UNIVERSITÀ DEGLI STUDI DI MILANO - BICOCCA
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea Magistrale in Informatica

A webcam-based human-computer interface for defining smooth 3D trajectories by tracking 2D hand landmarks

Relatore: Dr. Alessandro Giusti

Co-relatore: Dr. Loris Roveda

Co-relatore: Dr. Gianluigi Ciocca

Tesi di Laurea Magistrale di:
Umberto Cocca
Matricola 807191

Anno Accademico 2020-2021

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Umberto Cocca
Lugano, 23 February 2022

-dedica-

“Gutta cavat lapidem.”
lat. «la goccia scava la pietra»

Abstract

Robotic systems are increasingly being adopted for filming and photography purposes. By exploiting robots, in fact, it is possible to program complex motions for the camera, achieving high-quality videos/photographs. Our contribution is to propose an alternative to the joystick: being able to control a drone using just a hand and giving space to new ways of expressing video content, even to those who are novices. To achieve this, we capture 3D movements (using a hand tracking system) after building a hand gesture recognition and setting up a solid pipeline to detect 3D trajectories obtained from 2D pixel landmarks, hence estimating the orientation and distance from the camera. Trajectories are interpolated and smoothed with ridge regression. As proof, the captured trajectory was launched first on a drone, in simulation, implemented in the ROS framework, and later on a real drone called Dji Ryze Tello. The whole pipeline can be easily translated into another kind of task, e.g. interaction and communication in AR / VR.

Acknowledgements

Working on this thesis was an experience that enriched me. I was able to work on a big project that allowed me to put into practice most of the skills acquired in these years of university. I understood how complicated it is to take small steps forward in any area of knowledge day after day.

I am glad I worked on this thesis and I couldn't have asked for better regarding the support of my advisors: Dr. Alessandro Giusti, Dr. Loris Roveda and Dr. Gianluigi Ciocca, they consistently allowed this paper to be my own work but steered me in the right direction whenever they thought I needed it.

Finally, I must express my very profound gratitude to my parents, my girlfriend Mariagrazia Capano and my closest friends Andrea Bonfanti, Salvatore Scilipoti, Guido Piscopo, Giada Maino, Claudia Crimella, Silvia Traversa and Ivo Bettini for providing me with unfailing support and continuous encouragement through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Contents

Figures

Tables

Equations

Listings

Introduction

Chapter 1

Literature review

This chapter discusses previous research about the topic, providing a brief introduction to the approach we adopted.

Chapter 2

Background

This chapter provides some background concepts to understand the material presented in this thesis.

2.1 Regression

The goal of regression is to predict the value of one or more continuous target variables t given the value of a D-dimensional vector x of input variables. [Bishop, 2006]
Given a training data set comprising N observations x_n, where $n = 1, ..., N$, together with corresponding target values t_n, the goal is to predict the value of t for a new value of x.
From a probabilistic perspective, we aim to model the predictive distribution $p(t|x)$ because this expresses our uncertainty about the value of t for each value of x.

2.1.1 Linear Models for Regression

The simplest linear model for regression is one that involved a linear combination of the input variables:

$$y(x, w) = w_0 + w_1 x_1 + ...w_D x_D \tag{2.1}$$

where $x = (x_1, ..., x_D)^T$. This is often simply known as linear regression. The key property of this model is that it is a linear function of the parameters w_i, but also of x_i and establishes significant limitations on the model. It is possibile extend the class of models by considering linear combinations of fixed nonlinear functions of the input variables:

$$y(x,w) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(x) \tag{2.2}$$

where $\phi_j(x)$ are known as basis functions. The total number of parameters in this model will be M.
The parameter w_0 is called bias parameter. It is often convenient to define an additional dummy "basis function" $\phi_0(x) = 1$, so that:

$$y(x,w) = \sum_{j=0}^{M-1} w_j \phi_j(x) = w^T \phi(x) \tag{2.3}$$

Equation 2.1. By using non linear basis functions, we allow the function $y(x,w)$ to be a non linear function of the input vector x.

where $w = (w_0, ..., w_{M-1})$ and $\phi = (\phi_0, ..., \phi_{M-1})^T$.

A particle example of this model where there is a single input variable x is the polynomial regression. The basis functions take the form of powers of x so that $\phi_j(x) = x^j$.
There are other possibile choices for the basis functions as:

$$\phi_j(x) = e^{\frac{-(x-u_j)^2}{2s^2}} \tag{2.4}$$

Equation 2.2. These are usually referred to as "Gaussian" basis functions.

where u_j regulates the locations of the basis functions in input space, while the parameter s is their spatial scale. We can use simply the identity basis functions in which the vector $\phi(x) = x$.

2.1.2 Normal equations

The values of the coefficients will be determined by fitting the polynomial to the training data. This can be done by minimizing an error function that measures the misfit between the function $y(x,w)$, for any given value of w, and the training set data points. One simple choice of error function, which is widely used, is given by the Sum of Squared estimate of Errors (SSE) between the predictions $y(x_n,w)$ for each data

point x_n and the corresponding target values t_n (called also sum of squares regression [Valchanov, 2018]), so that we minimize:

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}[y(x_n, w) - t_n]^2 \tag{2.5}$$

Equation 2.3. Is a statistical technique used in regression analysis to determine the dispersion of data points and the function that best fits (varies least) from the data.

where the factor of $1/2$ is included for mathematical convenience. It is a nonnegative quantity that would be zero if, and only if, the function $y(x, w)$ were to pass exactly through each training data point.

We can solve the curve fitting problem by choosing the value of w for which $E(w)$ is as small as possible. Because the error function is a quadratic function of the coefficients w, its derivatives with respect to the coefficients will be linear in the elements of w, and so the minimization of the error function has a unique solution w^*. The resulting polynomial is given by the function $y(x, w^*)$.

We can write (2.5) in matrix notation as:

$$\frac{1}{2}(\phi w - t)^T(\phi w - t) \tag{2.6}$$

Equation 2.4. This is the sum of squares regression in matrix notation.

where ϕ is an NxM matrix, so that:

$$\phi = \begin{pmatrix} \phi_0(x_1) & \phi_1(x_1) & \dots & \phi_{M-1}(x_1) \\ \phi_0(x_2) & \phi_1(x_1) & \dots & \phi_{M-1}(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \dots & \phi_{M-1}(x_N) \end{pmatrix} \tag{2.7}$$

Equation 2.5. This is called the design matrix whose elements are given by $\phi_{nj} = \phi_j(x_n)$.

Therefore, we have to solve the optimization problem finding the minimum of the cost function $E(w)$:

$$w = \begin{pmatrix} w_0 \\ \vdots \\ w_{M-1} \end{pmatrix} \quad t = \begin{pmatrix} t_1 \\ \vdots \\ t_N \end{pmatrix} \tag{2.8}$$

Equation 2.6. Weights of $M-1$ features and target of N examples.

$$w^* = \arg\min_w \frac{1}{2}(\phi w - t)^T(\phi w - t) \tag{2.9}$$

Equation 2.7. The goal is to find the weight w that minimize the cost function $E(w)$.

So we compute the gradient and solving for w we obtain:

$$\begin{aligned} \nabla E(w) = \phi^T(\phi w - t) &= 0 \\ \phi^T \phi w - \phi^T &= 0 \\ \phi^T \phi w &= \phi^T t \\ w^* &= (\phi^T \phi)^{-1} \phi^T t \end{aligned} \tag{2.10}$$

Equation 2.8. They are known as the normal equations for the least squares problem

This is the real minimum because if we take the second derivative $\nabla^2 E(w) = \phi^T \phi$ and this is a symmetric matrix, so it's also positive definitive matrix, which means that this objective function that we try to minimize is convex. So if we find a stationary point, such that derivative is zero, we also find a global minium. Furthermore, to find a solution we need to invert this matrix $\phi^T \phi$, so we need some condition that assure this is invertible and this is the case when the columns of the matrix are linearly independent.

Once we find the solution w^* and we receive a new data point that we never seen during training, we just predict the new target t^* as:

$$t^* = w^{*^T} \phi(x) \tag{2.11}$$

Equation 2.9. The goal is to find the weight w that minimize the cost function $E(w)$.

2.2 The Problem Of Overfitting

If we use a set of features that is too expressive, for example a ten polynomial grade (x^{10}), then this interpolates very close our training data, because we have a model with 10 parameters where we can perfectly represents the data points. The risk is that the model overfit data points (see Fig. 2.1 (center)).

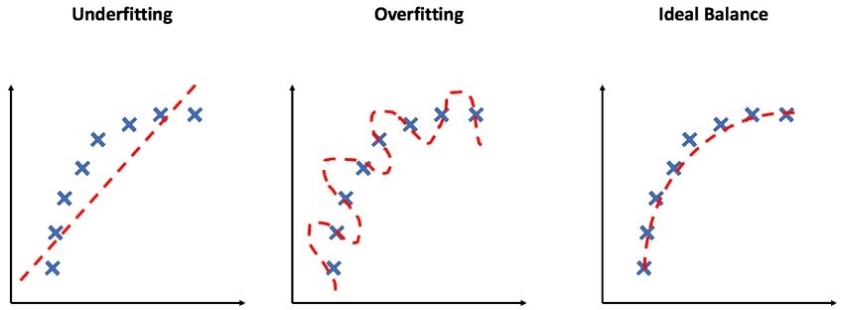


Figure 2.1. Image by "Applied Supervised Learning with R" book [a]. We can see high bias resulting in an oversimplified model (that is, underfitting); high variance resulting in overcomplicated models (that is, overfitting); and lastly, striking the right balance between bias and variance.

[a] https://subscription.packtpub.com/book/data/9781838556334/7/ch07lvl1sec82/underfitting-and-overfitting

On the other hand, if our model is not too expressive and not too complex our data will be linearly representable in the feature space and this means that the performance will be very poor (see Fig. 2.1 (left)). We want a trade-off between fits data and being able to generalize (see Fig. 2.1 (center)).

2.2.1 Regularization

We want to penalize for some features with parameters with high values, if our model is overfitting is very likely that the parameters of our model will have a big magnitude, this means that also the features supply to this parameters will be higher and very low and this cause a lot of problems.

In order to control over-fitting we introduce the idea of adding a regularization term to an error fuction, so that the total error function to minimized takes the form:

$$E(w) = E_D(w) + \lambda E_W(w) \tag{2.12}$$

Where λ is the regularization coefficient and it is the trade-off between how we well fit training set and how to establish the parameters w with low values, therefore having simple hypotesys avoiding over-fitting. E_D is the error based on dataset, while E_W is based on weights.

2.2.2 Cost Function

One of the simplest forms of regularizer is given by the sum-of-squares of the weight vector elements:

$$E_W(w) = \frac{1}{2} w^T w = \frac{||w||_2^2}{2} \tag{2.13}$$

Equation 2.10. Cost function | Regularisation term.

where $||w||_2$ is the euclidean norm $\sqrt{\sum_{i=1}^{n} x_i^2}$.

This is also called ridge regression, a method of estimating the coefficitents of multiple-regression models in scenarios where indipendent variables are highly correlated.
If we also consider the sum-of-squares error function given by:

$$E_D(w) = \frac{1}{2}[t_n - w^T \phi(x_n)]^2 \tag{2.14}$$

then the total error function becomes:

$$E(w) = \frac{1}{2} \sum_{n=1}^{N} [t_n - w^T \phi(x_n)]^2 + \frac{\lambda}{2} w^T w \tag{2.15}$$

Equation 2.11. This particular choice of regularizer is known in the machine learning literature as weight decay because in sequential learning algorithms, it encourages weight values to decay towards zero.

Setting the gradient of $E(w)$ w.r.t. w to zero, and solving for w, we obtain:

$$w^* = (\lambda I + \phi^T \phi)^{-1} \phi^T t \tag{2.16}$$

Equation 2.12. This is an extension of the least-squares solution 2.10.

From this side it denotes a better version than before because is also possibile prove that $(\lambda I + \phi^T \phi)$ is always invertible if $\lambda > 0$, therefore w^* always exists.

2.2.3 (Batch) Gradient Descent

Gradient descent is a first-order iterative optimization algorithm for finding the minimum w of a function $E(w)$. To achieve this goal, it performs two steps iteratively, until convergence:

- Compute the slope (gradient) that is the first-order derivative of the function at the current point
- Move-in the opposite direction of the slope increase from the current point by the computed amount

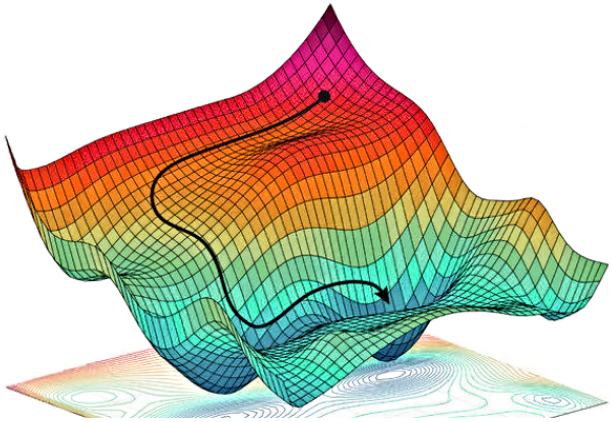


Figure 2.2. Gradient descent is based on the observation that if the multi-variable function $E(w)$ is defined and differentiable in a neighborhood of a point w_0 , then w_0 decreases fastest if one goes from w_0 in the direction of the negative gradient of $E(w)$ at w_0, $-\nabla E(w_0)$.

Let's now using Mean Squared Error (MSE), instead of using (2.5) due to the fact that we are able to reach obvious benefits: first of all to keep the value in a expressible range usable by computers, then to make the results comparable across samples regardless of the size of the sample. Infact, the SSE depends on how many terms are added up (note the case of millions/billions of data points). In addition, using SSE or MSE it still leads to find an equivalent solution.

$$E_D(w) = \frac{1}{2N} \sum_{n=1}^{N} [y(x_n, w) - t_n]^2 \tag{2.17}$$

Equation 2.13. Mean Squared Error. $1/2$ is added, as in SSE, so the derivative doesn't need a constant out front. We get away with it because the minima of $E_D(w)$ and $E_D(w)/2$ are achieved at the same value(s) of w.

Concerning the regularisation term $E_W(w)$ (2.13), we could also write:

$$E_W(w) = \frac{\sum_{m=1}^{M} w_m^2}{2} \tag{2.18}$$

Equation 2.14. Regularisation term.

Note that, conventionally, m starts from 1, and not from 0, even if it exists (2.8). Nevertheless, it's important to know that nothing change consistently even if we consider the 0th weight.

Combining together following (2.12)

$$E(w) = \frac{1}{2N} \left\{ \sum_{n=1}^{N} [y(x_n, w) - t_n]^2 + \lambda \sum_{m=1}^{M} w_m^2 \right\} \tag{2.19}$$

Equation 2.15. Gradient Descent.

Have some function $E(w)$
Want $\arg\min_w E(w)$)

Keep changing w to reduce $E(w)$ until we hopefully end up at a minimum:

$$
\begin{aligned}
&Repeat = \{ \\
&\quad w_0 := w_0 - \alpha \frac{1}{N} \sum_{n=1}^{N} [y(x_n, w) - t_n]^2 x_0 \\
&\quad w_j := w_j - \alpha \frac{1}{N} \sum_{n=1}^{N} [y(x_n, w) - t_n]^2 x_j + \frac{\lambda}{M} w_j \\
&\quad \}
\end{aligned}
\tag{2.20}
$$

Equation 2.16. Cost function for Gradient Descent in compact form.

where in compact form is:

$$
\begin{aligned}
&Repeat = \{ \\
&\quad w_j := w_j (1 - \alpha \frac{\lambda}{N}) - \alpha \frac{1}{N} \sum_{n=1}^{N} [y(x_n, w) - t_n]^2 x_j \\
&\quad \}
\end{aligned}
\tag{2.21}
$$

Equation 2.17. Cost function for Gradient Descent.

2.3 Artificial Neural Networks

The term Neural Network (NN) has its origins in attempts to find mathematical representations of information processing in biological systems. In fact, Artificial Neural Networks (ANNs), subset of Machine Learning (ML) field, are models inspired from the biological performance of human brain [Andina et al., 2007].

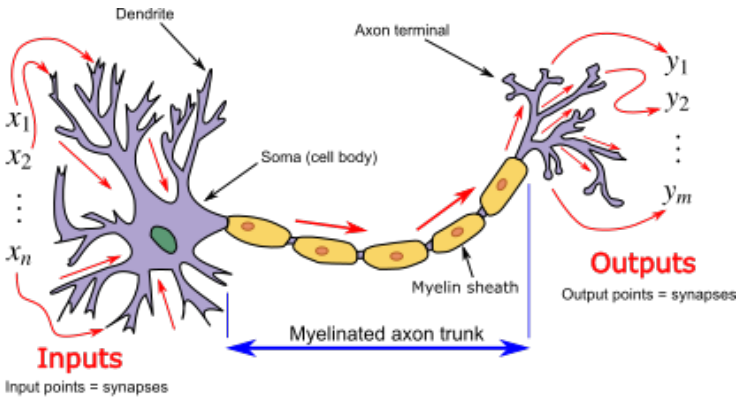


Figure 2.3. Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals. The signal represents a short electrical pulse called 'spike'.

Neurons have cel body (fig. 2.3) and a number of input wires called dendrites. Neurons also have an output wire called axon, used to send signals to other neurons. At a simplistic level the neuron is a computational unit that gets a number of inputs through its input wires, then does some computation and finally it sends outputs to other neurons connected to it in the brain.

2.3.1 Feedforward Fully-Connected Neural Networks

In (fig. 2.4) is visible a NN, seen as mathematical model. It's just a group of this different neurons strung together. Trying to underline an analogy with the biological systems: circles identify the cell body where they are feeded with some inputs that pass through the input wires, similar to the dendrites. The neuron does some computation and outputs some value on an output wire, where in the biological neuron it identifies the axon.

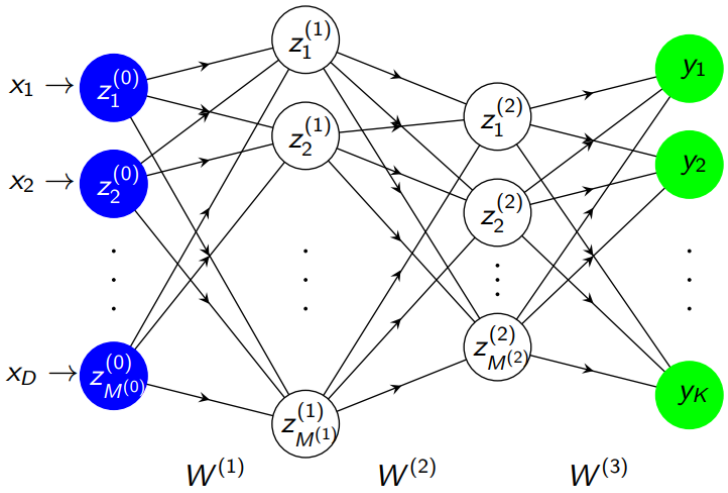


Figure 2.4. The image graphically shows a NN with three layers (or two hidden layers). The input layer has $M^{(0)}$ neurons and the output layer has K outputs. Neurons are organized in layers to allow parallel copmutation to avoid cyclic dependendices. The process of computing NN outputs from inputs is called forward propagation. This kind of network is also called Multi-layer perceptron.

The $z_m^{(l)}$ are neurons, each takes its input values and computes a single output value from them. Neurons are organized in layers $1, ..., L$ and usually the starting input is considered the 0th layer. Inputs $x_1, ..., x_D$ are occasionally called input layer/neurons (even though they do not compute anything). The output of the entire network is then $y = z^{(L)}$, called output layer. Instead, the internal layers are called hidden layers. Each layer $l \in \{1, ..., L\}$ has $M^{(l)}$ neurons.
$M^{(1)}$ neurons perform a preceptron-like computation:

$$u_m^{(1)} = (w_m^{(1)})^T x + b_m^{(1)}, \qquad z_m^{(1)} = f(u(1)_m), \qquad m = 1, ..., M^{(1)} \tag{2.22}$$

with a differentiable activation function f for gradient descent. This step is iterated multiple times taking the outputs of the previous step:

$$z^{(l-1)} = (z_m^{(l-1)})_{m=1,...,M^{(l-1)}} \tag{2.23}$$

as input of:

$$u_m^{(l)} = (w_m^{(l)})^T z^{(l-1)} + b_m^{(l)}, \qquad z_m^{(l)} = f(u_m^{(l)}) \tag{2.24}$$

where $m = 1, ..., M^{(1)}$ and $l = 2, ..., L$. Weights w are usually indendent for each step. Additionally define $z^{(0)}$ to be the input, i.e.

$$z^{(0)} = x \tag{2.25}$$

For each layer $l \in 1, ..., L$ the computation is

$$z_m^{(l)} = f((w_m^{(l)})^T z^{(l-1)} + b_m^{(l)}) \tag{2.26}$$

which can be written as a matrix multiplication:

$$z^{(l)} = f(W^{(l)} z^{(l-1)} + b^{(l)}) \tag{2.27}$$

Equation 2.18. Function that idetifies input transformation at each step l of the net.

The weights w are directed connections between the neurons, e.g. the neurons of layer 2 are connected to the ones of layer 1 by the weights $w_{mn}^{(2)}, m = 1, ...M^{(1)}, n = 1, ..., M^{(2)}$. The bias b varies according to the propensity of the neuron to activate, influencing its output.

$f()$ is an activation functions and need to be differentiable, because we wish to apply gradient descent training. For the hidden layers of the nework, the activation function must be nonlinear, because multiple linear computations can be collapsed to a single one, therefore in order to gain power from iterative computation we thus need nonlinear steps.
Many possible activation functions for the hidden layers of a NN exist:

- Sigmoid, Hyperbolic Tangent: Monotonic, squeeze output to a fixed range
- ReLU: "almost linear" (a clipped identity function)

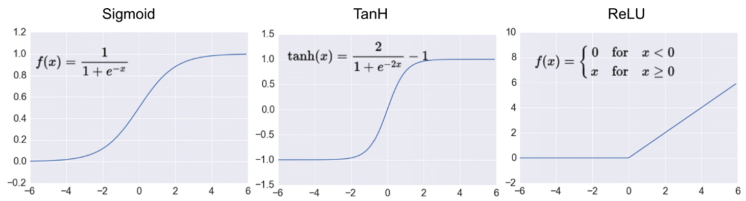


Figure 2.5. Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals. The signal represents a short electrical pulse called 'spike'.

One of the key characteristics of modern deep learning system is to use non-saturated activation function (e.g. ReLU) to replace its saturated counterpart (e.g. sigmoid, tanh). The advantage of using non-saturated activation function lies in two aspects: the first is to solve the so called "exploding/vanishing gradient" [Bengio et al., 1994], in particular on the difficulty of training Recurrent Neural Network (RNN) [Pascanu et al., 2013], while the second is to accelerate the convergence speed. More sophisticated activation function as the "leaky ReLU" trying to solve the dying ReLU problem [Lea]. In contrast to ReLU, in which the negative part is totally dropped, leaky ReLU assigns a noon-zero slope to it. Leaky ReLU and its variants are consistently better than ReLU in Convolutional Neural Network (CNN) [Xu et al., 2015].

Leaky Rectified linear activation is introduced in acoustic model [Maas et al.]. Mathematically, it is defined as follows:

$$f(x) = \begin{cases} \frac{x}{a} & \text{if } x < 0 \\ x & \text{otherwise} \end{cases} \tag{2.28}$$

Equation 2.19. Function that idetifies input transformation at each step l of the net.

where a is a fixed parameter in range $(1; +\text{inf})$. In original paper, the authors suggest to set a to a large number like 100.

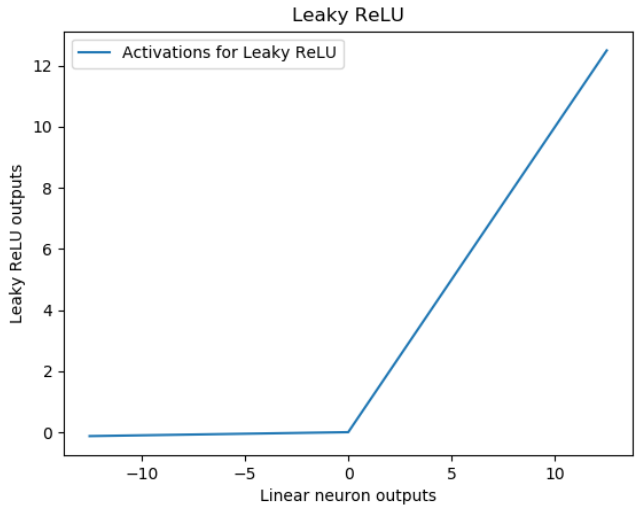


Figure 2.6. Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals. The signal represents a short electrical pulse called 'spike'.

2.3.2 NN Setup for Classification

For a classification task with K classes, we use a K-idmendional output layer. A sample $x \in R^D$ is classified as belonging to class k if the output neuron y_k has the maximal value:

$$c^* = \arg\max_k y_k \tag{2.29}$$

The problem is that the $argmax$ function is not differentiable. Therefore, this is solved by letting the NN output a probability distribution over classes:

$$y = (y_k)_{k=1,\dots,K} \qquad y_k \geq 0, \quad \sum_k y_k = 1 \tag{2.30}$$

The advantage is that we can derive a differentiable measure of the quality of the output on theoretical grounds, using probability theory. In order to make the network output a probability distribution, we take exponentials and normalize. This is the softmax nonlinearity:

$$S(y) = (\frac{e^{y_1}}{\sum_k e^{y_k}}, ..., \frac{e^{y_K}}{\sum_k e^{y_k}}) \tag{2.31}$$

Equation 2.20. Softmax nonlinearity.

In contrast to other activation functions, it is applied to the full last layer of the NN, not to each indipendent component. The hidden layers can have any nonlinear activation function.

The learning process is structured as a non-convex optimisation problem in which the aim is to minimise a cost function, which measures the distance between a particular solution and an optimal one.

If we assume a NN with softmax output, we can compute the loss by measuring the cross-entropy between the output distribution and the target distribution.

Encoding the target in one-hot style, e.g. if a sample belongs to class k, the target is:

$$t = (0, ..., 0, 1, 0, ..., 0) \tag{2.32}$$

We treat this as a probability distribution: in an ideal world, a perfect hypothesis y would exactly match this t, assigning probability 1 to the correct class, and probability 0 otherwise. The cross-entropy loss is defined as:

$$E_{CE} = -\sum_k (t_k \log y_k) \tag{2.33}$$

Equation 2.21. Cross-entropy loss.

The intuition about the cross-entropy corresponds to the number of additional bits needed to encode the correct output, given that we have access to the (possibly wrong) prediction of the network. One property of the cross-entropy loss is that is always non-negative. For efficiency and numerical stability, one should merge softmax loss and cross-entropy criterion into one function:

$$E_{CE+SM} = -\sum_{k}(t_k \log S_k(y)) \tag{2.34}$$

Equation 2.22. To train the network with backpropagation, you need to calculate the derivative of the loss. In the general case, that derivative can get complicated, but using the softmax and the cross entropy loss, that complexity fades away.

2.3.3 Optimisation algorithms

The choice of optimization algorithms strongly influence the effectiveness of the learning process as they update and calculate the appropriate and optimal values of that model. Specifically if we consider the gradient descent, the most popular optimization strategy used in machine learning, the extent of the update is determined by the learning rate λ, which guarantees convergence to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces. By the way, there are better optimization as the non linear conjugate gradient [Hager and Zhang, 2006], BFGS, the improved version to decrease memory usage L-BFGS [Saputro and Widyaningsih, 2017], etc... the main advantages are that no need to manually pick λ and often are faster than gradient descent, although more complex algorithms.

The optimiser chosen for this thesis project is Adam, an algorithm for first-order gradient-based optimisation of stochastic objective functions, based on adaptive estimates of lower-order moments [Kingma and Ba, 2017]. It is an extension to stochastic gradient descent that has recently seen broader adoption for Deep Learning (DL) applications in computer vision and Natural Language Processing (NLP).

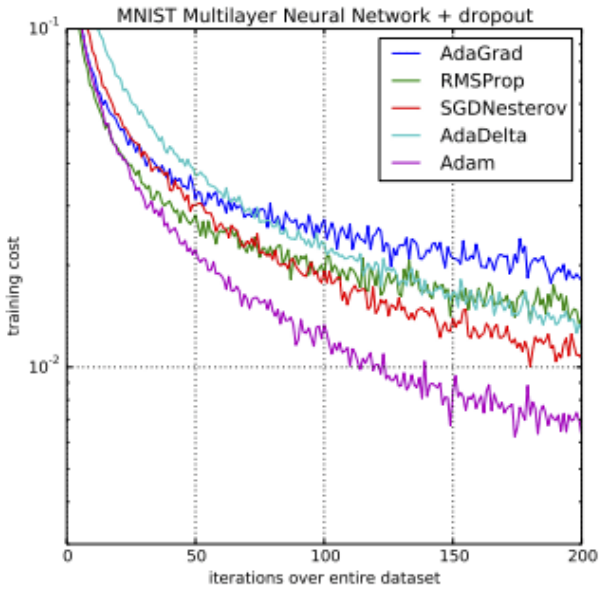


Figure 2.7. Comparison of Adam to other optimization algorithms training a Multilayer Perceptron[a].

[a]Taken from Adam: A Method for Stochastic Optimization, 2015

Chapter 3

Tools

This chapter introduces the tools used in this work, starting from the description of the target platform in, then describing the simulator in and finally mentioning the frameworks used for the implementation in.

3.1 Tello

Tello is a small quadcopter that features a Vision Positioning System and an onboard camera. Using its advanced flight controller, it can hover in place and is suitable for flying indoors. Tello caputers 5MP photos and stream until 720p live video. Its maximum flight time is approximately 12 minutes (tested in windless conditions at a consistent $15km/h$) and its maximum flight distance id $100m$ [Tech., 2018].

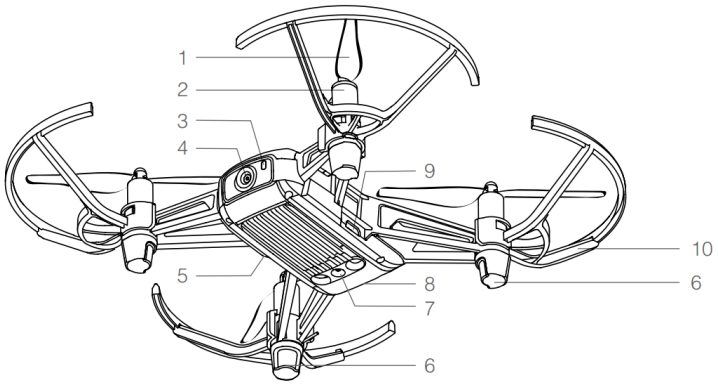


Figure 3.1. 1.Propellers; 2.Motors; 3.Aircraft Status Indicator; 4.Camera; 5.Power Button; 6.Antennas; 7.Vision Positioning System; 8.Flight Battery; 9.Micro USB Port; 10.Propeller Guards.

The Tello can be controlled manually using the virtual joysticks in the Tello app or using a compatible remote controller. It also has various Intelligent Flight Modes that be used

to make Tello perform maneuvers automatically. Propeller Guards can be used to reduce the risk of harm or damage people or objects resulting from accidental collisions with Tello aircraft.

3.1.1 Tello Command Types and Results

The Tello SDK connects to the aircraft through a Wi-Fi UDP port, allowing users to control the aircraft with text commands. There are Control and Set commands where return "ok" if the command was successful or "error" or an informational result code if the ocmmand failed. There are also Read commands that return the current value of the sub-parameters.

Main Tello Commands		
Command	**Description**	**Possible Response**
connect	Enter SDK mode.	ok / error
streamon	Turn on video streaming.	
streamoff	Turn off video streaming.	
takeoff	Auto takeoff.	
land	Auto landing.	
send_rc_control	Set remote control via four channels. Arguments: - left / right velocity: from -100 to +100 - forward / backward velocity: from -100 to +100 - up / down: from -100 to +100 - yaw: from -100 to +100	
get_battery	Get current battery percentage	from 0 to +100

Table 3.1. List of the main Tello functions of the python wrapper to interact with the Ryze Tello drone using the official Tello api.

3.2 Gazebo

Gazebo is a 3D simulator, offers the ability to accurately and efficiently simulate robots in complex indoor and outdoor environments. Thanks to Gazebo it was possible to launch the 3D trajectory acquired by hand through the webcam on a simulated drone.

3.3 Frameworks

DJITelloPy DJI Tello drone python interface using the official Tello SDK and Tello EDU SDK. This library has an implementation of all tello commands, easily retrieve a video stream, receive and parse state packets and other features.[1].

TensorFlow is an end-to-end open source platform for ML. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML[2].

NumPy is a highly optimized library for scientific computing that provides support for a range of utilities for numerical operations with a MATLAB-style syntax. manipulation[3].

OpenCV-Python OpenCV-Python is a library of Python bindings designed to solve computer vision problems. Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation. It makes use of Numpy.[4].

Robot Operating System is an open-source robotics middleware suite. It provides high-level hardware abstraction layer for sensors and actuators, an extensive set of standardized message types and services, and package management.[5].

Pandas is an open source library providing high-performance, easy-to-use data structures and data analysis tools[6].

Matplotlib is a comprehensive package for creating static, animated, and interactive visualisations in Python[7].

Seaborn Seaborn Python is a data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive statistical graphics. Because seaborn python is built on top of Matplotlib, the graphics can be further tweaked using Matplotlib tools and rendered with any of the Matplotlib backends to generate publication-quality figures.[8].

[1] https://github.com/damiafuentes/DJITelloPy
[2] https://www.tensorflow.org/
[3] https://numpy.org
[4] https://docs.opencv.org/4.x/index.html
[5] https://www.ros.org/
[6] https://pandas.pydata.org
[7] https://matplotlib.org
[8] http://seaborn.pydata.org/

`scikit-learn` is an open source package that provides simple and efficient tools for predictive data analysis, built on `NumPy`, `Scipy`, and `Matplotlib`[9].

[9] `https://scikit-learn.org`

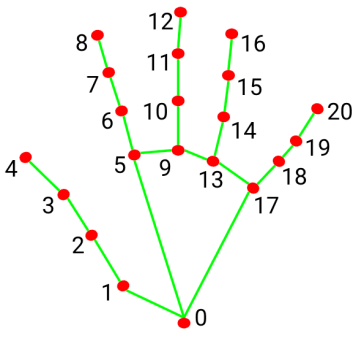
Chapter 4

Methodologies

This chapter illustrates the methodology used to approach the development process of this work.The application is divided into three main independent instances: gesture recognition, 3D trajectory detection and drone controller. First of all, the Section 4.1 to create a Deep Neural Network (DNN) model to recognize hand gestures, followed by Section 4.1.1 and 4.1.4 that describes the type of data used for this purpose and how they are generated. After, we will see about how orientation and camera-hand distance has been estimated in Section 4.2.1 and 4.2.2 to detect a 3D trajectory. Next, we focus on the drone controller in simulation and real. We conclude Section 4.4 with a detailed explanation of the entire pipeline.

4.1 Hand Gesture Recognition

MediaPipe has a python implementation for their Hand Keypoints Detector. It returns 2.5D coordinates of 21 hand landmarks (see Fig.4.1), consisting of x, y, and relative $depth$. For this project the $depth$ coordinate of each hand landmark have been deleted because we wanted to estimate them.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 4.1. Image from the open MediaPipe repository.

The coordinates of pixels in OpenCV follow a reference system (x, y) in which the origin is in top-left of each image frame, x is column-wise and y is row-wise on the computer screen. In order to normalize each point, the origin was converted from top-left to bottom-left. This gave us the classic cartesian coordinate system. After, for each landmark was computed the mean for x and y, converted in homogeneous coordinates and shifted them in order to match mean with origin. Note that shifting the data did not change how the data points are positioned relative to each other. Finally, each point has been scaled respect the max distance from mean point to all other hand points. This normalization permits to compare same gesture at different positions from the camera, if same orientation.

4.1.1 Data Acquisition and Description

Following the normalization of the data, it was thought of gestures that would allow firstly to acquire a 3D trajectory and secondly to interact directly with the drone.

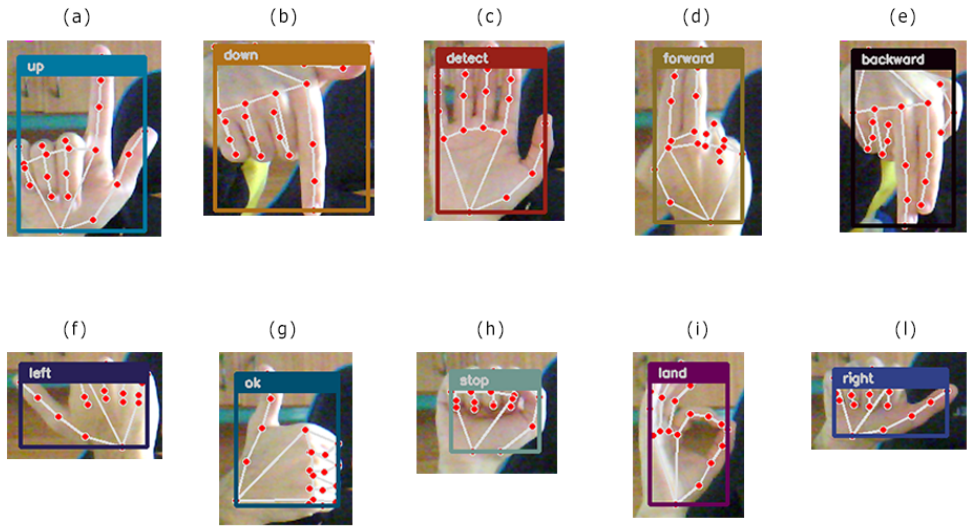


Figure 4.2. Full list of gestures that are available.

Below is the list of 10 gestures used in our project:

- `backward`: this command allows the drone to go backwards;
- `detect`: this command allows you to detect a three-dimensional (3D) trajectory;
- `down`: this command allows the drone to go down;
- `forward`: this command allows the drone to go forward;

- `land`: this command allows the drone to land;
- `left`: this command allows the drone to go left;
- `ok`: this command allows the drone to go close and execute a 3D trajectory;
- `right`: this command allows the drone to go right;
- `stop`: this command allows the drone to stop his movements;
- `up`: this command allows the drone to go up.

From the module that has been built is there the possibility to get data either from a webcam or from the drone camera. The dataset consists of 43 columns: 42 are the 21 $2D$ points' components, and the last is the target. 1000 examples were acquired, 100 images for each gesture. In total, the dataset is composed of 43000 elements. The python script to get data has also a restore in case of particular error (for example if images are acquired from the drone could be problems due to overheating or low battery). Given the structure of the generation script it is particularly easy to build a model with new gestures.

4.1.2 Pearson Correlation

Correlation is a measure of the linear relationship of 2 or more variables. Through correlation, we can predict one variable from the other. The logic behind using correlation for feature selection is that the good variables are highly correlated with the target. Furthermore, variables should be correlated with the target but should be uncorrelated among themselves.

If two variables are correlated, we can predict one from the other. Therefore, if two features are correlated, the model only really needs one of them, as the second one does not add additional information. We will use the Pearson Correlation here through the heatmap plot.

A heatmap is a two-dimensional graphical representation of data where the individual values that are contained in a matrix are represented as colors. The seaborn python package allows the creation of annotated heatmaps which can be tweaked using Matplotlib tools as per the creator's requirement.

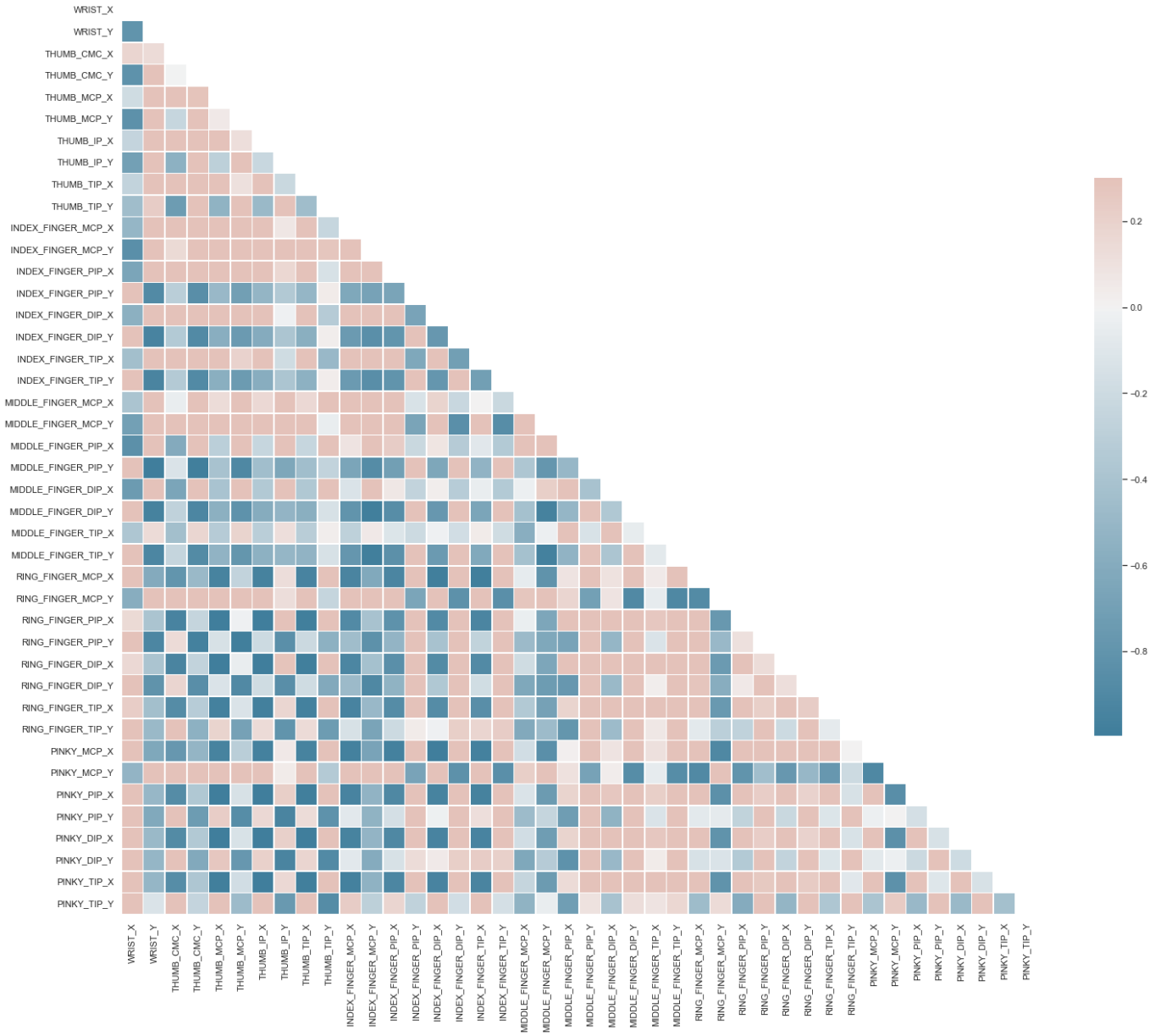


Figure 4.3. Heatmap is really useful to display a general view of numerical data, not to extract specific data point. It rapresents the correlation matrix and it is done using Pearson correlation.

Each square shows the correlation between the variables on each axis. Correlation ranges from -1 to $+1$. Values closer to zero implies weaker correlation (exact 0 implying no correlation). A value closer to 1 implies stronger positive correlation, that is as one increases so does the other and the closer to 1 the stronger this relationship is. A correlation closer to -1 is similar, but instead of both increasing, one variable will

decrease as the other increases. This is called negative correlation.
The diagonals are all 1 because those squares are correlating each variable to itself (so it's a perfect correlation). Larger the number associated (in absolute) the higher the correlation between the two variables. The plot is also symmetrical about the diagonal since the same two variables are being paired together in those squares. This is the reason why we we plot just the lower matrix.
Looking at the picture it is possible to see that there is a high correlation between variables. Therefore, we compared the correlation between features and remove one of two features that have an absolute correlation higher than 0.8

	WRIST_X	WRIST_Y	THUMB_CMC_X	INDEX_FINGER_MCP_Y	INDEX_FINGER_PIP_Y	MIDDLE_FINGER_MCP_X	MIDDLE_FINGER_PIP_X	MIDDLE_FINGER_DIP_X	RING_FINGER_DIP_Y
0	-0.259707	0.965687	0.207766	0.531605	-0.113953	-0.115013	0.085332	0.140984	-0.091693
1	-0.263627	0.953032	0.198660	0.547121	-0.129398	-0.083223	0.085907	0.153559	-0.095572
2	-0.321922	0.946766	0.099888	0.570557	-0.067859	-0.105317	0.134089	0.236692	-0.147661
3	-0.328812	0.895012	0.120297	0.513268	-0.093030	-0.059347	0.109070	0.187664	-0.048119
4	-0.328879	0.859546	0.095966	0.500062	-0.109972	-0.034755	0.139540	0.226688	-0.066398
...	...	...	...	...	...	...	...	...	...
994	-0.406536	-0.391501	-0.204471	-0.050516	0.265211	0.085998	0.035482	-0.153955	0.012629
995	-0.596513	-0.802604	-0.327698	-0.251534	0.205451	-0.220172	-0.032002	0.115846	0.380181
996	-0.402409	-0.629288	0.015719	-0.376210	0.228975	-0.039298	0.103746	0.070736	0.151951
997	-0.359158	-0.652018	0.101274	0.071518	0.466174	0.013573	0.024536	-0.019315	-0.191586
998	-0.154737	-0.723769	0.229609	0.254567	0.580679	-0.084856	-0.014974	-0.003328	-0.327776

Figure 4.4. Feature Selection: Feature Vs Feature.

Note: a better verion could be used, setting an absolute value, saying 0.5 as the threshold for selecting the variables. If we find that the predictor variables are correlated among themselves, we can drop the variable which has a lower correlation coefficient value with the target variable. High results are gained just finding that if the predictor variables are correlated among themselves we drop one of them. So we dropped all features except for 9 features: WRIST_X, WRIST_Y, THUMB_CMC_X, INDEX_FINGER_MCP_Y, INDEX_FINGER_PIP_Y, MIDDLE_FINGER_MCP_X, MIDDLE_FINGER_PIP_X, MIDDLE_FINGER_DIP_X, RING_FINGER_DIP_Y. These are the final features given by Pearson correlation.

4.1.3 PCA

PCA is a classical multivariate (unsupervised machine learning) non-parametric dimensionality reduction method that used to interpret the variation in high-dimensional interrelated dataset (dataset with a large number of variables). PCA reduces the high-dimensional interrelated data to low-dimension by linearly transforming the old variable into a new set of uncorrelated variables called principal component (Principal component (PC)) while retaining the most possible variation. The first component has the largest variance followed by the second component and so on. The first few components retain most of the variation, which is easy to visualize and summarise the feature of original high-dimensional datasets in low-dimensional space. PCA helps to assess which original samples are similar and different from each other.

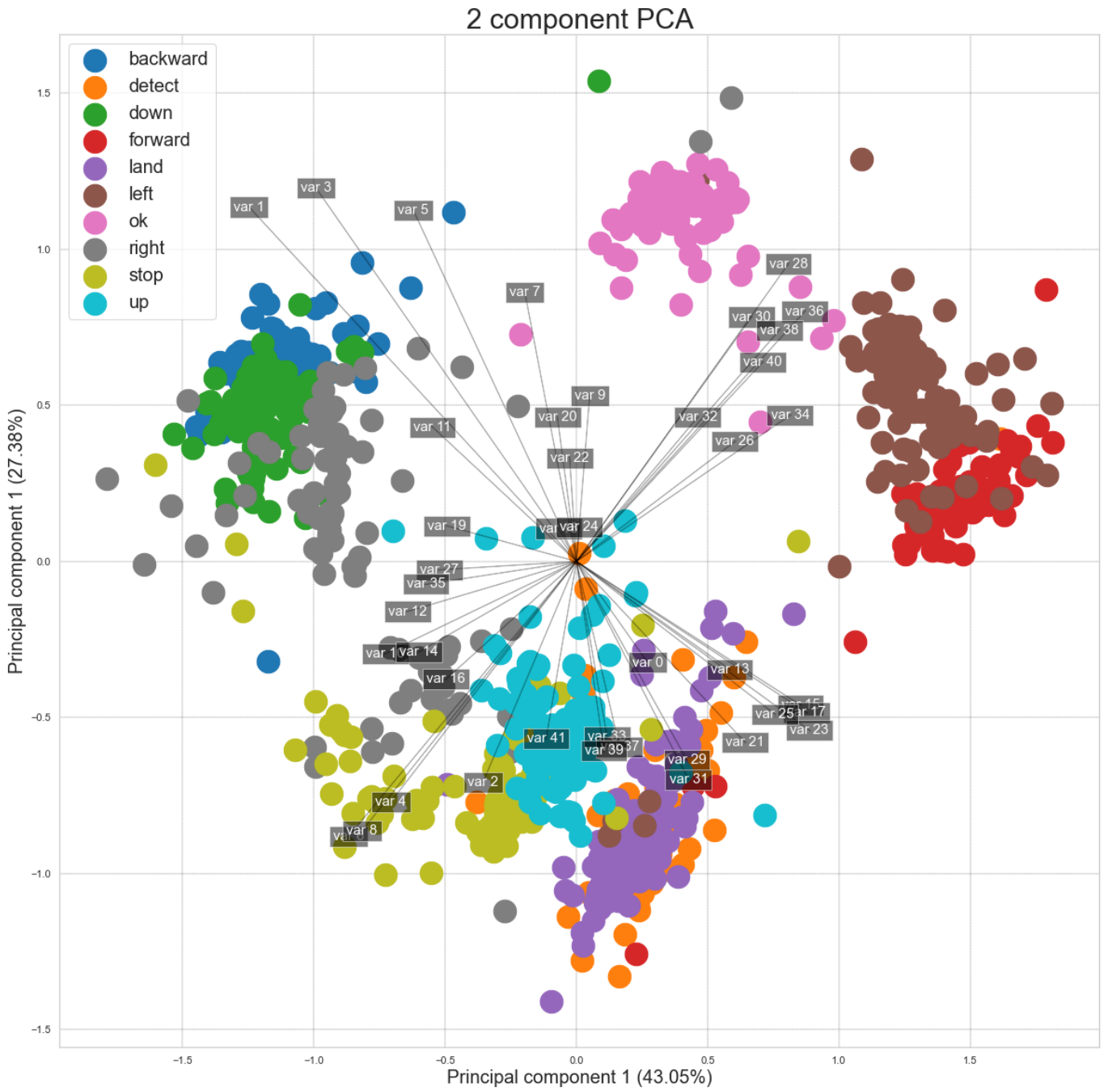


Figure 4.5. After dimensionality reduction, there usually is not a particular meaning assigned to each principal component. The new components are just the two main dimensions of variation.

As PCA is based on the correlation of the variables, it usually requires a large sample size for the reliable output. The sample size can be given as the absolute numbers or as subjects to variable ratios. The minimum absolute sample size of 100 or at least 10 or 5 times to the number of variables is recommended for PCA. On other hand, Comrey and Lee's (1992) have a provided sample size scale and suggested the sample size of 300 is good and over 1000 is excellent.

As the number of PCs is equal to the number of original variables, We should keep only the PCs which explain the most variance ($70-95\%$) to make the interpretation easier. More the PCs you include that explains most variation in the original data, better will be the PCA model. This is highly subjective and based on the user interpretation (Cangelosi et al., 2007).

For a lot of machine learning applications it helps to be able to visualize the dataset. Visualizing 2 or 3 dimensional data is not that challenging. However things are different when the dimension is above four. We can use PCA to reduce that n dimensional data into 2 or 3 dimensions so that we can plot and hopefully understand the data better. For this reason we can say that PCA can also be used for Data Visualization.

PCA is effected by scale so we need to scale the features in our data before applying PCA. In sklearn framework using StandardScaler helps standardize the dataset's features onto unit scale ($mean = 0$ and $variance = 1$) which is a requirement for the optimal performance of many machine learning algorithms. An alternative standardization is scaling features to lie between a given minimum and maximum value, often between zero and one, or so that the maximum absolute value of each feature is scaled to unit size. This can be achieved using MinMaxScaler or MaxAbsScaler, respectively. Problem about StandardScaler is when we have new data, and this cannot be scaled as the dataset used to build the ML model. We have choosen MinMaxScaler.

Biplots (see Fig. 4.5) are useful to visualize the relationships between variables and observations. The original data has n columns and we projects the original data which is m dimensional (where m is the number of rows, in other words, the sample number of the dataset) into 2 dimensions. From the biplot and loadings plot, we can see that a lot of variables are highly associated and forms cluster. If the variables are highly associated, the angle between the variable vectors should be as small as possible in the biplot. The length of PCs in biplot refers to the amount of variance contributed by the PCs. The longer the length of PC, the higher the variance contributed and well represented in space. The first two PCs contribute $70,43\%$ of the total variation in the dataset.

Scree plot (see Fig. 4.6) is another graphical technique useful in PCs retention. We should keep the PCs where there is a sharp change in the slope of the line connecting adjacent PCs. We took just only the 6 PC because they descrive around the 93% of the total variation in the dataset .

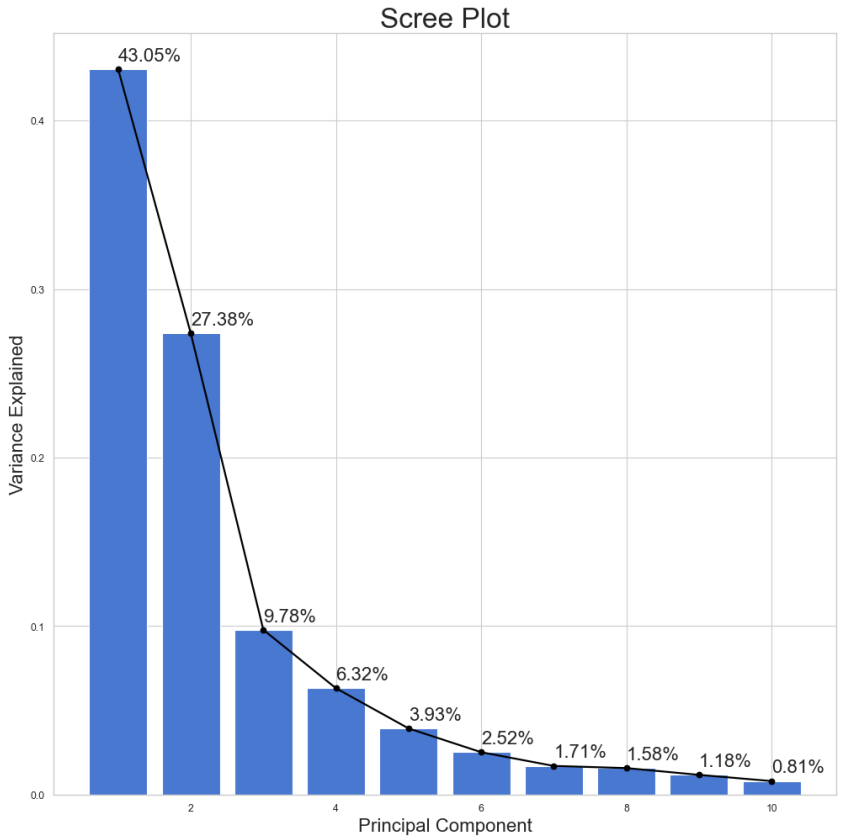


Figure 4.6. Scree plot.

4.1.4 Model

Now we are ready to choose a model. For classification tasks there are variety of different estimators/models that we can choose from. In the end, DNNClassifier from Tensorflow framework, with two hidden layers composed of 30 and 10 neurons each. Input layer is composed of 42 elements and the output layer of 10 classes (see Fig. 4.7). The model must choose among ten classes. The activation function is the Leaky ReLU and Adam the optimizer with learning rate 0.1, decay steps 10000 and decay rate 0.96. About steps for the training phase has been chosen 3000 as value. Loss is calculated by using softmax cross entropy.

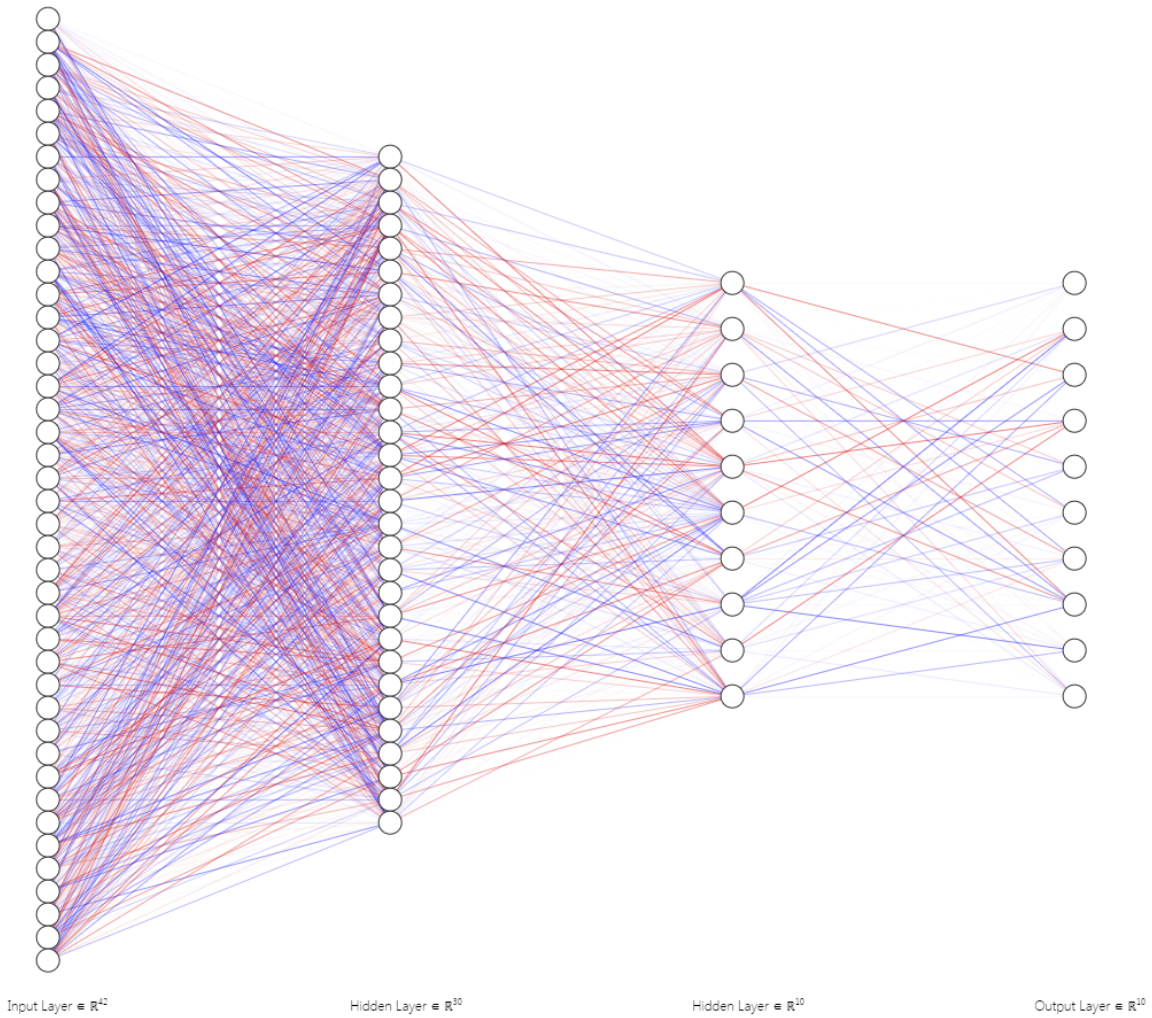


Figure 4.7. DNN image of the hand gesture recognition, generated by NNSVG[1]. This is the specific case when all the 42 variables were given as input.

Three models were genearated: one using the entire dataset, one using just only the selected features and finally one using PCA. Because of such a simple structure, high accuracy was reached for all of them. It is not required to retrain the model for each gesture in different illumination (or building a convolutional neural network), because MediaPipe takes over all the detection work.

4.2 3D trajectory detection

Trajectories will be detected using the detect gesture (see Fig. 4.2 (c)). The experiments about orientation estimation will be executed just only on that gesture.

4.2.1 Orientation estimation

In the following section, we will discuss the methods used to perform yaw, roll and pitch measurements starting from the pixels of the image captured with a camera. Note that yaw and pitch orientation estimates are made after the hand points are normalized, while roll is computed at the time of normalization, as described in. The normalization in this phase occurs by first of all making the translation so that the origin of the axes matches the midpoint of the hand, then all the points of the hand are scaled so that the maximum distance between them and the origin has a value of 1. Finally we rotate the hand w.r.t. the roll value computed during the normalization phase.

4.2.1.1 Turning and orientations

In order to find a value for the yaw we need to understand how to determine whether three points form a left-hand turn. This can be done by an orientation test, which is fundamental to many algorithms in computational geometry (citare cmsc754-spring2020-lects). Given an ordered triple of points $\langle p, q, r \rangle$ in the plane, we say that they have positive orientation if they define a counterclockwise oriented triangle (see Fig. 4.8 (a)), negative orientation if they define a clockwise oriented triangle (see Fig. 4.8 (b)), and zero orientation if they are collinear, which includes as well the case where two or more of the points are identical (see Fig. 4.8 (c)). It is important take care about the fact that the orientation depends on the order in which the points are given.

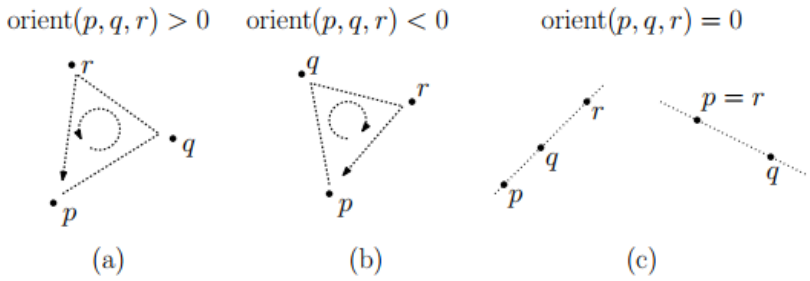


Figure 4.8. Orientations of the ordered triple (p, q, r).

Orientation is formally defined as the sign of the determinant of the points given in homogeneous coordinates, that is, by prepending a 1 to each coordinate. For example, in the plane, we define:
The sign of the orientation of an ordered triple is unchanged if the points are translated, rotated, or scaled (by a positive scale factor). In general, applying any affine transformation to the point alters the sign of the orientation according to the sign of the determinant of the matrix used in the transformation.

$$Orient(p,q,r) = det \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix} \tag{4.1}$$

Equation 4.1. Thus orientation generalizes the familiar 1-dimensional binary relations $<, =, >$.

If we consider points belonging to the same reference system and compute the orientation test then we can get information of how much the points are crushed together.

4.2.1.2 Roll estimation

To estimate the roll, we found the angle generated by the vertical axis passing over the wrist and the tip of the middle finger (see fig. 4.1).

4.2.1.3 Yaw estimation

First of all, to estimate the yaw, it is necessary to fix a hand. This project will be on the left-hand, so each comment will be in that direction. Despite this, the same can be said for the right hand. Then, we compute the orientation test (see Eq. 4.1) of three points $\langle p,q,r \rangle$. As already said, the order here is important to find the direction of orientation, therefore to know if the left-hand points left or right and with which intensity. We have seen empirically that if $roll < -5°$ or $roll > 5°$ it is better to consider $p =$ index_finger_mcp, $q =$ index_finger_pip and $r =$ index_finger_dip. Otherwise, if $-5° < roll < 5°$ we have noted that good results if $p =$ middle_finger_mcp, $q =$ middle_finger_pip and $r =$ middle_finger_dip. These hand points were chosen because they scale quite well with hand orientation.

The value of the orientation test is elevated to the second power, in order to be more stable around 0°. Then, it is divided by 1666, a value computed empirically to bring the results obtained from the orientation test to the range of $-90°$ to 90°. Since we used the quadratic form, this value will be very huge if greater than 90° or smaller than $-90°$, for this reason the part the exceeds has been reduced to be just the 10% of it.

4.2.1.4 Pitch estimation

To compute pitch estimation, we compute the middle point m between index_finger_mcp and index_finger_pip and then the difference between m and the thumb_tip, respect vertical axis (see fig. 4.1). Also here, as for yaw estimation, the value obtained is elevated to the second power, in order to be more stable around 0°. Then, we divided the result by 31 (empirically computed), to bring the results to the range of $-90°$ to 90°. Since we used the quadratic form, also here if the value is greater than 90° or smaller

than $-90°$ the results tends to be really big, therefore the part the exceeds has been reduced to be just the 10% of it.

4.2.2 Camera-hand distance estimation

In the initial phase of identifying a 3D trajectory, the palm of the hand is perpendicular to the range of action from the camera lens. At the beginning the hand is still and in the acquired image the same hand has a certain area h. If we consider the central point p of the hand as the average distance between all the reference points of the hand then this point p can be considered the origin of an orthogonal three-axis Cartesian reference system.
If we wanted to treat the hand as a material point, i.e. that does not change its orientation in space, then capturing all the midpoints of the hand and comparing the areas of the hand to estimate the depths over time t means obtaining those points that identify a trajectory 3d.
However, if we wanted to treat the hand as a body, and therefore also estimate its orientation in space, then things get complicated. In fact, although we would still be able to identify the two-dimensional displacement of the hand, in terms of height and width, as if we were treating the hand as a material point, instead we could no longer compare the areas of the hand because these can be altered due to the orientation of the hand, taken in a certain instant of time. This is why we have to estimate the orientation of the hand in space, because in this way we can project the points that describe the first hand detected, thus bringing us back to the previous case, comparing areas of the hand to estimate depth.

So after detecting the first gesture, the midpoint p was calculated and all points of the hand were translated in such a way that the midpoint p becomes the origin of the hand. Note that, unlike how it was done to normalize the data as an input to the neural network, here we do not scale the points w.r.t. the maximum distance between the midpoint and all other points of the hand. We do not do this because we do not want to lose the scaling information. Also all subsequent gestures (always referring to the "detect" gesture) with possible different orientation are captured and performed the same normalization process. At each frame we calculate the depth value by comparing the first hand acquired with the one in progress. Specifically, at the points that define the first hand acquired we implementd a matrix transformation: a 3D rotation along the 3 axes using the estimates of the yaw, roll and pitch calculated previously (see 4.2.1).
A rotation matrix rotates an object about one of the three coordinate axes or any arbitrary vector. The rotation matrix is more complex than the scaling and translation matrix since the whole $3x3$ upper-left matrix is needed to express complex rotations. It is common to specify arbitrary rotations with a sequence of simpler ones each along with one of the three cardinal axes. In each case, the rotation is through an angle, about the given axis. As in 4.2, 4.3 and 4.4, the three matrices R_X, R_Y and R_Z represent

transformations that rotate points through the angle θ in radians about the coordinate origin.

$$R_X(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & sin(\theta) & 0 \\ 0 & -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.2}$$

Equation 4.2. X-Rotation Matrix in homogeneous coordinates.

$$R_Y(\theta) = \begin{pmatrix} cos(\theta) & 0 & -sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ sin(\theta) & 0 & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.3}$$

Equation 4.3. Y-Rotation Matrix in homogeneous coordinates.

$$R_Z(\theta) = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.4}$$

Equation 4.4. Z-Rotation Matrix in homogeneous coordinates.

It must be further defined whether positive angles perform a clockwise (CW) or counterclockwise (CCW) rotation around an axis w.r.t. a specification of the orientation of the axis. Here, positive rotation angles cause a counterclockwise rotation about an axis as one looks inward from a point on the positive axis toward the origin. This is commonly the case for right-handed coordinate systems. Note that transformation matrices containing only rotations and translations are examples of rigid-body (solid-body) transformations, which do not alter the size or shape of an object. It is also possible to combine the three transformations to obtain a single matrix (see Eq. 4.5)

$$R(pitch, yaw, roll) = R_X(pitch) \cdot R_Y(yaw) \cdot R_Z(roll) \tag{4.5}$$

Equation 4.5. General rotation matrix, used to perform a rotation in Euclidean space.

Actually, instead of calculating the area of the hand, the mean of the sums of the distances between the hand midpoint p and all other reference hand landmark was calculated, to have a more robust metric. After having performed this last operation both on all the points of the current hand and on the first hand transformed with the general matrix, we finally made the difference between the first and the second value obtained previously. In this way we are able to obtain an estimate of the depth of the hand (which is performing the detecting gesture) even if it is oriented in space.

4.2.3 Univariate spline

Fitting data

4.3 Drone controller

4.3.1 Simulation

4.3.2 Real world

4.4 Pipeline

Chapter 5

Evaluation

Conclusions and Future works

Appendix A

List of Acronyms

3D three-dimensional

ANN Artificial Neural Network

CNN Convolutional Neural Network

DL Deep Learning

DNN Deep Neural Network

ML Machine Learning

MSE Mean Squared Error

NLP Natural Language Processing

NN Neural Network

PC Principal component

RNN Recurrent Neural Network

SSE Sum of Squared estimate of Errors

Bibliography

Leaky relu: improving traditional relu – machinecurve. https://www.machinecurve.com/index.php/2019/10/15/leaky-relu-improving-traditional-relu/. (Accessed on 01/11/2022).

Diego Andina, D. Pham, D. Andina, Antonio Vega-Corona, Juan Seijas, and J. Torres-Garcìa. *Neural Networks Historical Review*. 02 2007. ISBN 978-0-387-37450-5. doi: 10.1007/0-387-37452-3_2.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

W. W. Hager and H. Zhang. A survey of nonlinear conjugate gradient methods, 2006. Pacific journal of Optimization, vol. 2, no. 1, pp. 35–58.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. Citeseer.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.

Dewi Retno Sari Saputro and Purnami Widyaningsih. Limited memory broyden-fletcher-goldfarb-shanno (l-bfgs) method for the parameter estimation on geographically weighted ordinal logistic regression model (gwolr). In *AIP Conference Proceedings*, volume 1868, page 040009. AIP Publishing LLC, 2017.

Ryze Tech. Tello user manual v1.2, 2018. Available online: https://www.ryzerobotics.com/ (accessed on 21 January 2022).

Iliya Valchanov. Sum of squares total, sum of squares regression and sum of squares error, 2018. URL https://365datascience.com/tutorials/statistics-

tutorials/sum-squares/. [sum of squares total, denoted SST | Published on 5 Nov 2018].

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking, 2020.