes are: Class #1: take off, Class #2: land, Class #3: turn left, and Class #4: turn right.

Steps	Step #1				Step #2				Step #3			
Class	1	2	3	4	1	2	3	4	1	2	3	4
1	5	0	0	1	5	1	0	0	5	1	0	0
2	0	1	5	0	0	11	0	0	0	11	0	0
3	0	0	7	0	0	0	7	0	0	0	7	0
4	0	0	1	7	0	2	1	5	0	0	0	11
Confidant %	83.33	16.67	100	87.50	83.33	100	100	62.50	83.33	100	100	100

command to the classification module. The Retrainer then retrains the classifier and obtains new weights.

As presented in Table 5.4 and Table 5.5, when the classifier Retrainer is activated, $S = 5$ images from the gesture with the lowest confidence are taken by sending a specific command through the smart glove. After gathering the $S = 5$ images, with the mentioned system configuration, it takes around 4.2 seconds to retrain the classifier and to fit new weights to the running CNN model. After each step (gathering images, retraining based on gathered images, and fitting the new weights to the running CNN model), if the confidences are less than the threshold, the Classification Validation module reactivates the Retrainer. Repeatedly, the Retrainer receives the images taken from the user's hand, retrains based on those images, and fits the CNN model. The Classification Validation module continues reactivating the Retrainer until all the calculated confidences become more than the threshold.

In cases such as Scenario 2 and Scenario 3, where the system behavior is not what the user had expected, the drone will be frozen by receiving Gesture #15 through the smart glove. The drone will then take no commands unless the validation process is finished.

5.7.4 Scenario 4: Frame Contains Both Hands

As shown in Figures 5.12a and 5.12b, the user shows both hands at the correct angles performing the correct gestures. In the first step, the proposed algorithm detects the user in the frame and the user's joints through the Skeleton Detection module. Then, the algorithm sends the user's head RoI to the facial recognition module to find the main subject. After the facial recognition module identifies the main subject, the Skeleton Detection module finds the right-hand RoI (the algorithm also works on the left hand by changing the interested joint from joint 4 to 7; see Figure 4.13) and sends it to the classification module.

5.7.5 Scenario 5: Frame Without Right Hand

The user makes a gesture with the left hand, hiding the right hand, Figure 5.12d. As the first step, the Skeleton Detection module detects the subject in the frame and sends the subject's head RoI to the facial recognition module and returns a confirmation that the main subject has been detected. Then, the Skeleton Detection module searches for joint 4, and when it cannot find it in the list of the detected joints, the process ends until the user shows his or her right hand.

5.7.6 Scenario 6: Frame Contains Right Hand but There Is No Posture

The user shows no gestures and keeps the right hand down, Figures 5.14b, 5.14c, 5.15a and 5.15c. The proposed algorithm detects the right-hand joint and gets ready to capture the hand region and sends it to the classification module. Before doing so, the algorithm calculates the angle between joints 2 and 4 (Figure 4.13). The correct angle between joints 2 and 4 for a gesture in front of the camera is between 0 to 90 degrees. Therefore, if the user opens the right hand further than this range, the algorithm assumes that the right hand points toward the ground and the result is no gesture.

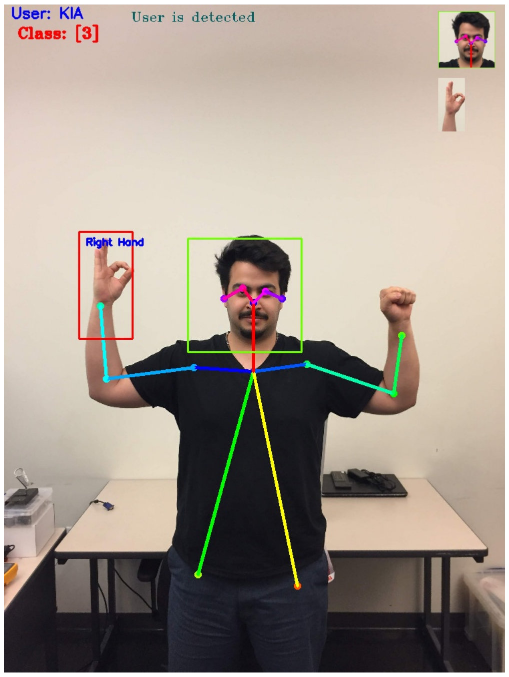
5.7.7 Scenario 7: Frame Contains Correct Hand

The subjects within the frame, including the main subject, show different gestures, and the proposed algorithm detects the main subject and his or her right-hand gestures.

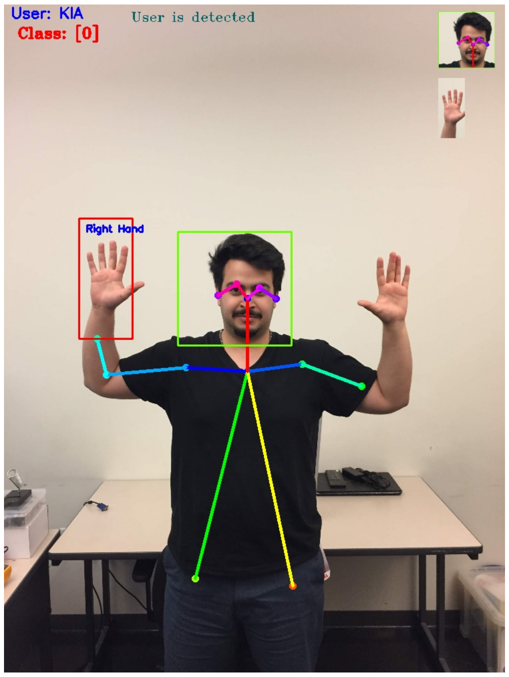
Note: in the following examples the classifier output number starts from zero to three for four classes, but the algorithm output that we mentioned in this thesis starts from one to four for four classes. The mapping Table 5.6 is provided to clarify this.

Table 5.6: Mapping table between classifier output and the algorithm output.

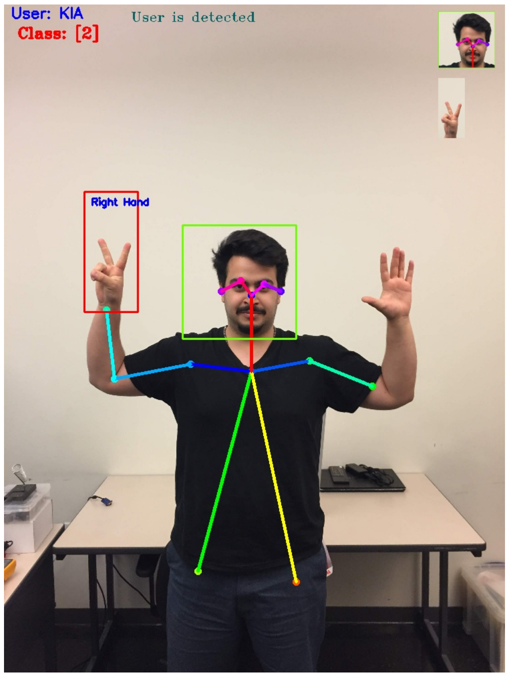
Class Numbers	
Classifier Output (top left corner in the images)	**Algorithm Output/Class (captions)**
0	1
1	2
2	3
3	4



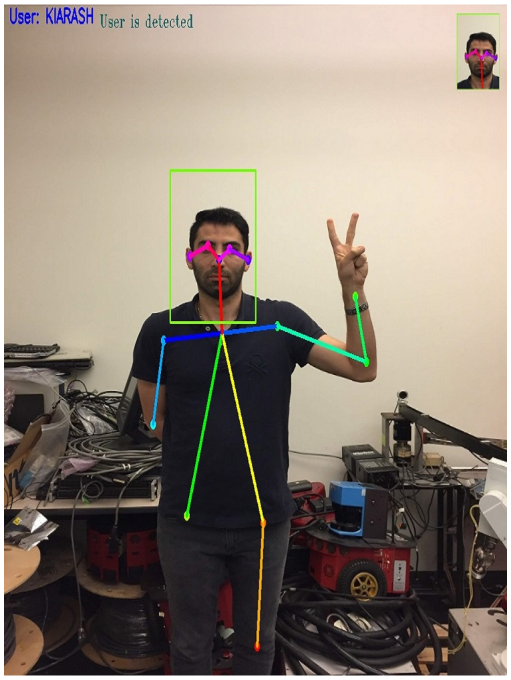
(a) Class #4: turn right



(b) Class #1: take off

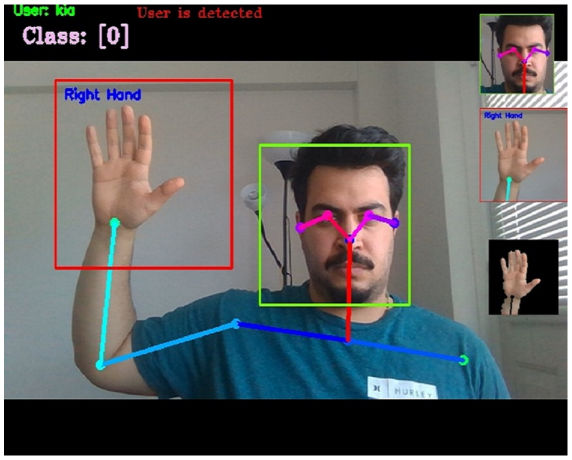


(c) Class #3: turn left

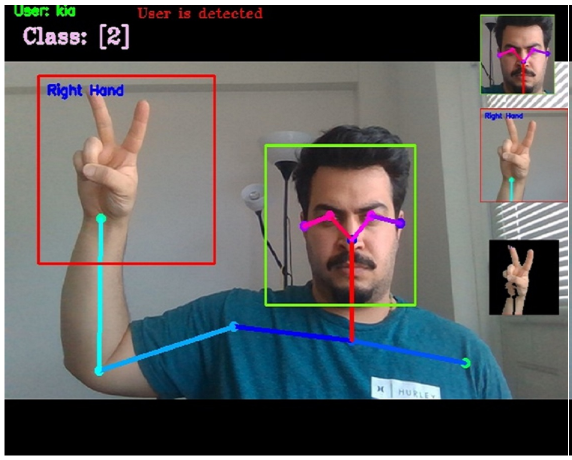


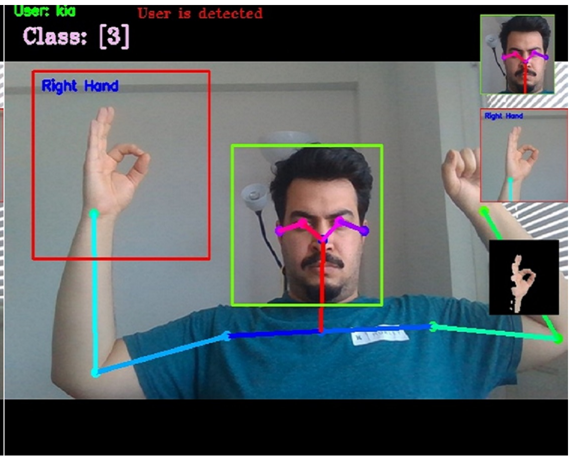
(d) No right hand

Figure 5.12: Gesture recognition on a single subject gesturing in front of the camera (series 1) . The commands are: Class #1: take off, Class #2: land, Class #3: turn left, and Class #4: turn right.



(a) Class #1: take off

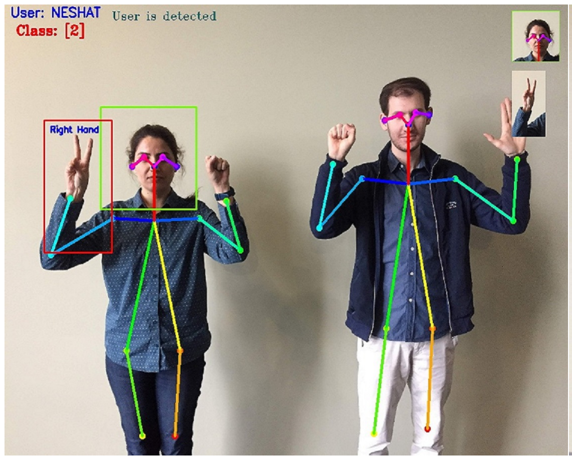




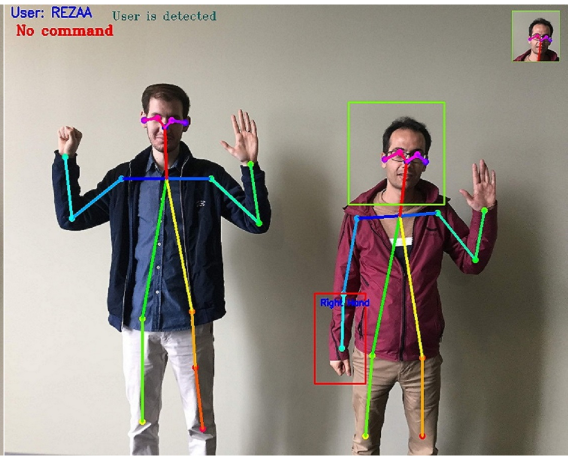
(b) Class #3: turn left

(c) Class #4: turn right

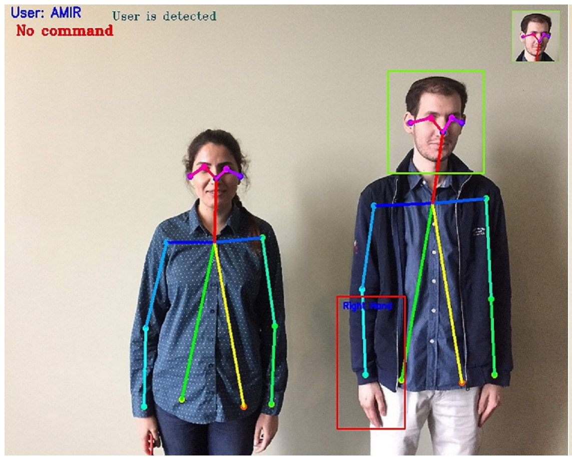
Figure 5.13: Gesture recognition on a single subject gesturing in front of the camera (series 2). The commands are: Class #1: take off, Class #2: land, Class #3: turn left, and Class #4: turn right.



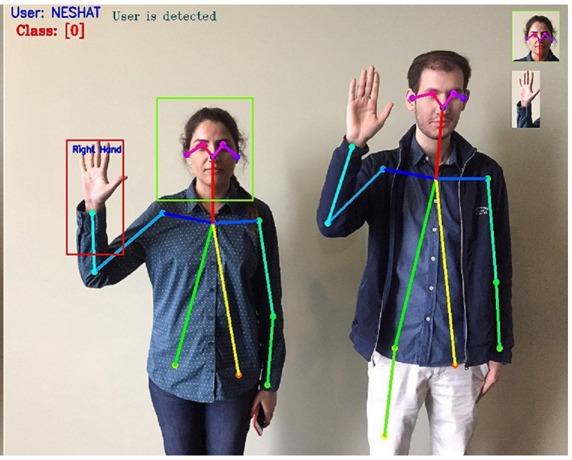
(a) Class #3: turn left



(b) No posture

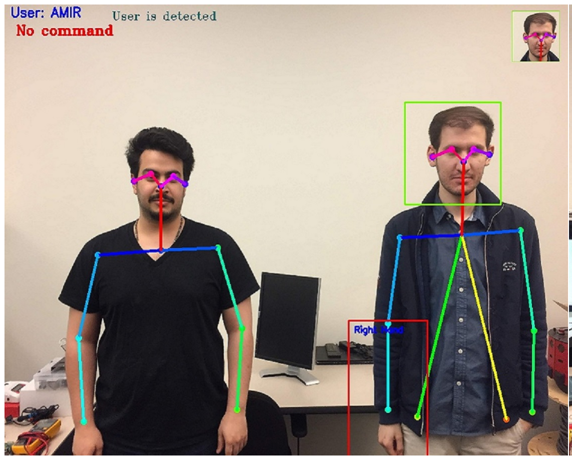


(c) No posture

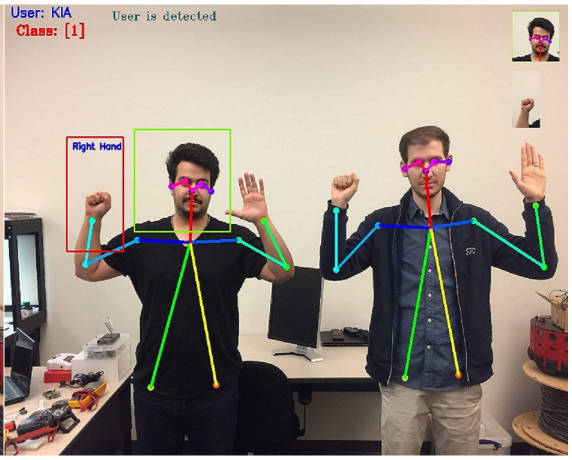


(d) Class #1: take off

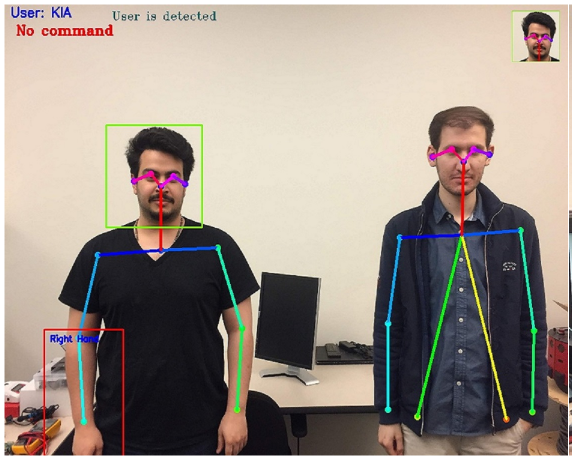
Figure 5.14: Gesture recognition on two (2) subjects including a main subject gesturing in front of the camera (series 1).



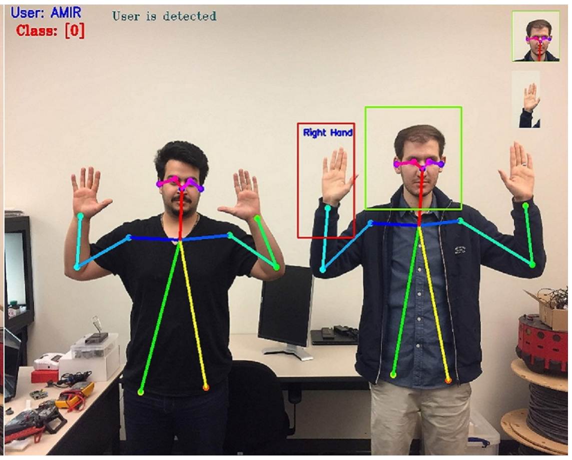
(a) No posture



(b) Class #2: land

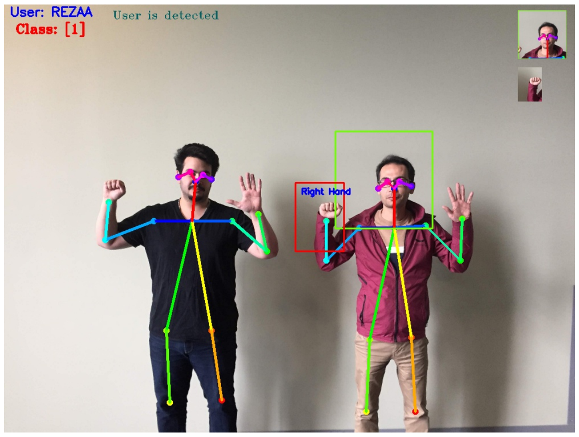


(c) No posture

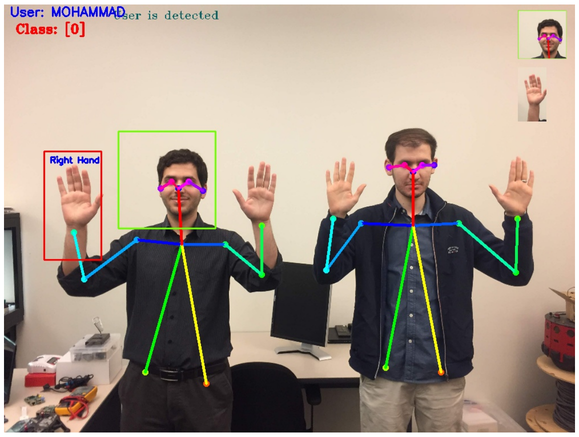


(d) Class #1: take off

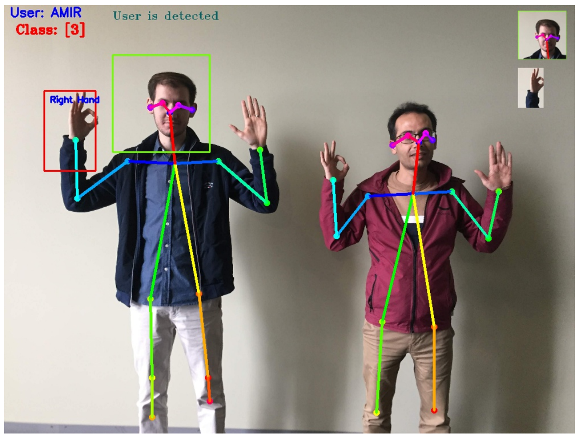
Figure 5.15: Gesture recognition on two (2) subjects including a main subject gesturing in front of the camera (series 2).



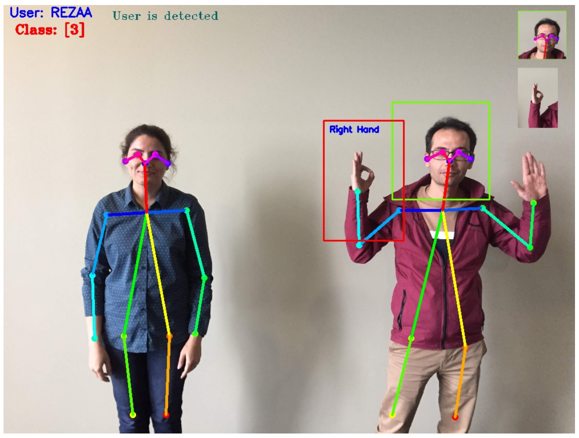
(a) Class #2: land



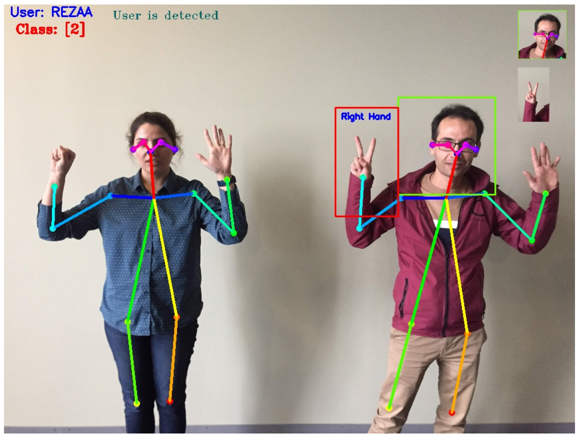
(b) Class #1: take off



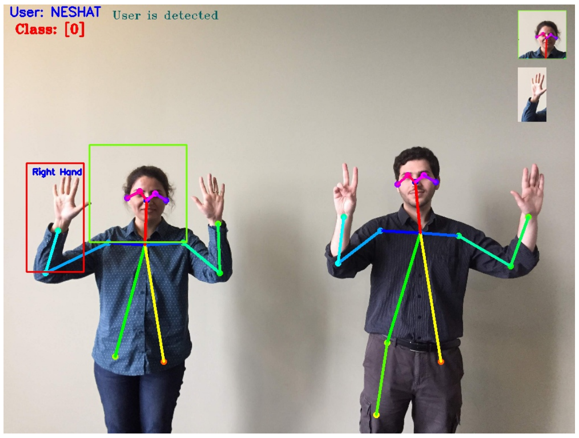
(c) Class #4: turn right



(d) Class #4: turn right

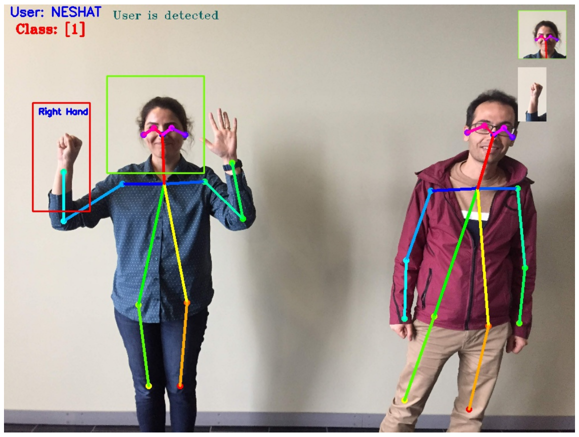


(e) Class #3: turn left

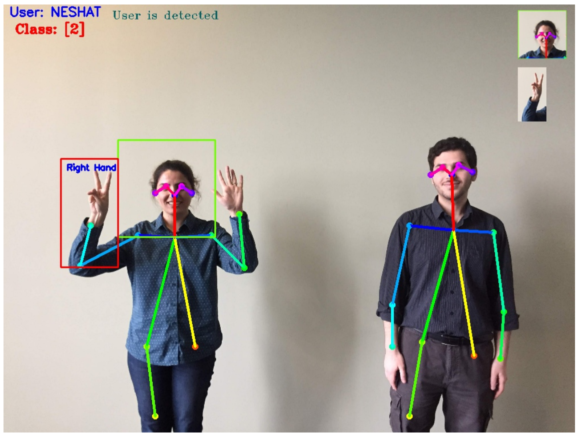


(f) Class #1: take off

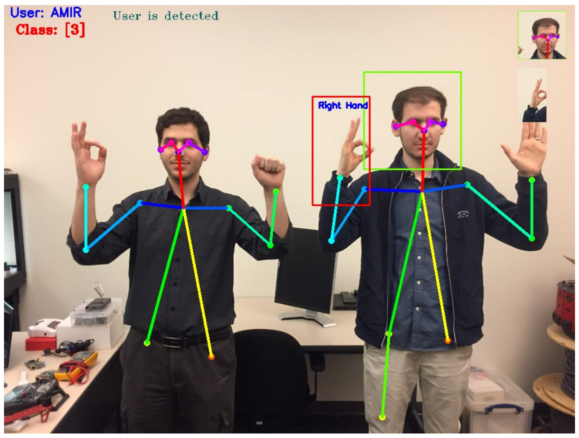
Figure 5.16: Gesture recognition on two (2) subjects including a main subject gesturing in front of the camera (series 3).



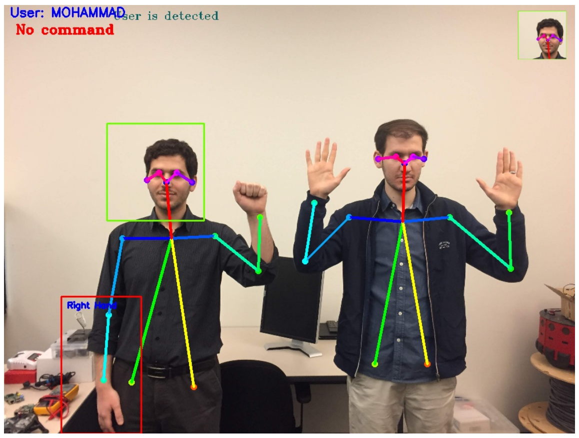
(a) Class #2: land



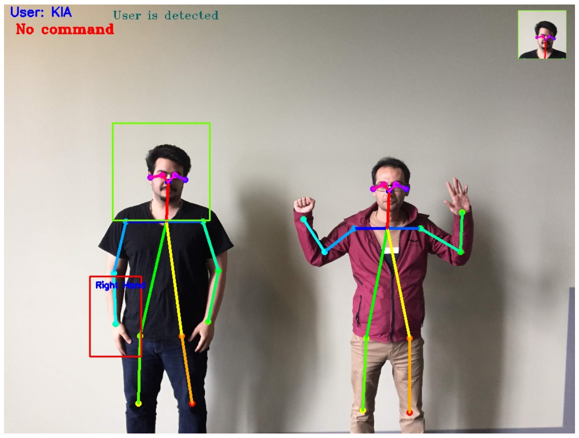
(b) Class #3: turn left



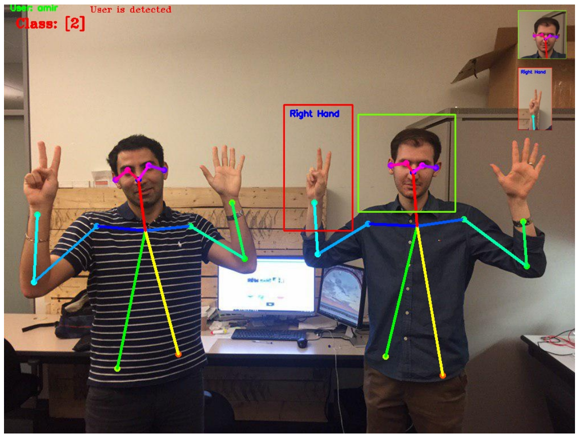
(c) Class #4: turn right



(d) No posture

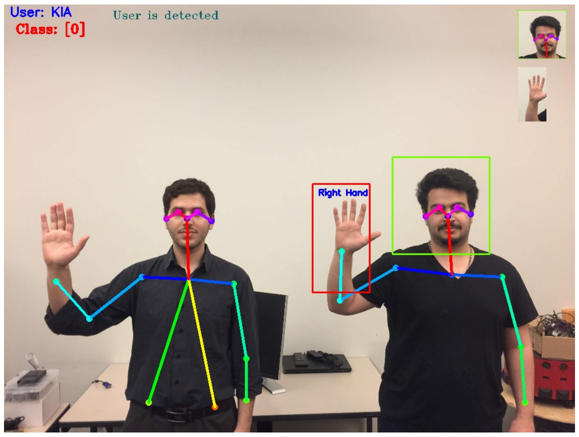


(e) No posture

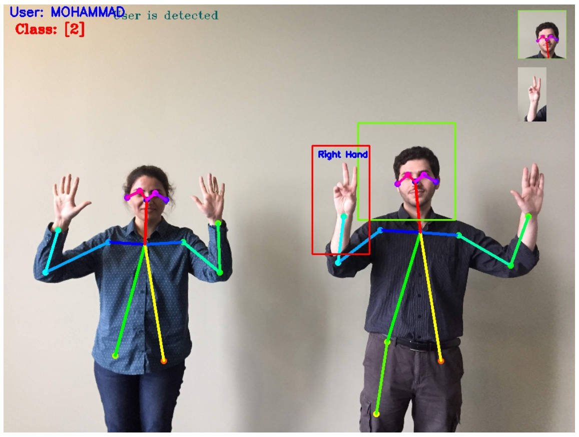


(f) Class #3: turn left

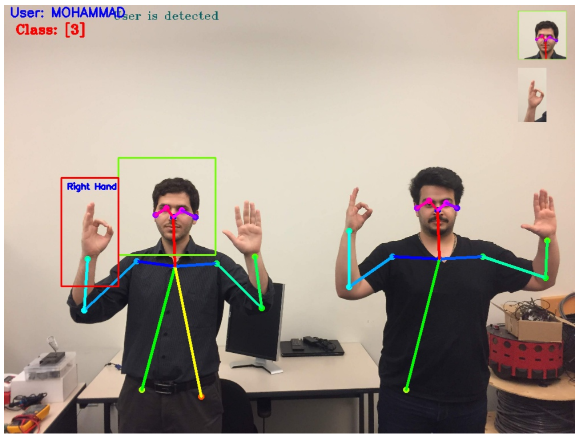
Figure 5.17: Gesture recognition on two (2) subjects including a main subject gesturing in front of the camera (series 4).



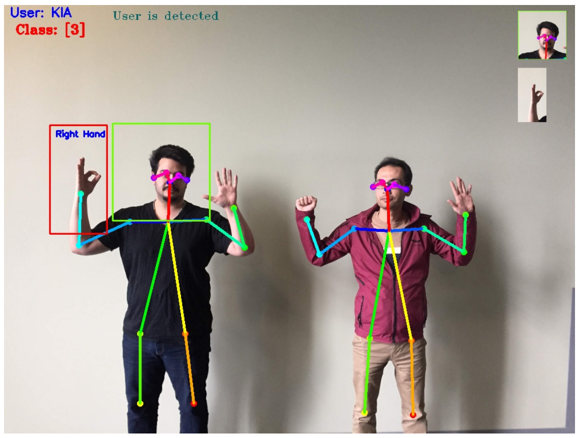
(a) Class #1: take off



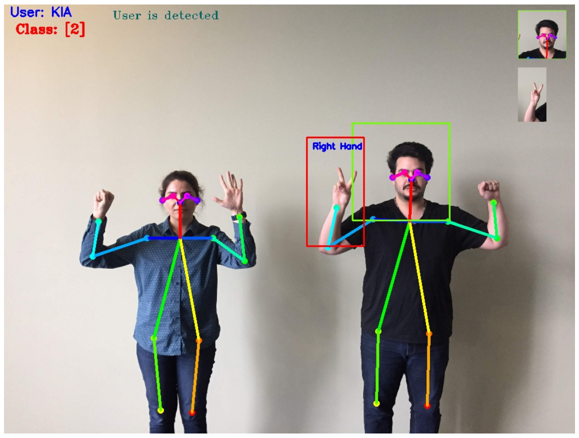
(b) Class #3: turn left



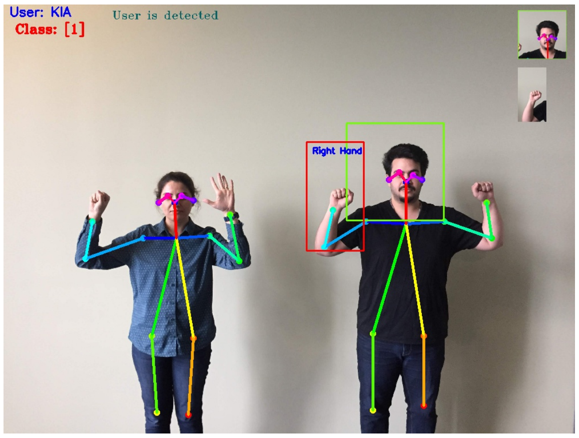
(c) Class #4: turn right



(d) Class #4: turn right

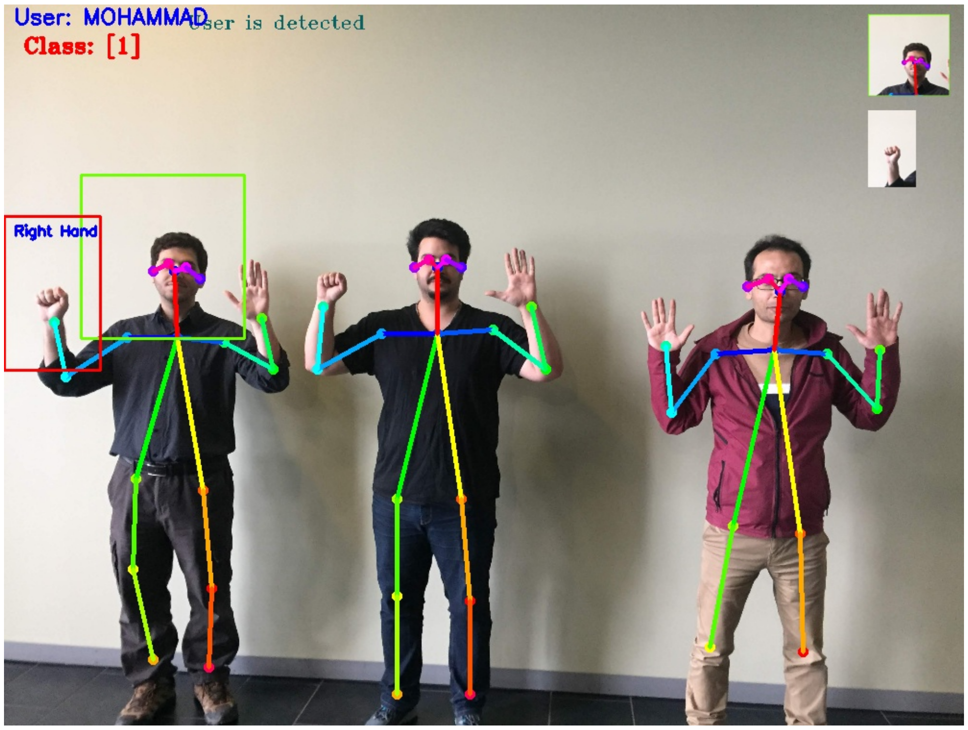


(e) Class #3: turn left

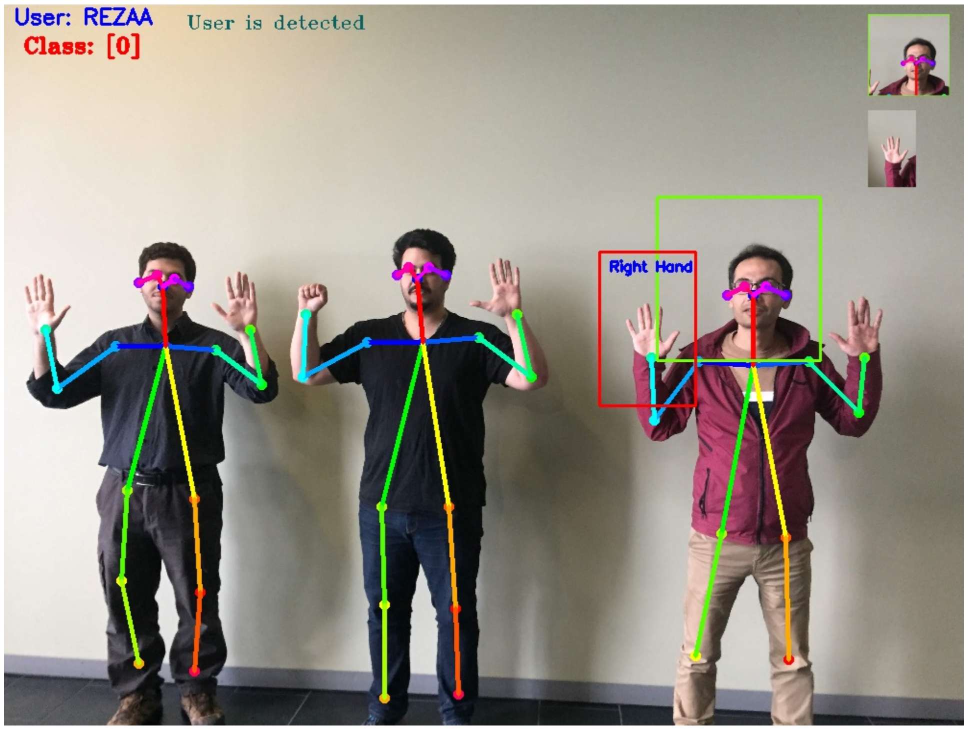


(f) Class #2: land

Figure 5.18: Gesture recognition on two (2) subjects including a main subject gesturing in front of the camera (series 5).

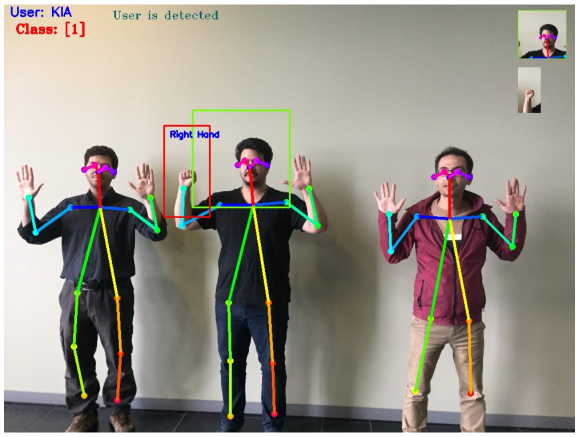


(a) Class #2: land

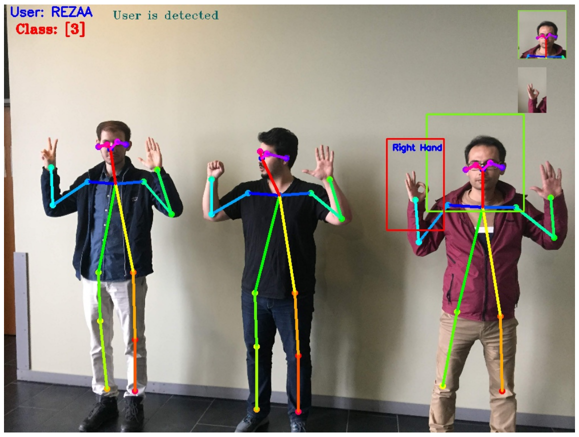


(b) Class #1: take off

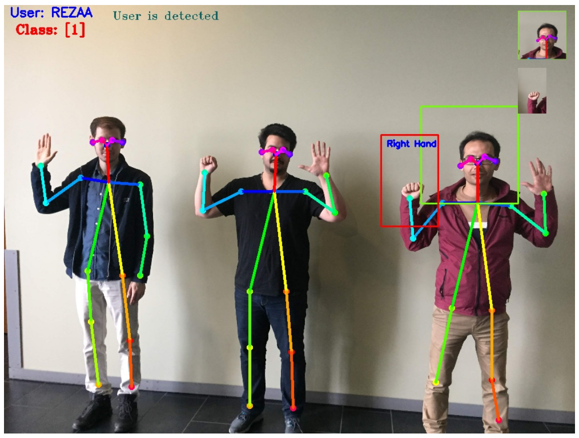
Figure 5.19: Gesture recognition on three (3) subjects including a main subject gesturing in front of the camera (series 1).



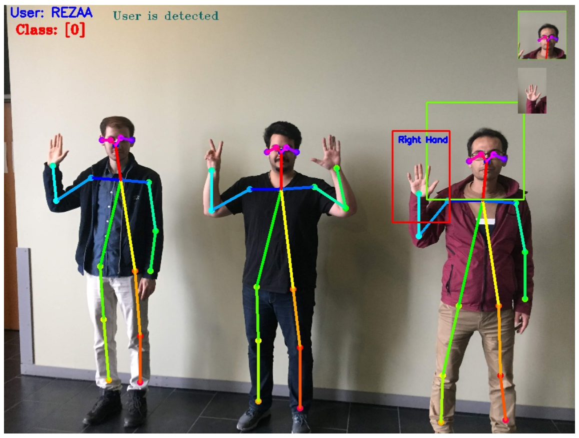
(a) Class #2: land



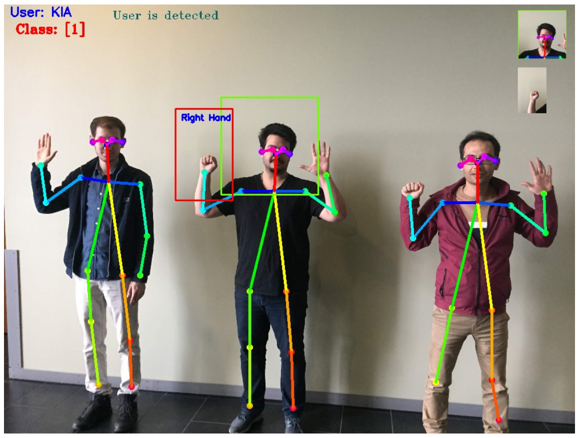
(b) Class #4: turn right



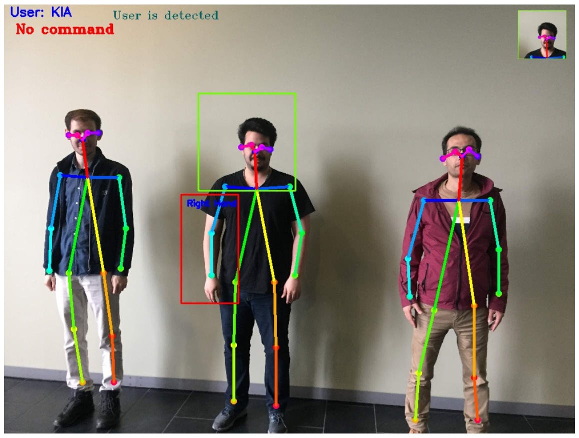
(c) Class #2: land



(d) Class #1: take off

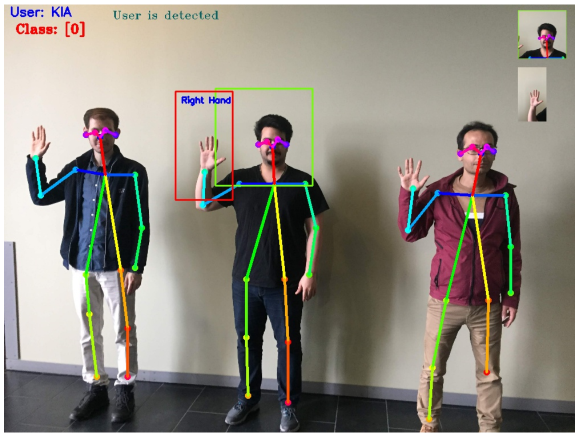


(e) Class #2: land

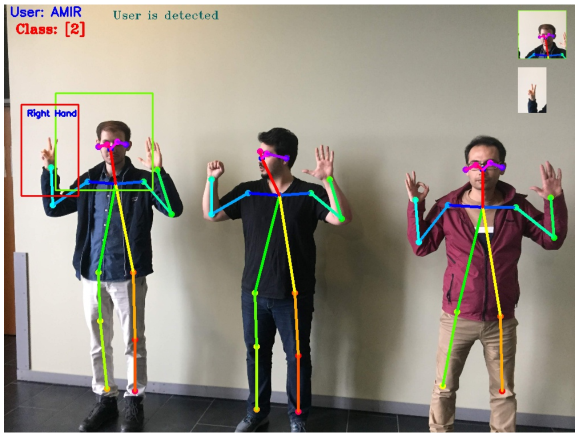


(f) No posture

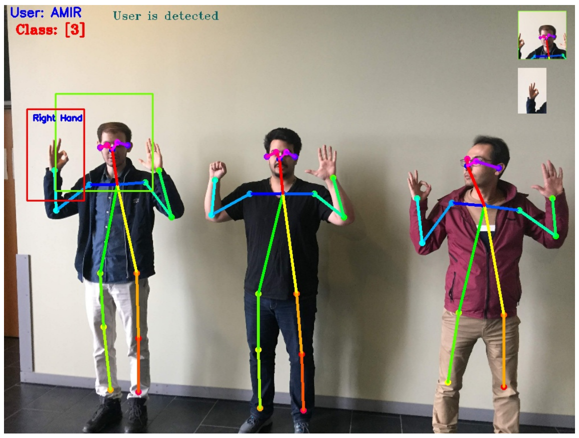
Figure 5.20: Gesture recognition on three (3) subjects including a main subject gesturing in front of the camera (series 2).



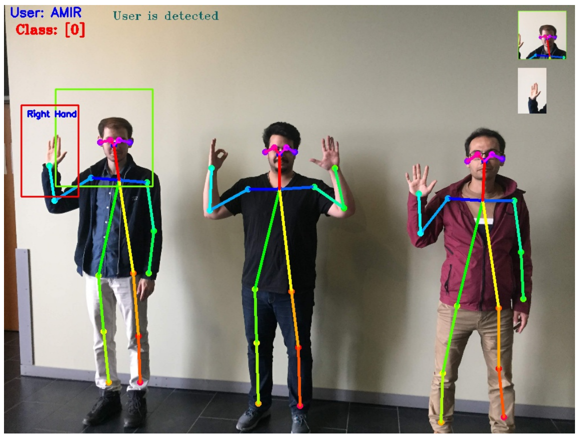
(a) Class #1: take off



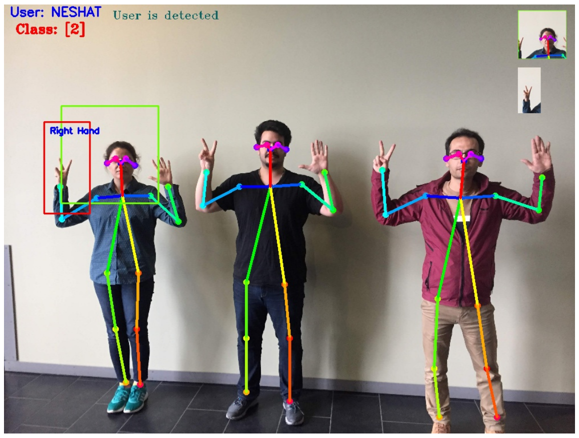
(b) Class #3: turn left



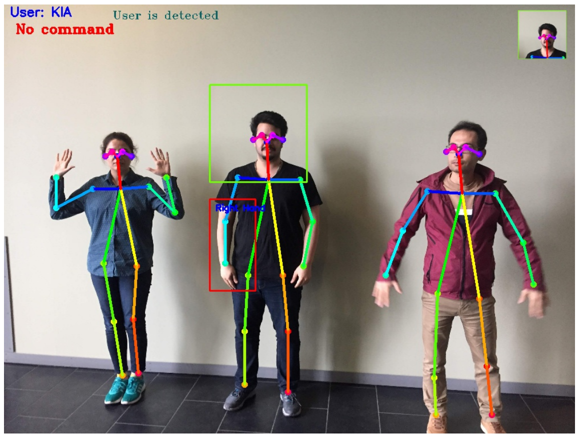
(c) Class #4: turn right



(d) Class #1: take off

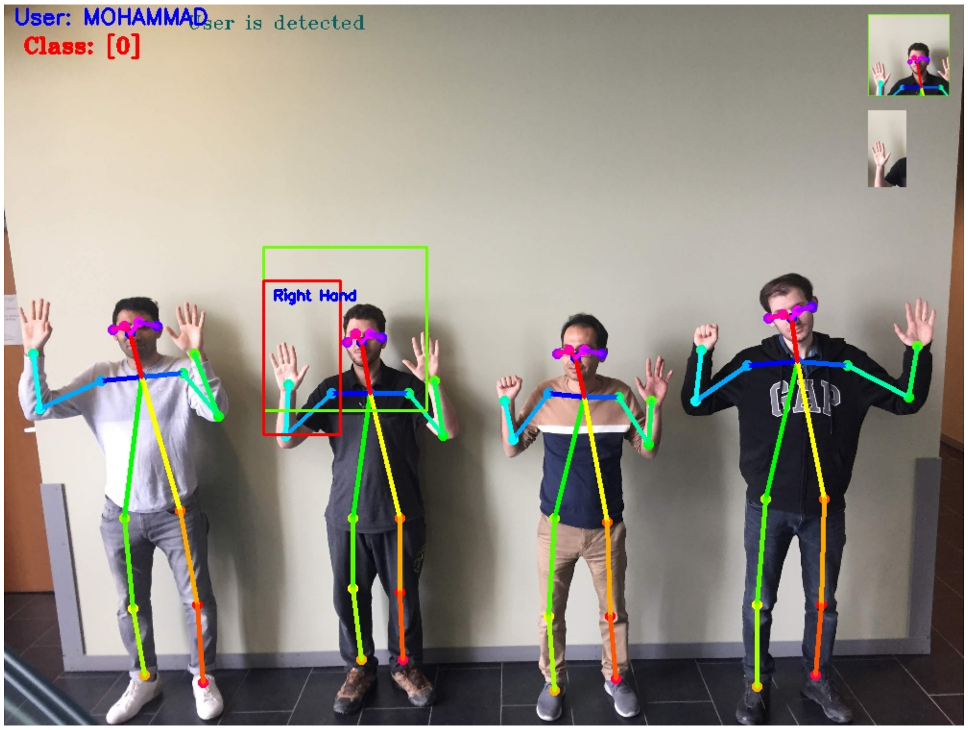


(e) Class #3: turn left

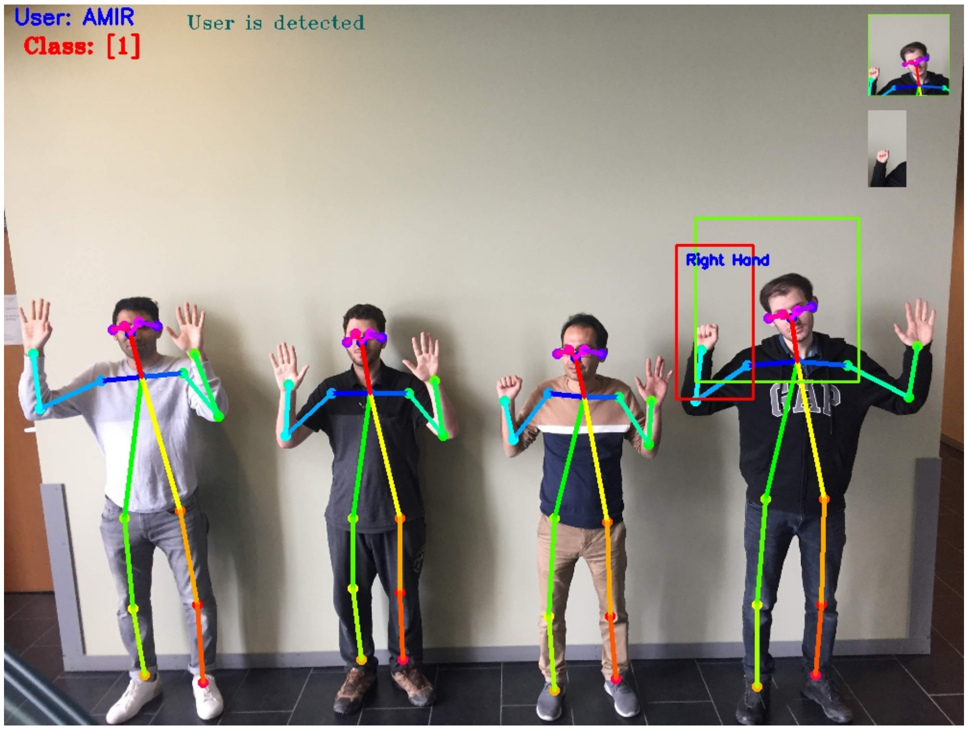


(f) No posture

Figure 5.21: Gesture recognition on three (3) subjects including a main subject gesturing in front of the camera (series 3).

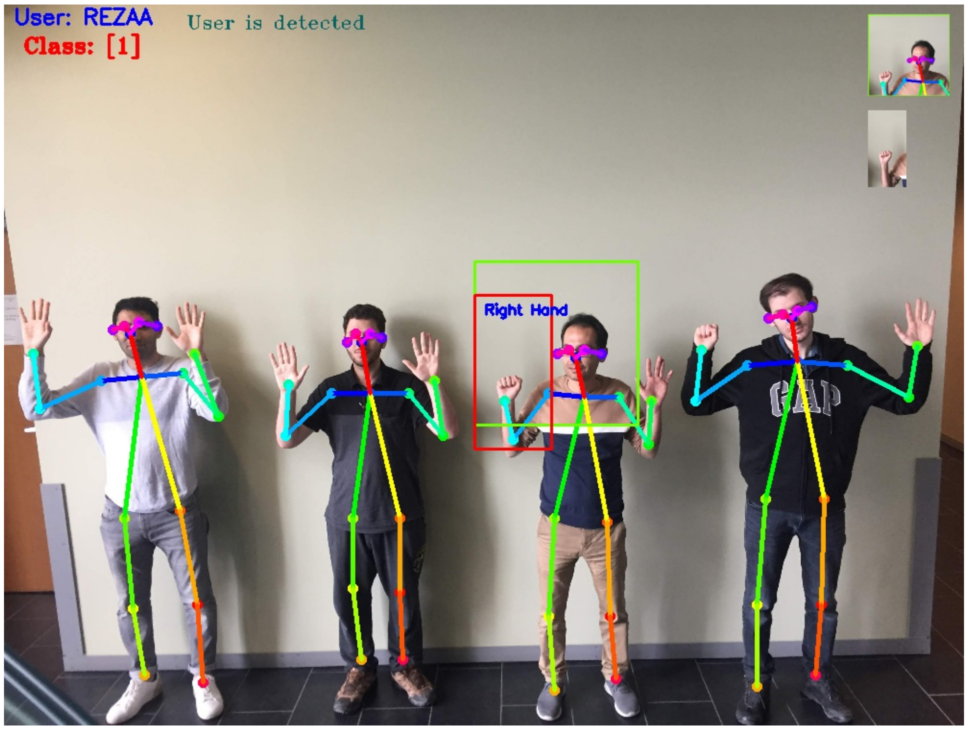


(a) Class #1: take off



(b) Class #2: land

Figure 5.22: Gesture recognition on group of four (4) subjects including a main subject performing in front of the camera (series 1).



(a) Class #2: land

Figure 5.23: Gesture recognition on group of four (4) subjects including a main subject performing in front of the camera (series 2).

Chapter 6

Conclusion and Future Work

We presented a gesture-based control system for multi-rotor aerial vehicles, which is able to detect and classify both right-hand and left-hand gestures in single- and multi-subject environments. The system is capable of operating with two different human-robot interaction subsystems: (i) a smart glove that produces continuous and discrete control commands and (ii) an image processing method that produces discrete commands using hand tracking and gesture recognition.

The proposed method has empowered with three separate modules for classifying the right-hand gesture in a single-subject environment: (i) a Hand Detection module which has been used to detect the hand RoI in the input frame and sent it to the classifier, (ii) a Gesture Classification module which is used to classify the detected hand in the received RoI from the hand detector module into one of the predefined classes, and (iii) a Retrainer module that retrains the running CNN model with the newly captured images and then obtains new weights to fit them to the running CNN model. The algorithm could distinguish between different scenarios when the user showed a posture correctly, hid his/her right hand, or held his/her right hand towards the ground.

The proposed method, also has four separate modules to classify the right-hand gesture in a multi-subject environment: (i) Skeleton Detection module that is capable of detecting all subjects present in the frame and sending their face regions to the facial recognition to detect the main subject, (ii) facial recognition module that captures one face image from the main subject, then uses that image to find the main subject among the received head regions from the Skeleton Detection module. The facial recognition module sends back the head ID of the main subject to the Skeleton Detection module to find the correct right-hand RoI. (iii) Gesture classifier which is used to classify the detected main subject right-hand RoI into one of the predefined classes, and (iv) Retrainer which retrains the running CNN model as mentioned in the single-subject environment.

To classify the left-hand gesture in both single- and multi-subject environments, a set of four flex sensors and an MPU sensor were implemented on a glove. Additionally, an SVM model has been deployed to classify the flex sensors output signals as a discrete command. After the SVM model classified the flex sensors outputs, the proposed algorithm used the exacts changes of the bending values and the MPU output signals as their continuous control command.

The proposed method had three separate validation modules to improve the system performances: (i) Gesture Classifier Validation that could retrain the model when the confidences were less than the threshold based on new data from a new user, (ii) multi-rotor aerial vehicles control validation that caused adjustments to the controller, and (iii) Human-in-the-Loop Validation that monitored the drone behavior.

Unlike the other algorithms and methods for posture detection and classification, the proposed method does not have limitations requiring a user to stand alone in front of the camera or to show just one hand to the camera. The proposed algorithm distinguished between subjects to find the main subject, then classified his/her right-hand gesture. The algorithm was studied on four different group populations of random subjects with various physical features.

The proposed control system is implemented for a multi-rotor aerial vehicle. However, it can be used for other controllable devices or any unmanned vehicles by redefining the gestures and some modifications for the targeted device.

6.1 Limitations

One of the limitations of the proposed method is the distance between the subjects and the camera. The subjects must be in the camera's field of view; otherwise the Skeleton Detection module can not detect all the subjects. The second limitation of the proposed method is the separation between the subjects. Since we are using 2D images, the Skeleton Detection module can not distinguish the subjects if there is no separation between the subjects. The third limitation of the proposed method is the lighting of the environment. Since the Skeleton Detection accuracy and Gesture Classification accuracy are based on the input frame, the proposed method might not be able to detect all the subjects and their gestures in poor lighting conditions.

6.2 Future Work

This project has many applications namely, control an unman aerial vehicle (UAV) or an unman ground vehicle (UGV) from a far distance where we can set a control station and the user can control the targeted robot through from the control station. As a part of the future work we want to add more wearable sensors such as:

- An inertial measurement unit (IMU) sensor will be embedded in the smart glove to measure the hand's acceleration and rotation accurately.
- An electromyography (EMG) sensor will be used on the user's arm to gather electrical signals generated by the user's muscles.

We will use these newly gathered data to train an SVM model to classify additional gestures and create additional commands for control purposes. We will develop a mapping system that helps the user to integrate the proposed algorithm with any controllable robot.

Furthermore, we will scale and develop this project as follows:

- Develop a gesture-based control system that will detect multiple main subjects in a multi-subject environment and classify their right/left-hands gestures. We will apply this multi-user feature to a multi-section robot (a robot with multiple hands, for instance) where each user can control a specific part of the robot, and the robot performs all the commands simultaneously.

- Develop smart glasses that have an in-built camera and integrate it into the proposed system. These smart glasses enable the user to perform the gestures in front of his/her face. The smart glasses can connect to other users to share the data and accomplish a multi-user task. We need to retrain the CNN model with the gestures from the back of the right hand. Nevertheless, the SVM model (smart glove) does not need any new training, and it will be connected to the glasses to send the data to the targeted UAV or UGV.

Bibliography

[1] K. K. Lekkala and V. K. Mittal, "Simultaneous aerial vehicle localization and human tracking," in *2016 IEEE Region 10 Conference (TENCON)*. IEEE, 2016, pp. 379–383.

[2] K. K. Lekkala and V. K. Mittal, "Accurate and augmented navigation for quadcopter based on multi-sensor fusion," in *2016 IEEE Annual India Conference (INDICON)*. IEEE, 2016, pp. 1–6.

[3] A. G. Perera, Y. Wei Law, and J. Chahl, "Uav-gesture: A dataset for uav control and gesture recognition," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[4] N. SaiChinmayi, C. Hasitha, B. Sravya, and V. Mittal, "Gesture signals processing for a silent spybot," in *2015 2nd International Conference on Signal Processing and Integrated Networks (SPIN)*. IEEE, 2015, pp. 756–761.

[5] Y. Ma, Y. Liu, R. Jin, X. Yuan, R. Sekha, S. Wilson, and R. Vaidyanathan, "Hand gesture recognition with convolutional neural networks for the multimodal uav control," in *2017 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*. IEEE, 2017, pp. 198–203.

[6] S.-Y. Shin, Y.-W. Kang, and Y.-G. Kim, "Hand gesture-based wearable human-drone interface for intuitive movement control," in *2019 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 2019, pp. 1–6.

[7] A. Asokan, A. J. Pothen, and R. K. Vijayaraj, "Armatron—a wearable gesture recognition glove: For control of robotic devices in disaster management and human rehabilitation," in *2016 International Conference on Robotics and Automation for Humanitarian Applications (RAHA)*. IEEE, 2016, pp. 1–5.

[8] D. Bannach, O. Amft, K. S. Kunze, E. A. Heinz, G. Troster, and P. Lukowicz, "Waving real hand gestures recorded by wearable motion sensors to a virtual car and driver in a mixed-reality parking game," in *2007 IEEE Symposium on Computational Intelligence and Games*. IEEE, 2007, pp. 32–39.

[9] M. Obaid, O. Mubin, C. A. Basedow, A. A. Ünlüer, M. J. Bergström, and M. Fjeld, "A drone agent to support a clean environment," in *Proceedings of the 3rd International Conference on Human-Agent Interaction*. ACM, 2015, pp. 55–61.

[10] T. Chouhan, A. Panse, A. K. Voona, and S. Sameer, "Smart glove with gesture recognition ability for the hearing and speech impaired," in *2014 IEEE Global Humanitarian Technology Conference-South Asia Satellite (GHTC-SAS)*. IEEE, 2014, pp. 105–110.

[11] K. Natarajan, T.-H. D. Nguyen, and M. Mete, "Hand gesture controlled drones: An open source library," in *Proceedings of the 1st International IEEE Conference on Data Intelligence and Security (ICDIS)*, Apr. 2018, pp. 168–175.

[12] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2016, pp. 21–37.

[13] K. Haratiannejadi, N. E. Fard, and R. R. Selmic, "Smart glove and hand gesture-based control interface for multi-rotor aerial vehicles," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2019, pp. 1956–1962.

[14] J. Hosang, R. Benenson, and B. Schiele, “Learning non-maximum suppression,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Jul. 2017, pp. 4507–4515.

[15] P. Zhu, J. Isaacs, B. Fu, and S. Ferrari, “Deep learning feature extraction for target recognition and classification in underwater sonar images,” in *Proceedings of the IEEE 56th Annual Conference on Decision and Control (CDC)*, Dec. 2017, pp. 2724–2731.

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

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, Jun. 2014.

[18] S. Cai, Y. Shu, W. Wang, M. Zhang, G. Chen, and B. C. Ooi, “Effective and efficient dropout for deep convolutional neural networks,” *arXiv preprint arXiv:1904.03392*, 2019.

[19] B. Karlik and A. V. Olgac, “Performance analysis of various activation functions in generalized mlp architectures of neural networks,” *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[20] Z. Zhang, “Improved adam optimizer for deep neural networks,” in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 2018, pp. 1–2.

[21] E. Dogo, O. Afolabi, N. Nwulu, B. Twala, and C. Aigbavboa, “A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks,” in *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*. IEEE, 2018, pp. 92–99.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[23] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 1, pp. 142–158, 2015.

[24] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[25] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[26] J. Zhang and L. Chen, "Clustering-based undersampling with random over sampling examples and support vector machine for imbalanced classification of breast cancer diagnosis," *Computer Assisted Surgery*, vol. 24, no. sup2, pp. 62–72, 2019.

[27] Z. Zhang, "Microsoft kinect sensor and its effect," *IEEE Multimedia*, vol. 19, no. 2, pp. 4–10, 2012.

[28] A. Sanna, F. Lamberti, G. Paravati, and F. Manuri, "A kinect-based natural interface for quadrotor control," *Entertainment Computing*, vol. 4, no. 3, pp. 179–186, 2013.

[29] G. A. Ten Holt, M. J. Reinders, and E. Hendriks, "Multi-dimensional dynamic time warping for gesture recognition," in *Proceedings of the Thirteenth Annual Conference of the Advanced School for Computing and Imaging*, 2007, pp. 23–32.

[30] J. Triesch and C. Von Der Malsburg, "A system for person-independent hand posture recognition against complex backgrounds," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 12, pp. 1449–1453, 2001.

[31] S. Bodiroža, H. I. Stern, and Y. Edan, "Dynamic gesture vocabulary design for intuitive human-robot dialog," in *Proceedings of the seventh annual ACM/IEEE International Conference on Human-Robot Interaction*, Mar. 2012, pp. 111–112.

[32] S. Bodiroža, G. Doisy, and V. V. Hafner, "Position-invariant, real-time gesture recognition based on dynamic time warping," in *Proceedings of the 8th ACM/IEEE International Conference on Human-Robot Tnteraction*, Mar. 2013, pp. 87–88.

[33] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011, pp. 1297–1304.

[34] T. Simon, H. Joo, I. Matthews, and Y. Sheikh, "Hand keypoint detection in single images using multiview bootstrapping," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1145–1153.

[35] A. Corradini, "Dynamic time warping for off-line recognition of a small gesture vocabulary," in *Proceedings of the IEEE ICCV Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems*, Jul. 2001, pp. 82–89.

[36] G. Doisy, A. Jevtić, and S. Bodiroža, "Spatially unconstrained, gesture-based human-robot interaction," in *Proceedings of the 8th ACM/IEEE International Conference on Human-Robot Interaction*, Mar. 2013, pp. 117–118.

[37] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 142–158, Jan. 2016.

[38] K. A. Bhaskaran, A. G. Nair, K. D. Ram, K. Ananthanarayanan, and H. N. Vardhan, "Smart gloves for hand gesture recognition: Sign language to speech conversion system," in *Robotics and Automation for Humanitarian Applications (RAHA), 2016 International Conference on*. IEEE, 2016, pp. 1–6.

[39] J. Bae, A. Larson, R. M. Voyles, R. Godzdanker, and J. Pearce, "Development and user testing of the gestural joystick for gloves-on hazardous environments," in *Robot*

and Human interactive Communication, 2007. RO-MAN 2007. The 16th IEEE International Symposium on. IEEE, 2007, pp. 1096–1101.

[40] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[41] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "Openpose: realtime multi-person 2d pose estimation using part affinity fields," *arXiv preprint arXiv:1812.08008*, 2018.

[42] S. Qiao, Y. Wang, and J. Li, "Real-time human gesture grading based on openpose," in *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. IEEE, 2017, pp. 1–6.

[43] P. J. Thilaga, B. A. Khan, A. Jones, and N. K. Kumar, "Modern face recognition with deep learning," in *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*. IEEE, 2018, pp. 1947–1951.

[44] O. Déniz, G. Bueno, J. Salido, and F. De la Torre, "Face recognition using histograms of oriented gradients," *Pattern Recognition Letters*, vol. 32, no. 12, pp. 1598–1603, 2011.

[45] A. Bulat and G. Tzimiropoulos, "How far are we from solving the 2d & 3d face alignment problem? (and a dataset of 230,000 3d facial landmarks)," in *International Conference on Computer Vision*, 2017.