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

Master's Thesis submitted to the
Faculty of Informatics of the *Università della Svizzera Italiana*
in partial fulfillment of the requirements for the degree of
Master of Science in Informatics

presented by
Umberto Cocca

under the supervision of
Dr. Alessandro Giusti
co-supervised by
Dr. Gianluigi Ciocca, Dr. Loris Roveda

February 2022

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

A tutti quelli che hanno fatto di me
quel che sono oggi e quel che sarò domani

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

Abstract

Robotic systems are increasingly being adopted for filming and photography purposes. In fact, by exploiting robots, it is possible to program complex motions, achieving high-quality videos and 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 shooting video content, even to those who are novices. To achieve this, after constructing a Deep Neural Network, which recognizes the gestures of a hand, and later creating a solid pipeline, which detect 3D trajectories obtained from 2D reference points, we captured 3D movements using a state-of-the-art hand tracking system. We estimated the orientation of the hand, and, with this information, we estimated depth as well. 3D trajectories were interpolated and noise-purified with 3 Ridge Regressions. Firstly, as a proof of concept, the captured trajectory was launched in a simulation on a drone, implemented in the ROS framework, and later in 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 has been an enriching experience. I was able to work on a great 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 to have worked on this thesis and I could not ask 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 Maria Grazia and my closest friends Andrea, Salvatore, Guido, Giada, Claudia, Silvia, Leyla and Ivo for providing me with unfailing support and continuous encouragement. This accomplishment would not have been possible without them. Thank you.

Contents

Figures

Tables

Equations

Listings

Introduction

Robots are increasingly taking over from what has traditional been the preserve of human creative people. Film creators are progressively turning to automated systems for recording moving images. Producing new, targeted, and engaging content can be a stressful, time-consuming endeavor. This is why more and more creators and advertisers are embracing tools that take the sting out of the process. Cinema robot not only decreases production time, but it allows teams to shoot a variety of angles that would be physically impossible or cost a fortune in labor.

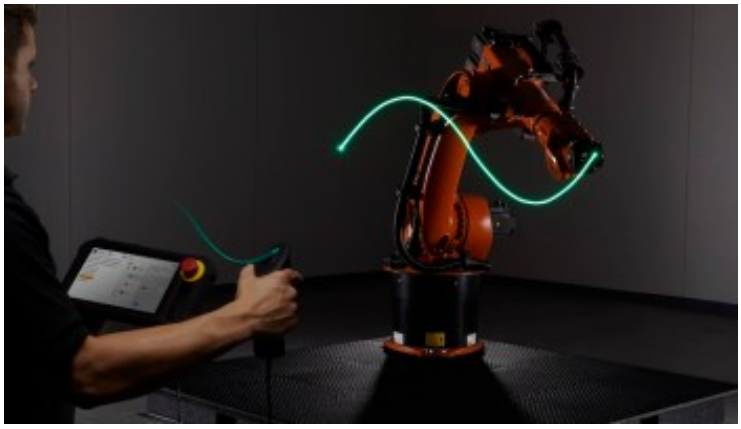


Figure 1. Guide a robot with the motion of the hand using a joystick.

Unfortunately, it is known that robot programming may be a very laborious task. They have to synchronize robot motion with objects in the frame. There are also other requirements that need to be taken into account, such as camera settings (focal length, aperture time and desired depth of field). The credit for trying to speed up the process of robot programming goes to different companies. For example, KUKA, a partner of andyRobot, has developed a plug-in for industry leading animation software, Autodesk Maya, that allows KUKA robots to be programmed by anyone who knows how to animate inside Maya [**?**]: a robot program can be created by simply dragging the three-dimensional (3D) model of the robot through space in the virtual world and setting keyframes. Furthermore, SISU Cinema Robotics can be named: the company has

developed a way to guide the robot with the motion of the hand using a joystick [**?**] (see Fig. 1).

Figure 2. Drones in filmmaking.

Drone filming and aerial photography (see Fig. 2) present a wide range of benefits and opportunities to businesses and brands looking to capture stunning aerial imagery and to produce incredible new perspectives [**?**]. There can be quite a steep initial learning curve when it comes to getting to grips with drone technology. Furthermore, to capture high-quality content, it is important to use equipment that is both easy to operate and correctly set up as it is also important to understand how to fly smoothly and consistently capture professional and smooth looking aerial footage. In fact, not every position of the drone are visually interesting angles, as sometimes it is about understanding composition and depth that makes all of the difference. For the success of the video shooting, it is extremely crucial to look for a specific angle that seems unique, cinematic, and suggestive that will hold the attention of the audience. A lot of drones have additional features that help pilots capture great looking footage, such as centre points that help frame shots. Additionally, filters can be added to certain drone cameras, allowing them to control the amount of light entering the lens without compensating with shutter speed.

Our goal is being able to control a drone using just the motion of a hand for shooting, without the use of a joystick. The system can also be used to program a trajectory easily. This can be very interesting because it is an intuitive system: basically, no experience is required in piloting. In addition, the one who shoots could potentially also be the actor. In spite of that, ease of use does not imply that the system does not require attention: it is quite simple to get caught up in the task at hand, and completely lose track of

the flight time. Some drones, such as DJI, will begin automatically landing when the battery begins to get low, however there are still a lot of other drones that do not have this intelligence and will literally crash out of the sky when it runs out of battery. Most crashes are actually caused by battery voltage drop [**?**]. Not moderating the flight time and not keeping a close eye on the battery levels, can lead to disastrous accidents [**?**].

In order to achieve this goal, research has been carried out in the literature to see if a hand tracking system already exists. There is some solution such as [**?**] where, with the extensive experiments with a prototype GPU-based implementation of the proposed method, demonstrates that accurate and robust 3D tracking of hand articulations can be achieved in near real-time (15Hz), using a Kinect sensor. A more recent work [**?**] marking another important Virtual Reality (VR) input milestone, in the evolution of VR input, is the announcement of hand tracking on Oculus Quest enabling natural interaction in using hands on an all-in-one standalone device. All without the need for a controller, external sensors, gloves, or a PC to power it. And finally, one of the latest research of Google Group research [**?**]: a solution that does not require any additional hardware and performs in real-time on mobile devices (the one actually used in this project and widely spoken in the literature chapter 1). Thanks to this system it was possible to build a Deep Neural Network (DNN) to recognize specific hand gestures. The orientation of a specific gesture (called "detect gesture") was also estimated in order to obtain also information on depth. Through hand gesture recognition and orientation estimation, a pipeline to detect a 3D trajectory was defined. However, this trajectory $G()$ was disturbed by estimation errors. Hence, two different data fitting algorithms were applied on the trajectory $G()$: first 3 splines and then 3 ridge regression, in both cases for each component X,Y and Z of $G()$. This phase allowed to go from trajectory $G()$ to $G_{smooth}()$.

As a Proof of Concept (POC), the captured trajectory was launched in simulation on a drone, implemented in the Robot Operating System (ROS) framework. After that, the DJI Ryze Tello drone (see Sec. 3.1) was used to test the application in real life. DJITelloPy is the DJI Tello drone python interface used in the project to make the application communicating with the drone. The trajectory veracity was visible to the naked eye. The correctness of the fixed dimension of the trajectory was extremely important for the project's purpose since, if the range of action in which the drone operates is uncontrolled, then it could get dangerous for those around it.

The project's objective was reached with success and the entire pipeline for the acquisition of the trajectory can be exploited in Augmented Reality (AR) and VR scenarios as it would allow people to interact with virtual objects and/or perform actions.

Chapter 1

Literature Review

While being natural to people, robust real-time hand perception is a decidedly challenging computer vision task, as hands often occlude themselves or each other and lack high contrast patterns. MediaPipe [**?**] offers cross-platform and customized ML solutions for live and streaming media, and among these there is one that deals with the task of finger/hand tracking. Since this solution was used at the basis of the built system, the main elements that characterize their classifier will be described. In this chapter, the section is divided into two parts: in the first one the architecture of the system will be described, while the second will focus on the results obtained.

Mediapipe proposes an efficient hand tracking pipeline that can track multiple hands, and a hand pose estimation model that is capable of predicting 2.5D hand landmarks with only Red Green Blue (RGB) image input [**?**]. The solution does not require any additional hardware and performs in real-time on mobile devices.

The above mentioned hand tracking solution utilizes a ML pipeline consisting of a palm detector and a hand landmark model. The first one operates on a full input image to locate palms via an oriented hand bounding box. The second one, operates on the cropped hand bounding box provided by the palm detector returning high-fidelity 2.5D landmarks. Providing the accurately cropped palm image to the hand landmark model drastically reduces the need for data augmentation (e.g. rotations, translation and scale); furthermore, it allows the network to dedicate most of its capacity towards landmark localization accuracy. In a real-time tracking scenario, the bounding box has been derived from the landmark prediction of the previous frame as input for the current frame, thus avoiding applying the detector on every frame. Instead, the detector is only applied on the first frame, or when the hand prediction indicates that the hand is lost.

To detect initial hand locations, a single shot detector model optimized for mobile real-

time application has been deployed. This model has to work across a variety of hand sizes with a large scale span and be able to detect occluded and self-occluded hands. First, instead of a hand detector a palm detector has been trained, because estimating bounding boxes of rigid objects like palms and fists is significantly simpler than detecting hands with articulated fingers. Second, an encoder-decoder feature extractor has been used for a larger scene-context awareness even for small objects. Lastly, the focal loss has been minimized: the focal loss is an extension of the cross-entropy loss function that would down-weight easy examples and focus training on hard negative, used mainly for dense object detection.

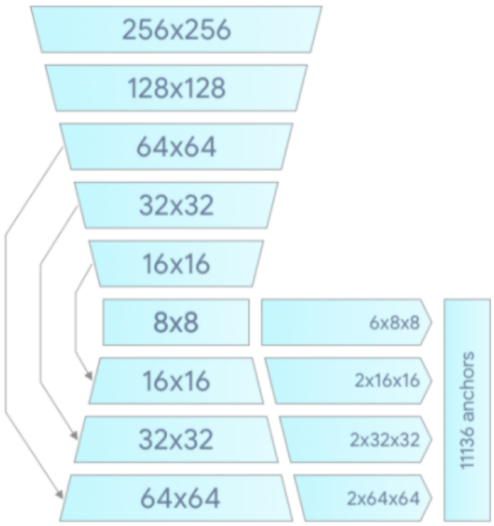


Figure 1.1. Palm detector model architecture.

After running palm detection over the whole image, the subsequent hand landmark model performs precise landmark localization of 21 2.5D coordinates inside the detected hand regions via regression. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions.

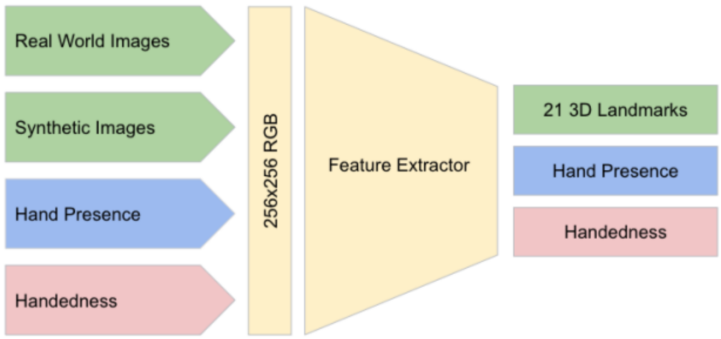


Figure 1.2. Architecture of hand landmark model.

The model has three outputs:

- 21 hand landmarks consisting of x, y, and relative depth (note that depth was not used for this project). The 2D coordinates are learned from both real-world images as well as synthetic datasets;
- A hand flag indicating the probability of hand presence in the input image;
- A binary classification of handedness, e.g. left or right hand.

For the hand landmark model, the experiments showed that the combination of real-world and synthetic datasets provided the best results. Only real-world images have been evaluated. The target was to achieve real-time performance on mobile devices, therefore the experiment was done with different model sizes; it has been found that the "Full" model (see Table 1.3) provided a good trade-off between quality and speed. Increasing model capacity further introduced only minor improvements in quality but decreased significantly in speed.

Model	Params (M)	MSE	Time(ms) Pixel 3	Time(ms) Samsung S20	Time(ms) iPhone11
Light	1	11.83	6.6	5.6	1.1
Full	1.98	10.05	16.1	11.1	5.3
Heavy	4.02	9.817	36.9	25.8	7.5

Figure 1.3. Hand landmark model performance characteristics.

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 [**?**]. 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, the aim is to model the predictive distribution $p(t|x)$ because this expresses the 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:

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

Equation 2.1. Linear combination of the input variables.

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 possible extend the class of models by considering linear combinations of fixed nonlinear functions of the input variables:

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

Equation 2.2. Linear combinations of fixed nonlinear functions of the input variables.

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 2.3. Linear combinations of fixed nonlinear basis functions. By using non linear basis functions, the function $y(x,w)$ can 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 possible choices for the basis functions as:

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

Equation 2.4. Gaussian basis function. 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. The identity basis functions in which the vector $\phi(x) = x$ can be simply used.

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 [**?**]), so that the following is minimised:

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

Equation 2.5. 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.

The curve fitting problem can be solved 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 $y(x, w^*)$.

The formula (2.5) can be written in matrix notation as:

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

Equation 2.6. 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 2.7. Design matrix. The elements are given by $\phi_{nj} = \phi_j(x_n)$.

Therefore, the optimization problem has to be solved 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 2.8. 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 2.9. Optimization problem for ridge regression. The goal is to find the weight w that minimize the cost function $E(w)$.

The gradient is computed and solving for w the result obtained is:

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

Equation 2.10. They are known as the normal equations for the least squares problem.

This is the real minimum because if the second derivative $\nabla^2 E(w) = \phi^T \phi$ is taken, this is a symmetric matrix, so it's also positive definitive matrix, which means that this objective function that is being minimized is convex. So if a stationary point is found, such that derivative is zero, a global minimum is also found. Furthermore, to find a solution the matrix $\phi^T \phi$ needs to be inverted, so some condition that assures this is invertible is needed, and this is the case when the columns of the matrix are linearly independent.

Once the solution w^* is found and a new data point that has never been seen during training is received, the new target t^* is predicted as:

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

Equation 2.11. The goal is to find the weight w that minimize the cost function $E(w)$.

2.2 The Problem of Overfitting

If an expensive set of features is used (for example a ten polynomial grade x^{10}), then the model interpolates very close the training data, because there is a model with 10 parameters where the data points can be perfectly represented. The risk is that the model overfit data points (see Fig. 2.1 (center)).

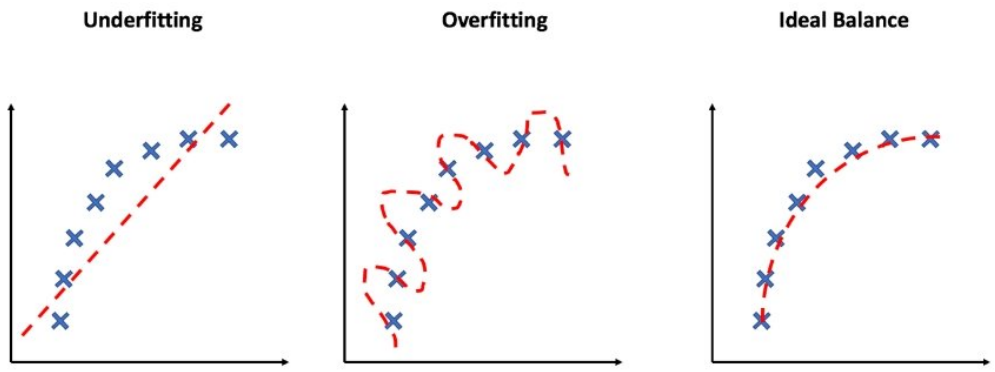


Figure 2.1. Image by "Applied Supervised Learning with R" book [a]. It is possible to see high bias resulting in an oversimplified model (that is underfitting); high variance resulting in over-complicated models (that is overfitting); and lastly, striking the right balance between bias and variance.

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

On the other hand, if the 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)). A trade-off between fits data and being able to generalize (see Fig. 2.1 (center)) is desired.

2.2.1 Regularization

Some features with high values parameters have been penalized, if the model is over-fitting, it is very likely that its parameters will have a big magnitude. This means that also the features that supply to these parameters will be higher and very low and this can cause a lot of problems.

In order to control over-fitting the idea of adding a regularization term to an error function is introduced, so that the total minimization error function takes the form of:

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

Equation 2.12. Total Minimization Error Function.

Where λ is the regularization coefficient and it is the trade-off between how well fit training set is and how to establish the parameters w with low values, therefore having simple hypotheses avoiding over-fitting. $E_D(w)$ is the error based on dataset, while $E_W(w)$ 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 2.13. 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 coefficients of multiple-regression models in scenarios where independent variables are highly correlated. The total error function becomes:

$$E(w) = \frac{1}{2} \sum_{n=1}^{N} [t_n - w^T \phi(x_n)]^2 + \frac{\lambda}{2} w^T w. \tag{2.14}$$

Equation 2.14. Loss Function for Ridge Regression. 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, the result obtained is:

$$w^* = (\lambda I + \phi^T \phi)^{-1} \phi^T t. \tag{2.15}$$

Equation 2.15. Ridge regression solution. This is an extension of the least-squares solution 2.10.

From this side it denotes a better version than before because is also possible 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.

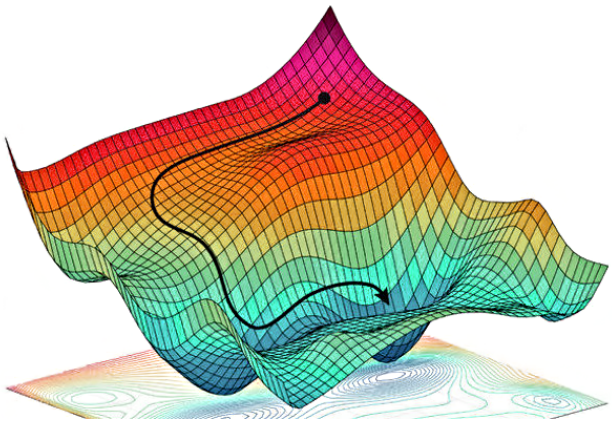


Figure 2.2. Gradient descent is based on the observation that if the multi-variable function $E(w)$ is defined and differentiable in a neighborhood of a point w_0 , then w_0 decreases fastest if one goes from w_0 in the direction of the negative gradient of $E(w)$ at w_0, $-\nabla E(w_0)$.

Let's now use Mean Squared Error (MSE), instead of using (2.5) due to the fact that it is possible 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. In fact, 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.

$$E_D(w) = \frac{1}{2N} \sum_{n=1}^{N} [y(x_n, w) - t_n]^2. \tag{2.16}$$

Equation 2.16. Mean Squared Error. $1/2$ is added, as in SSE, so the derivative doesn't need a constant out front. The problem is not an issue, 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), it could have been written:

$$E_W(w) = \frac{\sum_{m=1}^{M} w_m^2}{2}. \tag{2.17}$$

Equation 2.17. 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 the 0th weight is considered.

Combining together following (2.12)

$$E(w) = \frac{1}{2N} \left\{ \sum_{n=1}^{N} [y(x_n, w) - t_n]^2 + \lambda \sum_{m=1}^{M} w_m^2 \right\}. \tag{2.18}$$

Equation 2.18. Loss Function for Ridge Regression alternative form.

The problem is set as follows:
- Let $E(w)$
- Find $\arg\min_w E(w)$)

Keep changing w to reduce $E(w)$ until a minimum is hopefully reached:

$$
\begin{aligned}
&Repeat = \{ \\
&\quad w_0 := w_0 - \alpha \frac{1}{N} \sum_{n=1}^{N} [y(x_n, w) - t_n]^2 \, x_0; \\
&\quad w_j := w_j - \alpha \frac{1}{N} \sum_{n=1}^{N} [y(x_n, w) - t_n]^2 \, x_j + \frac{\lambda}{M} w_j. \\
&\} \qquad (2.19)
\end{aligned}
$$

Equation 2.19. Gradient Descent for Ridge Regression.

where in compact form is:

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

Equation 2.20. Gradient Descent for Ridge Regression in compact form.

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 [**?**].

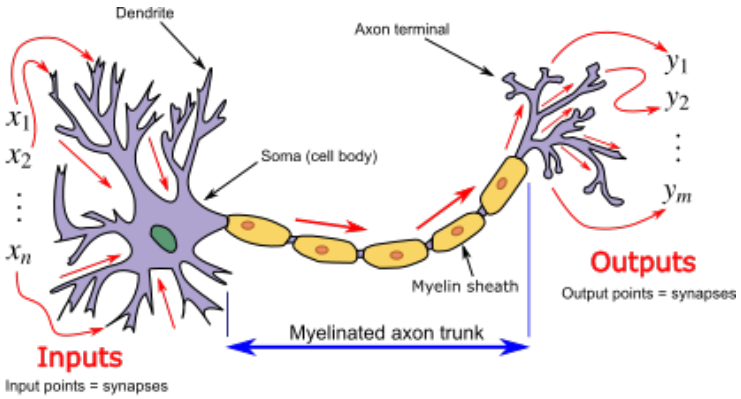


Figure 2.3. Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals. The signal represents a short electrical pulse called 'spike'.

Neurons have cell 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, 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) it is possible to see a NN, seen as mathematical model. It is just a group of these different neurons strung together. Trying to underline an analogy with the biological systems: circles identify the cell body where they are fed 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.

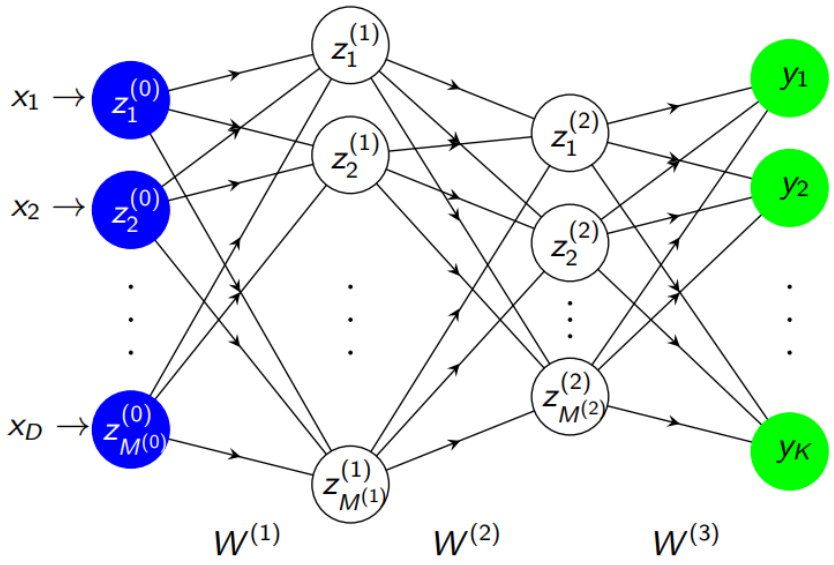


Figure 2.4. The image graphically shows a NN with three layers (or two hidden layers). The input layer has $M^{(0)}$ neurons and the output layer has K outputs. Neurons are organized in layers to allow parallel computation to avoid cyclic dependencies. The process of computing NN outputs from inputs 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 0-th 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 perceptron-like computation:

$$u_m^{(1)} = (w_m^{(1)})^T x + b_m^{(1)}; \qquad z_m^{(1)} = f(u_m^{(1)}); \qquad m = 1, ..., M^{(1)}. \tag{2.21}$$

Equation 2.21. Neurons functions of the first layer.

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

$$z^{(l-1)} = (z_m^{(l-1)})_{m=1,...,M^{(l-1)}}. \tag{2.22}$$

Equation 2.22. Chain of neurons functions.

as input of:

$$u_m^{(l)} = (w_m^{(l)})^T z^{(l-1)} + b_m^{(l)}; \qquad z_m^{(l)} = f(u_m^{(l)}). \tag{2.23}$$

Equation 2.23. Neurons functions of a specific layer l.

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

$$z^{(0)} = x. \tag{2.24}$$

Equation 2.24. Input layer as zeroth layer.

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

$$z_m^{(l)} = f((w_m^{(l)})^T z^{(l-1)} + b_m^{(l)}). \tag{2.25}$$

Equation 2.25. Forward propagation.

which can be written as a matrix multiplication:

$$z^{(l)} = f(W^{(l)} z^{(l-1)} + b^{(l)}). \tag{2.26}$$

Equation 2.26. Forward propagation as a matrix multiplication. Function that identifies 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 function and needs to be differentiable, so that gradient descent training is applicable. For the hidden layers of the network, 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, nonlinear steps are needed.
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)

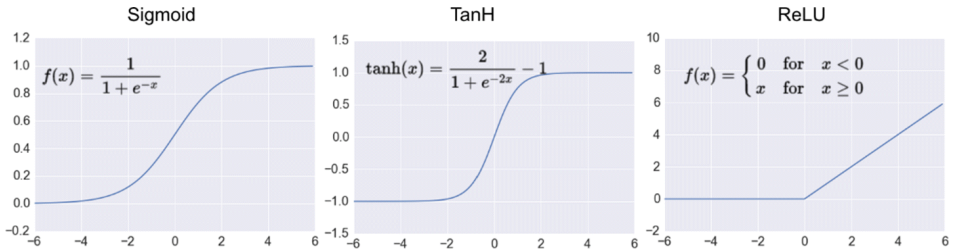


Figure 2.5. Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals. The signal represents a short electrical pulse called 'spike'.

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" [**?**], in particular on the difficulty of training Recurrent Neural Network (RNN) [**?**], while the second is to accelerate the convergence speed. More sophisticated activation function as the "leaky ReLU" try to solve the dying ReLU problem [**?**]. In contrast to ReLU, in which the negative part is totally dropped, leaky ReLU assigns a non-zero slope to it. Leaky ReLU and its variants are consistently better than ReLU in Convolutional Neural Network (CNN) [**?**].

Leaky Rectified linear activation is introduced in acoustic model [**?**]. Mathematically, it is defined as follows:

$$f(x) = \begin{cases} \frac{x}{a} & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}. \tag{2.27}$$

Equation 2.27. Function that identifies 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.

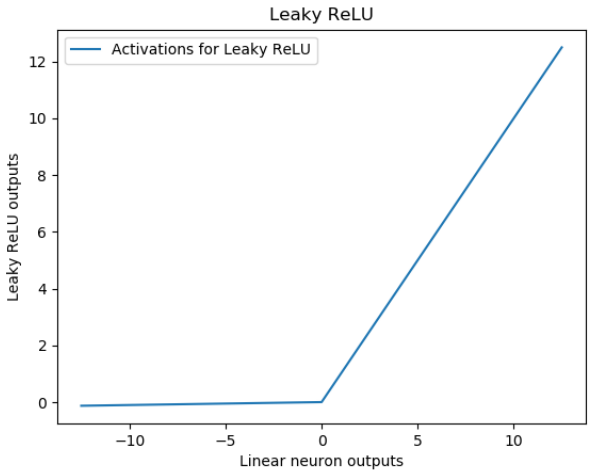


Figure 2.6. Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals. The signal represents a short electrical pulse called "spike".

2.3.2 Neural Network Setup for Classification

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

$$c^* = \arg\max_k y_k. \tag{2.28}$$

Equation 2.28. Sample of a class k.

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

$$y = (y_k)_{k=1,\ldots,K}; \qquad y_k \geq 0; \quad \sum_k y_k = 1. \tag{2.29}$$

Equation 2.29. Probability distribution over classes.

The advantage is that a differentiable measure of the quality of the output on theoretical grounds is derivable, using probability theory. In order to make the network output a probability distribution, exponentials are taken and normalized. This is the softmax nonlinearity:

$$S(y) = \left(\frac{e^{y_1}}{\sum_k e^{y_k}}, ..., \frac{e^{y_K}}{\sum_k e^{y_k}} \right). \tag{2.30}$$

Equation 2.30. 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 a NN with softmax output is assumed, can be computed 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:

$$t = (0, ..., 0, 1, 0, ..., 0). \tag{2.31}$$

Equation 2.31. One-hot style target.

This is treated 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:

$$E_{CE} = -\sum_k (t_k \log y_k). \tag{2.32}$$

Equation 2.32. Cross-entropy loss.

The intuition about the cross-entropy corresponds to the number of additional bits needed to encode the correct output, given that the (possibly wrong) prediction of the network is accessible. 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:

$$E_{CE+SM} = -\sum_k (t_k \log S_k(y)). \tag{2.33}$$

Equation 2.33. Softmax cross entropy. To train the network with Backpropagation, the calculation of the derivative of the loss is needed. 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 Backpropagation

The term Backpropagation strictly refers only to the algorithm for computing the gradient, not how the gradient is used. First the mathematical aspect will be presented in 2.3.3.1 and then the algorithm 2.3.3.2.

2.3.3.1 Training

To train a NN is possible to use the Gradient Descent. This requires to compute the gradient of the NN error w.r.t. each weight. In order to perform this computation in an efficient way it is important to derive the Backpropagation algorithm. Since the Eq. 2.34 has this form then we have chain rule 2.35, assuming that the non-linearity f is computed independently for each neuron, that is always true except for the softmax nonlinearity.

$$z_n^{(l)} = f(u_n^{(l)}) = f\left(\sum_m w_{mn}^{(l)} z_m^{(l-1)} + b_n^{(l)}\right). \tag{2.34}$$

$$\begin{aligned}
\frac{\partial z_n^{(l)}}{\partial w_{mn}^{(l)}} &= f'\left(u_n^{(l)}\right) z_m^{(l-1)}; \\
\frac{\partial z_n^{(l)}}{\partial b_n^{(l)}} &= f'\left(u_n^{(l)}\right); \\
\frac{\partial z_n^{(l)}}{\partial w_m^{(l-1)}} &= f'\left(u_n^{(l)}\right) w_{mn}^{(l)}.
\end{aligned} \tag{2.35}$$

Equation 2.34. Chain Rule.

for any $l = 1, ..., L$. Now, assuming that for a given sample x the error $E(y) = E(z^{(L)})$, then, for the last layer, gradients are immediately calculable as in Eq 2.36.

$$\begin{aligned}\frac{\partial E}{\partial w_{mn}^{(L)}} &= \frac{\partial E}{\partial z_n^{(L)}} \frac{\partial z_n^{(L)}}{\partial w_{mn}^{(L)}} = \frac{\partial E}{\partial z_n^{(L)}} f'\left(u_n^{(l)}\right) z_m^{(l-1)};\\ \frac{\partial E}{\partial b_n^{(L)}} &= \frac{\partial E}{\partial z_n^{(L)}} \frac{\partial z_n^{(L)}}{\partial b_n^{(L)}} = \frac{\partial E}{\partial z_n^{(L)}} f'\left(u_n^{(l)}\right).\end{aligned} \tag{2.36}$$

Equation 2.35. Gradients for the last layer.

This calculation is more complicated for the lower layers since all paths leading to a certain weight must be considered. In Eq. 2.37 the general case.

$$\begin{aligned}\frac{\partial E}{\partial z^{(l)}} &= \begin{pmatrix} \frac{\partial E}{\partial z_1^{(l)}} & \cdots & \frac{\partial E}{\partial z_{M^{(l)}}^{(l)}} \end{pmatrix} \in \mathbb{R}^{1xM^{(l)}};\\ \frac{\partial z^{(l)}}{\partial z^{(l-1)}} &= \begin{pmatrix} \frac{\partial z_1^{(l)}}{\partial z_1^{(l-1)}} & \cdots & \frac{\partial z_1^{(l)}}{\partial z_{M^{(l-1)}}^{(l-1)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_{M^{(l)}}^{(l)}}{\partial z_1^{(l-1)}} & \cdots & \frac{\partial z_{M^{(l)}}^{(l)}}{\partial z_{M^{(l-1)}}^{(l-1)}} \end{pmatrix} \in \mathbb{R}^{M^{(l)}xM^{(l-1)}};\\ \frac{\partial z^{(l)}}{\partial w_{ij}^{(l)}} &= \begin{pmatrix} \frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}} \\ \vdots \\ \frac{\partial z_{M^{(l)}}^{(l)}}{\partial w_{ij}^{(l)}} \end{pmatrix} \in \mathbb{R}^{M^{(l)}x1}.\end{aligned} \tag{2.37}$$

Equation 2.36. Backpropagation Training.

Furthermore, it is possible to decompose $\frac{\partial z^{(l+1)}}{\partial z^{(l)}}$ into the gradient of the nonlinearity and the network part as in Eq 2.42.

$$\frac{\partial z^{(l+1)}}{\partial z^{(l)}} = \frac{\partial z^{(l+1)}}{\partial u^{(l+1)}} \frac{\partial u^{(l+1)}}{\partial z^{(l)}}. \tag{2.38}$$

We define $F^{(l+1)} := \frac{\partial z^{(l+1)}}{\partial u^{(l+1)}}$ and $W^{(l+1)} := \frac{\partial u^{(l+1)}}{\partial z^{(l)}}$. Then, by chain rule:

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \frac{\partial E}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial z^{(L-1)}} \cdots \frac{\partial z^{(l+1)}}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial w_{ij}^{(l)}}. \tag{2.39}$$

The bias update is similar. Let $\delta^{(l)}$ be the gradient of the loss w.r.t. the activation of the l-th layer:

$$\delta^{(l)} = \frac{\partial E}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial z^{(L-1)}} \cdots \frac{\partial z^{(l+1)}}{\partial z^{(l)}} \in \mathbb{R}^{1xM^{(l)}}. \tag{2.40}$$

and note that:

$$\delta^{(l)} = \delta^{(l+1)} \frac{\partial z^{(l+1)}}{\partial z^{(l)}} = \delta^{(l+1)} F^{(l+1)}; W^{(l+1)} \qquad \delta^{(L)} = \frac{\partial E}{\partial z^{(L)}}. \tag{2.41}$$

Finally, combining prior results:

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \delta^{(l)} \frac{\partial z^{(l)}}{\partial w_{ij}^{(l)}} = \delta^{(l)} F^{(l)} z_i^{(l-1)}. \tag{2.42}$$

2.3.3.2 Gradient Descent by Backpropagation

Follows the complete algorithm to perform a gradient step in NN training, using the Backpropagation algorithm to compute the gradients:

1. Perform the forward pass, save intermediate results
2. For l = L,...,1,
3. Compute $\delta^{(l)}$ from $\delta^{(l+1)}$ (except for $l = L$, the recursion start)
4. Compute and save the weight gradients for layer l
5. Update all weights simultaneously: $w = w - \eta \nabla w$

where η is the learning rate, and ∇w collects the gradients. In the forward case we compute neurons activation from layer 1 to layer L; instead, in the backward case we compute errors from layer L to layer 1.

2.3.4 Optimisation Algorithms

Gradient descent can be adapted to minimize the loss function of a predictive model on a training dataset, such as a classification or regression model. This adaptation is called stochastic gradient descent, it represents an extension of the gradient descent optimization algorithm for minimizing a loss function of a predictive model on a training dataset **?**. The algorithm is referred to as "stochastic" because the gradients of the target function w.r.t. the input variables are probabilistic approximation. A challenge when using stochastic gradient descent to train a neural network is how to calculate the gradient for nodes in hidden layers in the network. This requires a specific technique from calculus called the chain rule and an efficient algorithm that implements it that can be used to calculate gradients for any parameter in the network. This algorithm is called Backpropagation 2.3.3. Hence, to sum up: Gradient Descent is an optimization method and Backpropagation is used to compute gradients which are required for gradient descent.

The choice of optimization algorithms strongly influences the effectiveness of the learning process, as they update and calculate the appropriate and optimal values of that model. Specifically, if the gradient descent is considered, which is 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 [**?**], BFGS, the improved version to decrease memory usage L-BFGS [**?**], etc... the main advantages are that there is 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 [**?**]. 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).

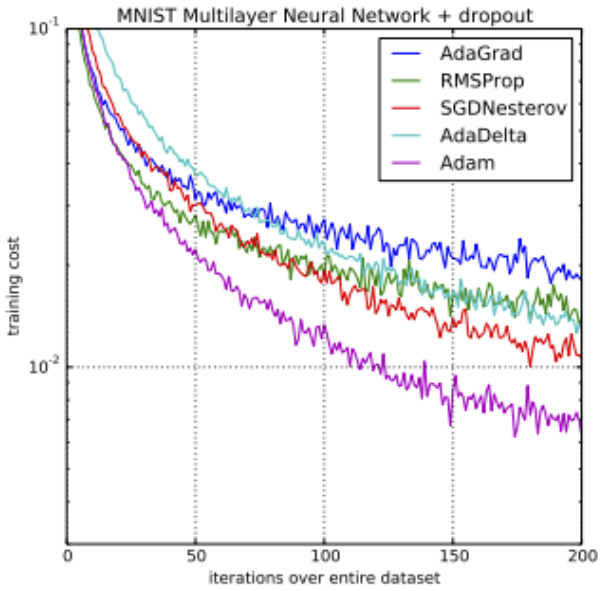


Figure 2.7. Comparison of Adam to other optimization algorithms training a Multilayer Perceptron[a].

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

To be more specific, Adam combines the advantages of two other extensions of stochastic gradient descent:

- Adaptive Gradient Algorithm (AdaGrad), maintains a per-parameter learning rate that improves performance on problems with sparse gradients e.g. NLP and computer vision problems;
- Root Mean Square Propagation (RMSProp), maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight. Therefore, the algorithm works pretty well on online and non-stationary problems e.g. noisy.

Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance). In the Fig. 2.7 the performance of Adam in action.

Chapter 3

Tools

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

As a general overview, in the image 3.1 are the programs and frameworks used to realize this project. The tools are grouped according to their characteristics. For example, on the left we find git, visual studio etc... for code creation and management. At the center tools for viewing graphics and image processing such as Matplotlib and Photoshop. On the right the processing of three-dimensional models and the execution of Visual Effects as Blender and After Effects. In other the area of machine learning through Tensorflow and Scikit-learn.

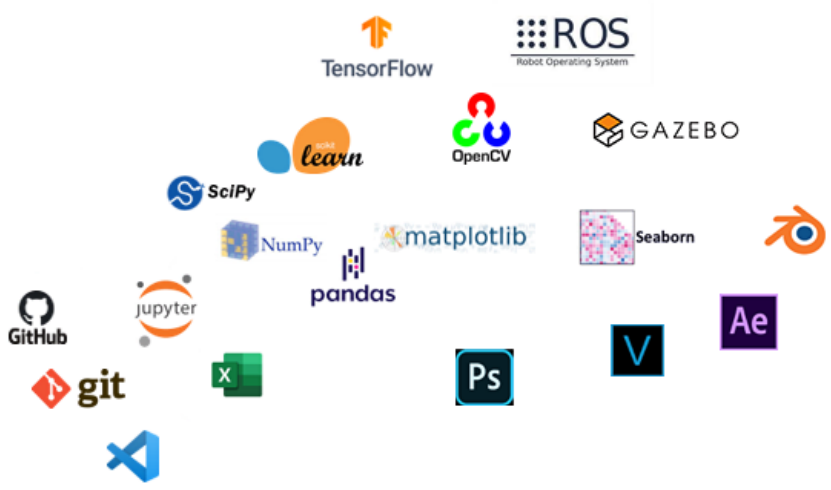


Figure 3.1. The image delineates all the main tools used to achieve the goal of the project.

3.1 DJI Ryze 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 it is suitable for flying indoors. Tello captures $5MP$ photos and streams 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 is $100m$ [**?**].

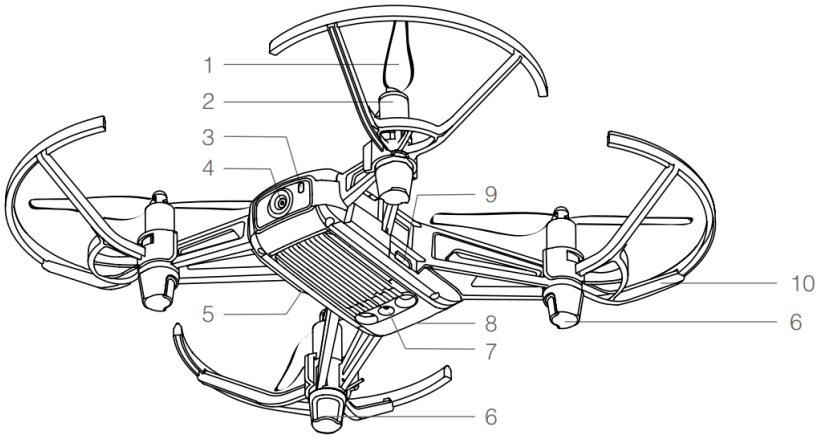


Figure 3.2. 1.Propellers; 2.Motors; 3.Aircraft Status Indicator; 4.Camera; 5.Power Button; 6.Antennas; 7.Vision Positioning System; 8.Flight Battery; 9.Micro USB Port; 10.Propeller Guards.

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 can 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 Command Types and Results

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 that return "ok" if the command was successful, "error" or an informational result code if the command 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.	6*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 Simulation Tools

The tools used for the simulation are ROS 3.2.1 the meta-operating system, virtualization to use Ubuntu 3.2.2 and Gazebo to operate with the model of the Tello in 3D 3.2.3.

3.2.1 Robot Operating System

ROS is an open-source, meta-operating system for robot. It provides the services from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. The ROS runtime "graph" is a peer-to-peer network of processes that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including

synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server.

The primary goal of ROS is to support code reuse in robotics research and development. ROS is a distributed node framework that allows executables to be individually designed and freely coupled to runtime. These processes can be grouped into Packages and Stacks, which can be easily shared and distributed. ROS also supports a federated system of code Repositories that enable collaboration to be distributed as well. ROS currently only runs on Unix-based platforms. Software for ROS is primarily tested on Ubuntu and Mac OS X systems, though the ROS community has been contributing support for Fedora, Gentoo, Arch Linux and other Linux platforms. A port to Microsoft Windows for ROS is possible, it has not yet been fully explored.

3.2.2 Virtualization

In this regard, we worked on a virtual machine to simulate the Ubuntu system, installed on windows 7. This environment, called a "virtual machine", is created by the virtualization software by intercepting access to certain hardware components and certain features. The physical computer is then usually called the host, while the virtual machine is often called a guest. Most of the guest code runs unmodified, directly on the host computer, and the guest operating system thinks like it is running on a real machine. VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. Not only is VirtualBox an extremely feature rich, high performance product for enterprise customers, it is also the only professional solution that is freely available as Open Source Software.

3.2.3 Gazebo

While ROS serves as the interface for the robot, Gazebo is a 3D simulator, that offers the ability to accurately and efficiently simulate robots in complex indoor and outdoor environments. Gazebo can use multiple high-performance physics engines, such as Open Dynamics Engine (ODE), Bullet, etc (the default is ODE). It provides realistic rendering of environments achieving high-quality lighting, shadows and textures. It can model sensors that "see" the simulated environment, such as laser range finders, cameras (including wide-angle), Kinect style sensors, etc. Gazebo is an independent project like any other project used by ROS **?**. For this project ROS Noetic with a 11.x version of Gazebo was used. Thanks to Gazebo it was possible to launch the 3D trajectory acquired by hand through the webcam on a simulated drone.

3.3 Frameworks

The main frameworks used in this project are listed below.

DJITelloPy DJI Tello drone python interface uses the official Tello SDK and Tello EDU SDK. This library has an implementation of all Tello commands, easily retrieves a video stream, receives and parses 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].

[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`

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].

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

[8] http://seaborn.pydata.org/
[9] https://scikit-learn.org

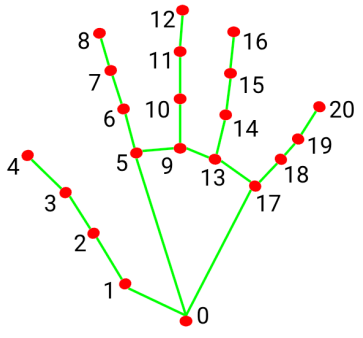
Chapter 4

Methods

This chapter illustrates the method used to approach the development process of this work. The application is divided into three main independent instances: hand 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 specific hand gestures, followed by Section 4.1.1 and 4.1.4 which describes the type of data used for this purpose and how they are generated. After that, it will be shown how orientation and depth were estimated in Section 4.2.1 and 4.2.2 to detect a 3D trajectory. Next, there will be a focus on the drone controller in simulation and real. Section 4.4 is concluded with a detailed explanation of the entire pipeline during 3D trajectory capturing.

4.1 Hand Gesture Recognition

MediaPipe has a python implementation for its hand key-points 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 coordinated depth of each reference point of the hand was eliminated to be estimated.



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 the top-left of each RGB 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 resulted in the classic Cartesian coordinate system. After that, the mean for x and y for each landmark was computed, converted in homogeneous coordinates and shifted them to match the mean with the origin. Note that shifting the data did not change how the data points were positioned w.r.t. each other. Finally, each point was scaled w.r.t. maximum 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

After the data normalization, gestures that would allow to acquire a $3D$ trajectory and to that would interact directly with the drone were designed.

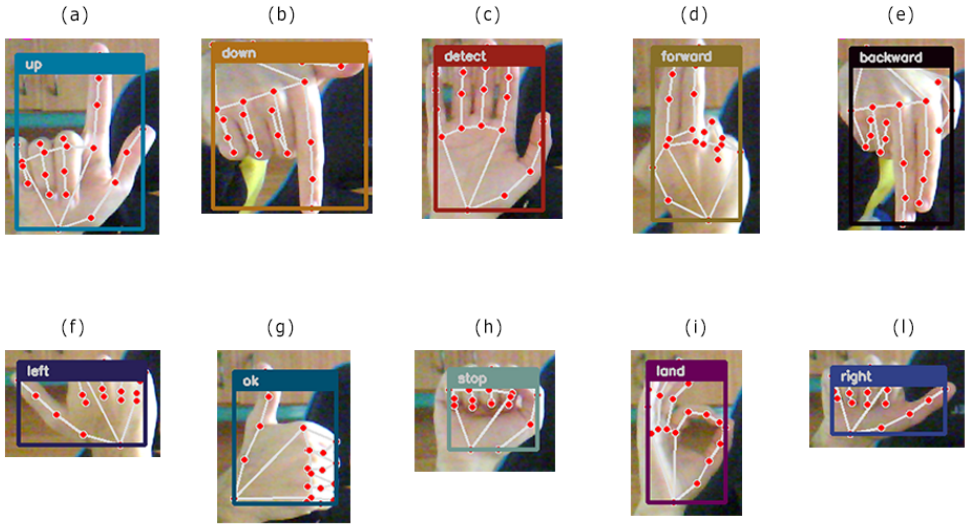


Figure 4.2. Full list of gestures that are available.

Below, there is the list of 10 gestures used in this project:

- `backward`: this command allows the drone to go backwards;
- `detect`: this command allows the user to detect a 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 conclude the 3D trajectory capturing and execute it;
- `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 there is the chance to get data either from a webcam or from the drone camera. The dataset consists of 43 columns: 42 are the 21 two-dimensional (2D) points' components, and the last one 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 errors (e.g. if images are acquired from the drone, there could be problems due to overheating or low battery). Given the structure of the generation script, it was particularly easy to build a model with new gestures.

4.1.2 Pearson Correlation

Correlation is a measure of the linear relationship of two 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, one is predictable 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. The Pearson Correlation through the heatmap plot has been used here.

A heatmap is a 2D 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.

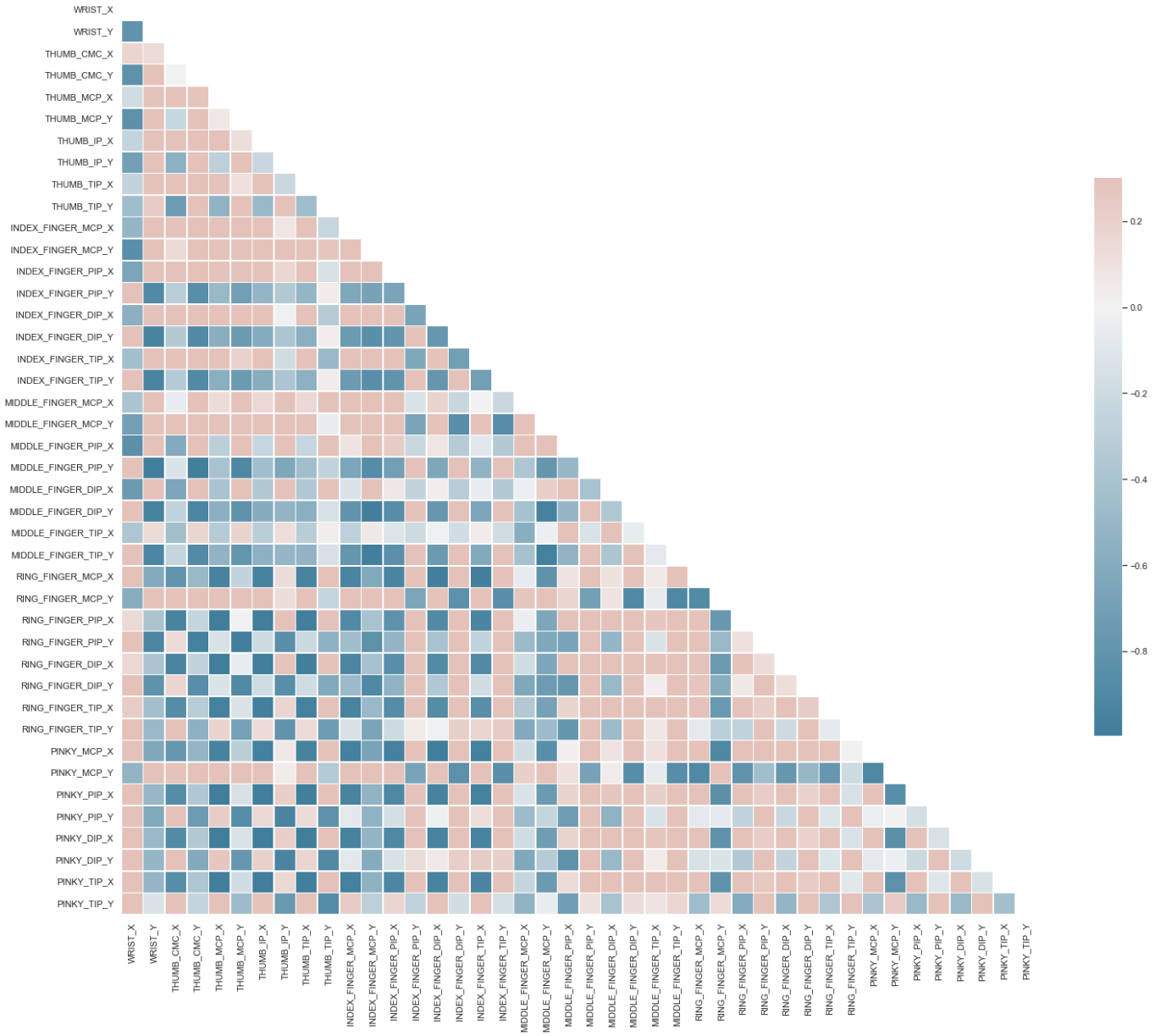


Figure 4.3. Heatmap is really useful to display a general view of numerical data, not to extract specific data point. It represents the correlation matrix and it is done using Pearson correlation.

Each square shows the correlation between the variables on each axis. Correlation ranges from -1 to $+1$. Values closer to zero imply 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). The larger the number associated (in absolute) the higher the correlation between the two variables. The plot is also symmetrical to the diagonal since the same two variables are being paired together in those squares. This is the reason why just the lower matrix has been plotted.

Looking at the picture, it is possible to see that there is a high correlation between variables. Therefore, the correlation between features has been compared and one of two features that have an absolute correlation higher than 0.8.

After plotting the correlation matrix, an high level of correlation between each specified couple of variables has been found. Therefore, for those couples whose correlation where higher than 80% one variable has been removed, since if two features are correlated, the model only needs one as the second does not bring additional information.

	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 version could be used, setting an absolute value, saying 0.5 as the threshold for selecting the variables. If the predictor variables are correlated among themselves, the variable which has a lower correlation coefficient value with the target variable can be dropped. High results are gained just finding that if the predictor variables are correlated among themselves then one of them should be dropped. After this, features were selected, dropping columns that were highly correlated going from 42 to 9 variables. The final features given by Pearson correlation are:

- `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`.

4.1.3 Principal Component Analysis

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.

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, only the PCs should be kept, which explain the most variance ($70-95\%$) to make the interpretation easier. The more PCs that explains most variation in the original data are included, the better the PCA model will be. This is highly subjective and based on the user interpretation [**?**].

For a lot of machine learning applications, it helps to be able to visualize the dataset. Visualizing two or three dimensional data is not that challenging. However, things are different when the dimension is above four.PCA can be used to reduce that n dimensional data into two or three dimensions so that data can be plotted and understood. For this reason it is possible to say that PCA can also be used for Data Visualization.

PCA is effected by scale so scaling the features is needed in the data before applying PCA. In Sklearn Framework using StandardScaler() function helps to standardize the dataset's features onto unit scale ($mean = 0$ and $variance = 1$), which is a requirement

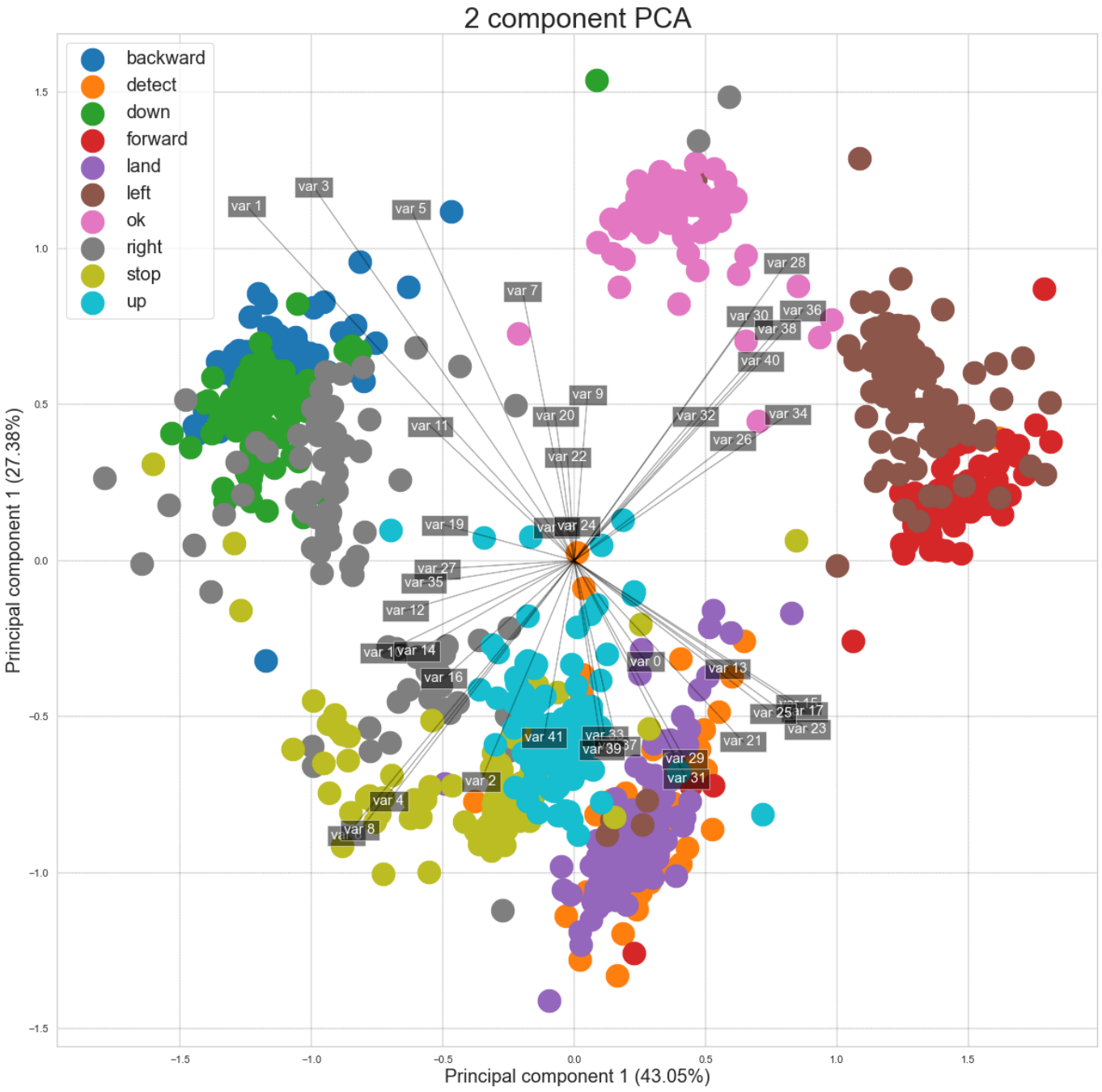


Figure 4.5. After dimensionality reduction, there usually is not a particular meaning assigned to each principal component. The new components are just the two main dimensions of variation.

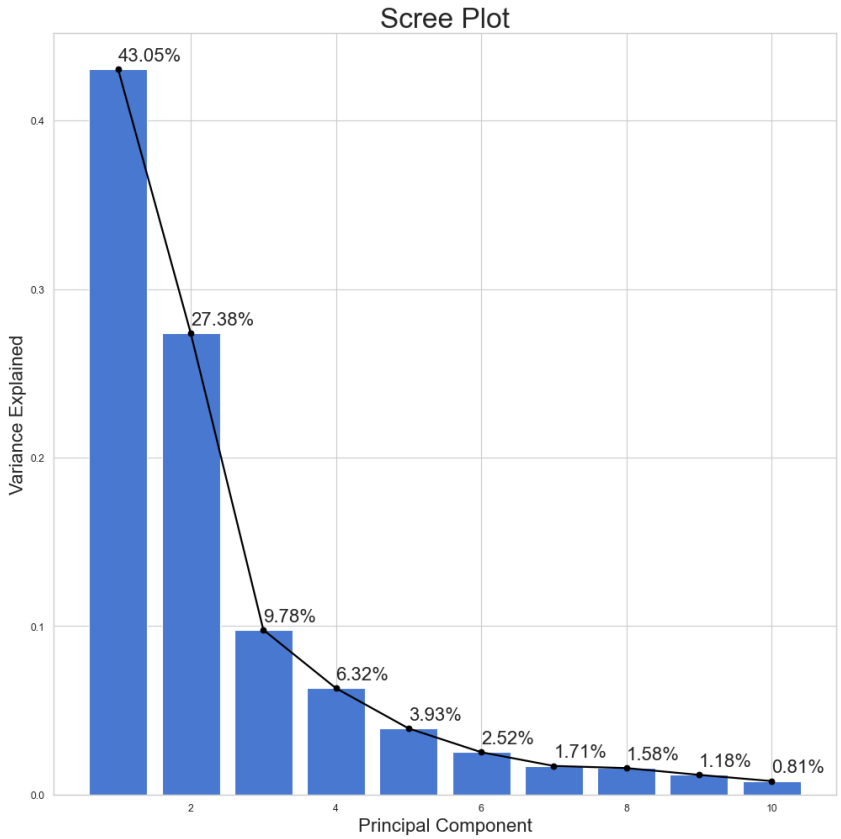


Figure 4.6. Scree plot.

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 function, respectively. A Problem about StandardScaler is when there is new data, and this cannot be scaled as the dataset used to build the ML model.MinMaxScaler function has been chosen.

Biplots (see Fig. 4.5) are useful to visualize the relationships between variables and observations. The original data has n columns and the original data which is m dimensional into 2D is projected (where m is the number of rows, in other words, the sample number of the dataset). From the biplot and loadings plot, it is possible to 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 repre-

sented in space. The first two PCs contribute about 70, 43% of the total variation in the dataset.

Scree plot (see Fig. 4.6) is another graphical technique useful in PCs retention. The PCs where there is a sharp change in the slope of the line connecting adjacent PCs shuold be kept. Only the 6 PC have been taken because they describe around the 93% of the total variation in the dataset.

4.1.4 Model

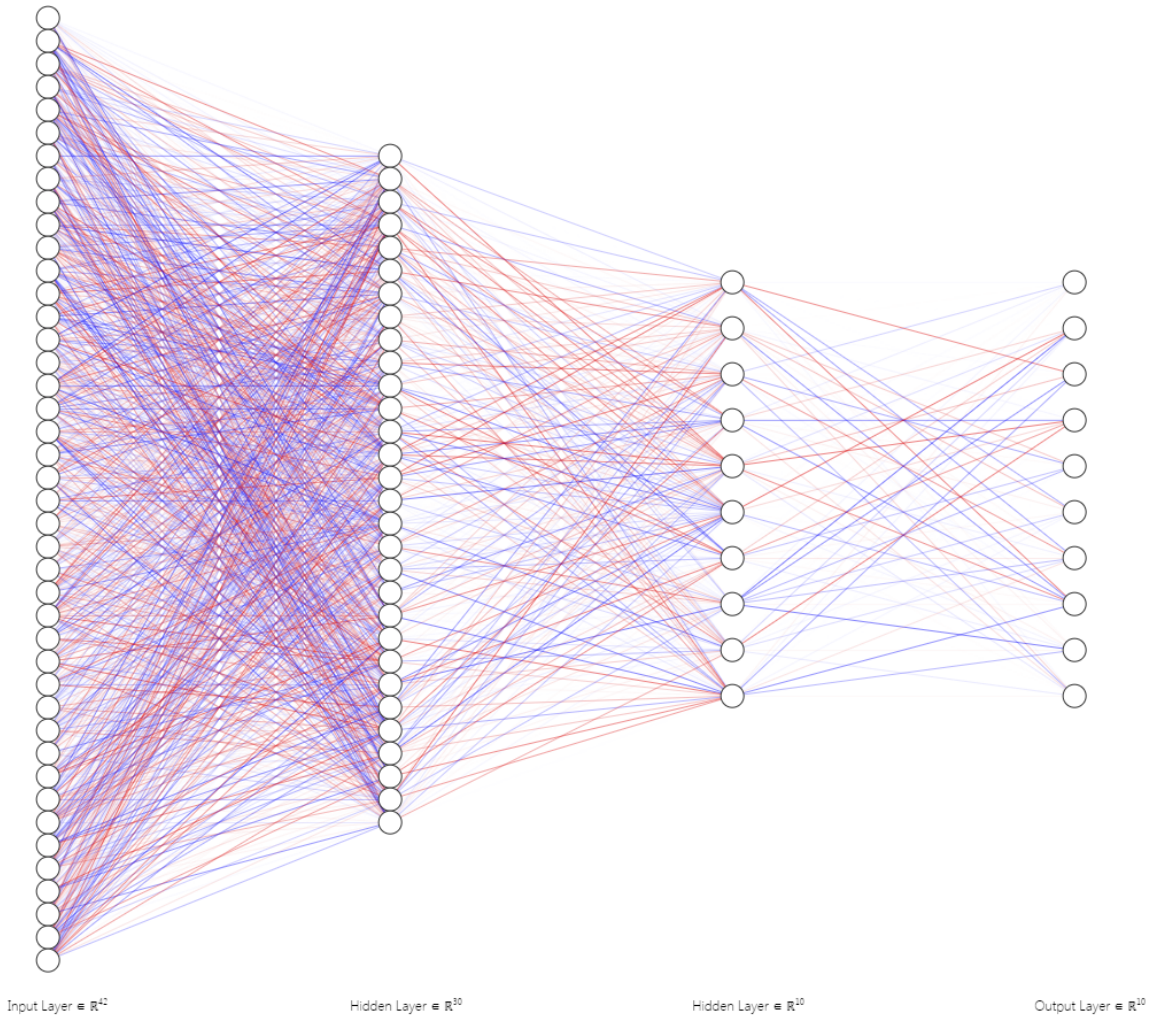


Figure 4.7. DNN image of the hand gesture recognition, generated by NNSVG[1]. This is the specific case when all the 42 variables were given as input.

For classification tasks there are a variety of different estimators/models from which it is possible to choose. In the end, DNNClassifier from Tensorflow Framework was used, with two hidden layers composed of 30 and 10 neurons each. Input layer was composed of 42 elements and the output layer of 10 classes (see Fig. 4.7). The model must choose among ten classes; hence, the gestures introduced in 4.1.1. The activation function used was the Leaky ReLU. Then, we proceeded to use Adam as optimizer with parameters learning rate 0.1, decay steps 10000 and decay rate 0.96. Regarding "steps", for the training phase was chosen 3000 as value. Loss is calculated by using softmax cross-entropy.

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

4.2 3D Trajectory Detection

Trajectories have been identified using the "detect" gesture (see Fig. 4.2 (c)). The experiments about orientation estimate will be executed only on that gesture.

4.2.1 Orientation Estimation

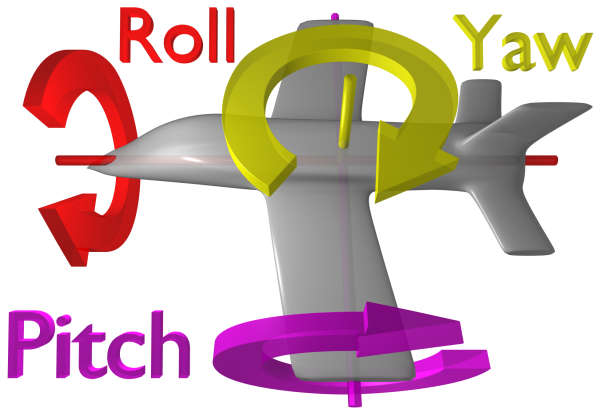


Figure 4.8. Aircraft principal axes.

In the following section, the methods used to perform yaw, roll and pitch measurements starting from the pixels of the image captured with a camera will be discussed. Note that yaw and pitch orientation estimates are made after the hand points are normalized,

while the roll is computed at the time of normalization. Normalization at this stage occurs first by making the translation so that the origin of the axes corresponds to the middle point 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, the hand is rotated w.r.t. the roll, the value computed during the normalization phase.

4.2.1.1 Turning and Orientations

In order to find a value for the yaw, it is needed 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 [**?**]. Given an ordered triple of points $\langle p,q,r\rangle$ in the plane, they have a positive orientation if they define a counterclockwise oriented triangle (see Fig. 4.9 (a)), negative orientation if they define a clockwise oriented triangle (see Fig. 4.9 (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. 4.9 (c)). It is important to take care about the fact that the orientation depends on the order in which the points are given.

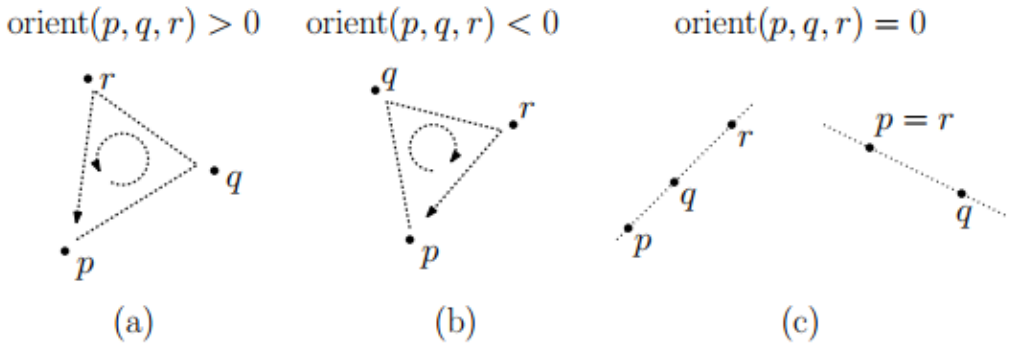


Figure 4.9. Orientations of the ordered triple (p, q, r).

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 see Eq. 4.1.

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

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

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.

If points belonging to the same reference system are considered and the orientation test is computed then it is possible to get information on how much the points are crushed together.

4.2.1.2 Roll Estimation

To estimate the roll, the angle generated by the intersection of the vertical axis passing from the "wrist", and the axis passing through the "wrist" and the “middle fingertip” was found (see fig. 4.1).

4.2.1.3 Yaw Estimation

To estimate the yaw, the orientation test (see Eq. 4.1) of three points $\langle p, q, r \rangle$ was computed. The order is important to find the the clockwise or counterclockwise orientation, hence, to know whether the left hand points left or right and with what intensity. Empirically, if $roll < -5$ or $roll > +5$ it was better to consider $p =$ "index finger mcp", $q =$ "index finger pip" and $r =$ "index finger dip". Otherwise, if $-5 < roll < +5$ results were good 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 was raised to the second power, in order to be more stable around 0 (keeping the sign of orientation). Then, it was scaled the definition range in a valid range by associating -90 to a, and +90 to b, where a and b are values returned from the previous step and the hand takes a position corresponding to $yaw = -90$ and $yaw = +90$. Since the quadratic form has been used the value will grow very quickly if greater than +90 or less than -90, for this reason, the part that exceeds has been reduced to 10% of it.

4.2.1.4 Pitch Estimation

To estimate the pitch, the middle point m between "index finger mcp" and "index finger pip" and then the difference between m and the "thumb tip" w.r.t. vertical axis was computed (see fig. 4.1). Also, in here, as for the yaw estimation, the value obtained is raised to the second power, to be more stable around 0 (keeping the sign of orientation). Then, it was scaled the definition range in a valid range by associating -90 to a, and +90 to b, where a and b are values returned from the previous step and the hand

takes a position corresponding to $pitch = -90$ and $pitch = +90$. Since the quadratic form has been used the value will grow very quickly if greater than +90 or less than -90, for this reason, the part that exceeds has been reduced to 10% of it.

4.2.2 Depth 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 the central point p of the hand is considered as the average distance among all the reference points of the hand, then p can be considered the origin of an orthogonal three-axis Cartesian reference system.

If the hand was treated 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 (see Fig. 4.10).

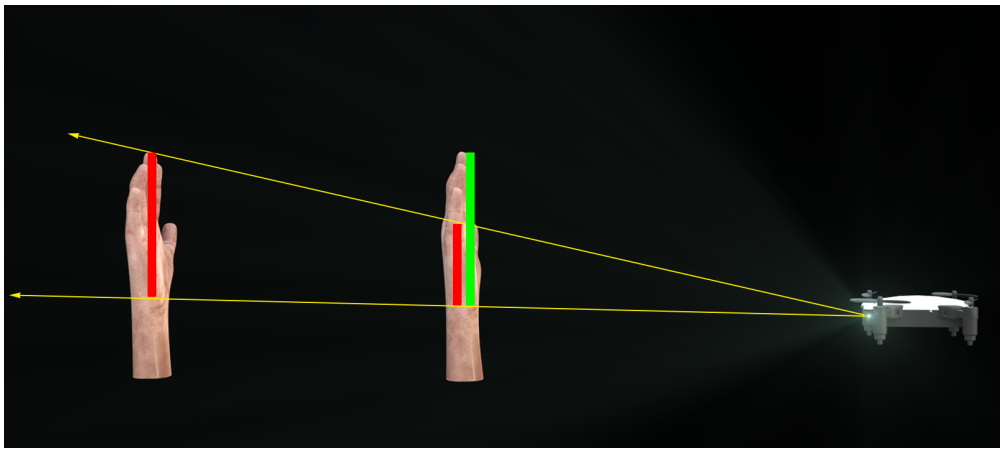


Figure 4.10. Depth estimate - Same orientation. On the left is the first h_1 hand captured at time t_1, in the middle is the h_2 hand captured at time t_2. So what happened was that the hand moved closer to the camera. We note that the projection of the 2D points of h_1 in h_2 have a distance (red segment) less than that of the points in h_2 (green segment). Image rendered with Blender.

However, if the hand was treated as a body, and therefore also estimate its orientation in space, then things get complicated. In fact, although identifying 2D hand displacement, in terms of height and width, would be possible as if the hand was being treated as a material point, instead the areas of the hand could not be compared because these

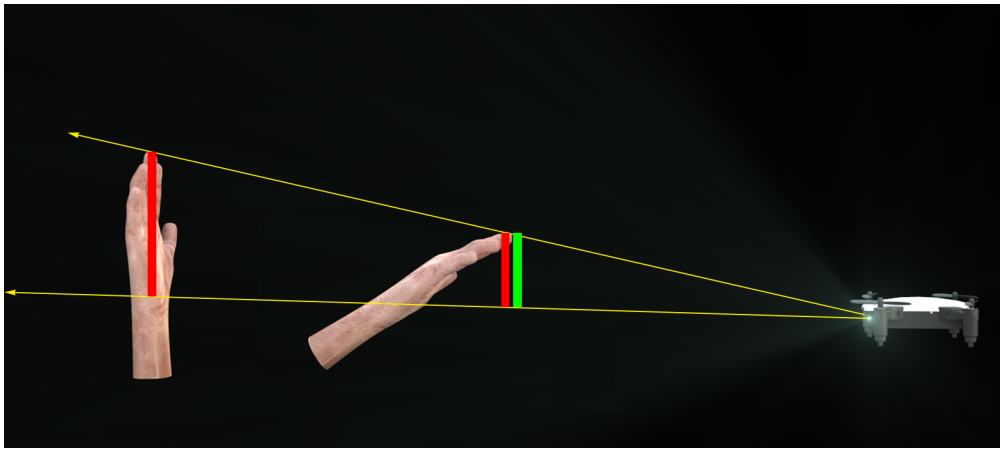


Figure 4.11. Depth estimate - Different orientation. On the left is the first h_1 hand captured at time t_1, in the middle is the h_2 hand captured at time t_2. Here, the hand in addition to moving forward has also changed its orientation in space. We note that the projection of the 2D points of h_1 in h_2 have a distance (red segment) equal to the points in h_2 (green segment). So we can not compare the two results through this metric. Image rendered with Blender.

can be altered due to the orientation of the hand (see Fig. 4.11). This is why the orientation of the hand has to be estimated in space, because in this way the points that describe the first hand detected can be projected. Thus, bringing it back to the previous case, comparing areas of the hand to estimate depth.

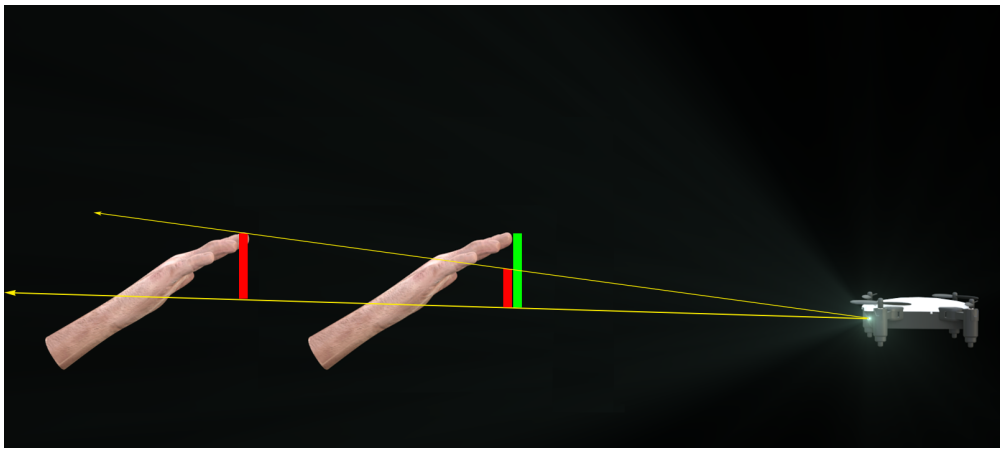


Figure 4.12. Depth estimate - 3D rotation. On the left is the first h_1 hand captured at time t_1, in the middle is the h_2 hand captured at time t_2. Here the hand h_1 is oriented in space using the estimation of the orientation on the hand h_2. We note that the projection of the 2D points of h_1 in h_2 have a distance (red segment) less then that of the points in h_2 (green segment). So now we are able to compare the two results through this metric. Image rendered with Blender.

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 NN, here the points were not scaled w.r.t. the maximum distance between the midpoint and all other points of the hand. This is not done 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 the depth value is calculated by comparing the first hand acquired with the one in progress 4.12. Specifically, at the points that define the first hand acquired a matrix transformation was implemented: a 3D rotation along the three 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_Z, R_Y and R_X represent transformations that rotate points through an angle θ in radians about the coordinate origin. Borrowing aviation terminology, these rotations will be referred to as yaw, pitch, and roll [**?**].

$$R_Z(\alpha) = \begin{pmatrix} cos\alpha & -sin\alpha & 0 & 0 \\ sin\alpha & cos\alpha & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.2}$$

Equation 4.2. Z-Rotation Matrix in homogeneous coordinates. A yaw is a counterclockwise rotation of α about the z-axis.

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

Equation 4.3. Y-Rotation Matrix in homogeneous coordinates. A pitch is a counterclockwise rotation of β about the y-axis.

It must be further defined whether positive angles perform a clockwise (CW) or coun-

$$R_X(\gamma) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\gamma & -sin\gamma & 0 \\ 0 & sin\gamma & cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.4}$$

Equation 4.4. X-Rotation Matrix in homogeneous coordinates. A roll is a counterclockwise rotation of γ about the x-axis.

terclockwise (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. The yaw, pitch, and roll rotations can be used to place a 3D body in any orientation. A single rotation matrix R can be formed by multiplying the yaw, pitch and roll rotation matrices (see Eq. 4.5).

$$R(\alpha,\beta,\gamma) = R_X(\alpha)\cdot R_Y(\gamma)\cdot R_Z(\beta) = \begin{pmatrix} cos\alpha\, cos\beta & cos\alpha\, sin\beta\, sin\gamma - sin\alpha\, cos\gamma & cos\alpha\, sin\beta\, cos\gamma + sin\alpha\, sin\beta & 0 \\ sin\alpha\, cos\beta & sin\alpha\, sin\beta\, sin\gamma + cos\alpha\, cos\gamma & sin\alpha\, sin\beta\, cos\gamma - cos\alpha\, sin\gamma & 0 \\ -sin\beta & cos\beta\, sin\gamma & cos\beta\, cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.5}$$

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

It is important to state that $R(\alpha,\beta,\gamma)$ performs the roll first, then the pitch, and finally the yaw. If the order of these operations is changed, a different rotation matrix would result.

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 has been finally made. In this way, it is possible obtain a depth estimate of the hand (which is performing the detecting gesture) even if it is oriented in space.

4.2.3 Errors

The trajectory acquired in the previous phase is pure and for this reason it presents some noise. This noise is conditioned by a series of dependent events. In fact, at the base of everything, there is the error given by the precision with which the "detect" gesture is detected by the Mediapipe solution. In fact, although the classifier has a high level of accuracy, being a solution in real time and designed to work on smartphone devices, it has limits dictated by the computational components that means that the classifier cannot have an architecture that is too complex, and therefore this translates that in some frames the hand landmark is not correctly captured. Above this first level of error, there are three other incorrectness which are connected to the estimates of the orientation components. These inaccuracy are the ones bring more problems to the whole system. Finally, the error that describes the correctness with which the depth of the points of the trajectory was estimated was found. The latter represents the inaccuracy that reports the greatest inaccuracy to the system because it is a combination of the four errors described above.

In the idea of a practical application, the acquisition of trajectories can be done from a computer webcam or from a mobile phone camera. In this case it is possible to speak of acquisition with a fixed camera in space. Another level of application could be that of direct interaction with the drone, therefore with a potentially moving camera. Although the drone can be programmed to make it remain stable in one point in space, it does not imply that it is also capable of withstanding atmospheric agents, specifically the wind. In the latter use case, an additional level of error is added to the base of the whole system. it is easy to understand that the error is not something negligible and that it must necessarily be managed.

4.2.4 Smoothing

In the different areas of science, smoothing a dataset means creating an approximating function that attempts to capture important patterns in the data, while leaving out noise or other fine-scale structures/rapid phenomena. In smoothing, because of noise, the data points of a signal are modified so individual points higher than the adjacent points are reduced, and points that are lower than the adjacent points are increased leading to a smoother signal. Smoothing may be used in two important ways that can aid in data analysis by being able to extract more information from the data as long as the assumption of smoothing is reasonable and able to provide analyses that are both flexible and robust.

The trajectory $G(x, y, z)$ acquired in a certain time t, two different smoothing algorithms have been computed on it: spline and ridge regression. Splines are mathematical functions that describe an ensemble of polynomials which are interconnected with

each other in specific points called the knots of the spline. They are used to interpolate a set of data points with a function that shows a continuity among the considered range; this also means that the splines will generate a smooth function, which avoids abrupt changes in slope. Compared to the more classical fitting methods, the main advantage of splines is that the polynomial equation is not the same throughout the whole range of data points. Instead, the fitting function can change from one interval to the subsequent one, allowing for fitting and interpolation of very complicated point distributions.

One of the main characteristics of splines is their continuity; they are continuous along the entire interval in which they are defined; this allows for the generation of a smooth curve, that fit our set of data points. One of the main advantages of using splines for fitting problems, instead of single polynomials, is the possibility of using lower degree polynomial functions to describe very complicated functions. Indeed, if a single polynomial function was needed, the degree of the polynomial usually would increase with the complexity of the function that has to be described; increasing the degree of the fitting polynomial could introduce the Runge's phenomenon problem (see Fig 4.13), in which oscillation can occur between points when interpolating using high-degree polynomials.

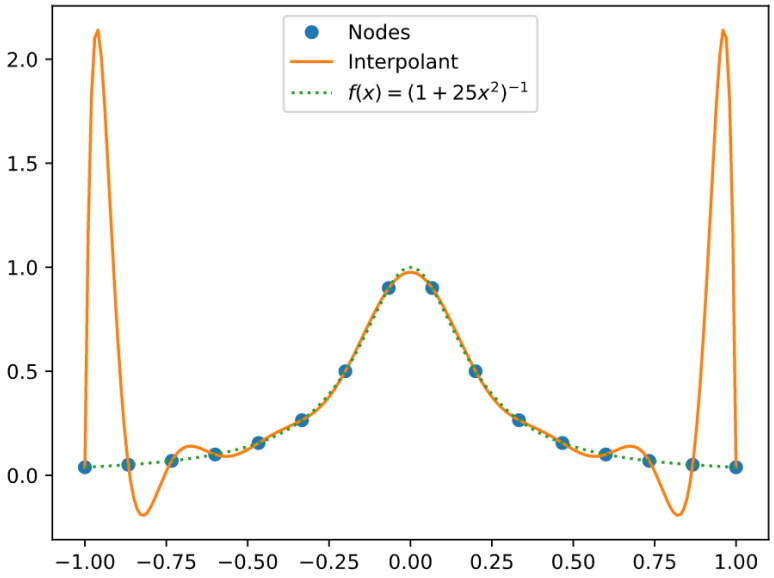


Figure 4.13. Runge's phenomenon problem.

From the Scipy package there is the function UnivariateSpline() to calculate 1D smoothing spline $y = spl(x)$ of degree k to the provided x, y data. One of the most important parameters is k, which refers to the degree of the polynomials that define the spline

segments and it varies between one and five (e.g. $k = 3$ is a cubic spline). Increasing the degree of the polynomials allows a better fitting of more complicated functions, but in order not to introduce artifacts in the fit, the best practice is to use the lower degree that allows for the better fitting procedure. Another relevant parameter is s, it is a float number which defines the smoothing factor, which directly affects the number of knots present in the spline. Once a specific value of s is fixed, the number of knots will be increased until the difference between the value of the original data points and their respective datapoints along the spline is less than the value of s.

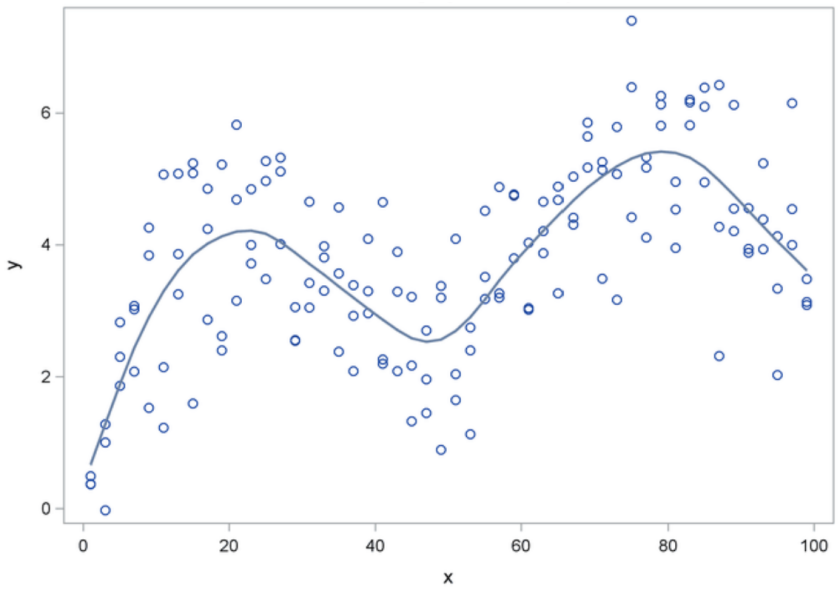


Figure 4.14. Smoothing spline 1D

Another technique of data fitting is Linear regression. Linear regression refers to a model that assumes a linear relationship between input variables and the target variable. With a single input variable, this relationship is a line, and in higher dimensions it can be thought of as a hyperplane that connects the input variables to the target variable. The coefficients of the model are found via an optimization process that seeks to minimize the SSE between the predictions and the expected target values. A problem with linear regression is that estimated coefficients of the model can become large, making the model sensitive to inputs and possibly unstable, especially for problems with few samples or less samples than input variables.

One approach to address the stability of regression models is to change the loss function to include additional costs for a model that has large coefficients. Linear regression models that use these modified loss functions during training are referred to collectively as penalized linear regression. One popular penalty is to penalize a model based on the

sum of the squared coefficient values (beta). This is called an $L2$ penalty. The effect of this penalty is that the parameter estimates are only allowed to become large if there is a proportional reduction in SSE. In effect, this method shrinks the estimates towards 0 as the lambda penalty becomes large (these techniques are sometimes called "shrinkage methods"). This penalty can be added to the cost function for linear regression and is referred to as Tikhonov regularization or Ridge Regression more generally. A hyperparameter is used called "lambda" that controls the weighting of the penalty to the loss function. A value of 1 will fully weight the penalty; a value of 0 excludes the penalty. Very small values of lambda, such as $1e-3$ or smaller are common. From the Scipy package there is the function Ridge() to solve a regression model where the loss function is the linear least squares function and regularization is given by the $L2$-norm.

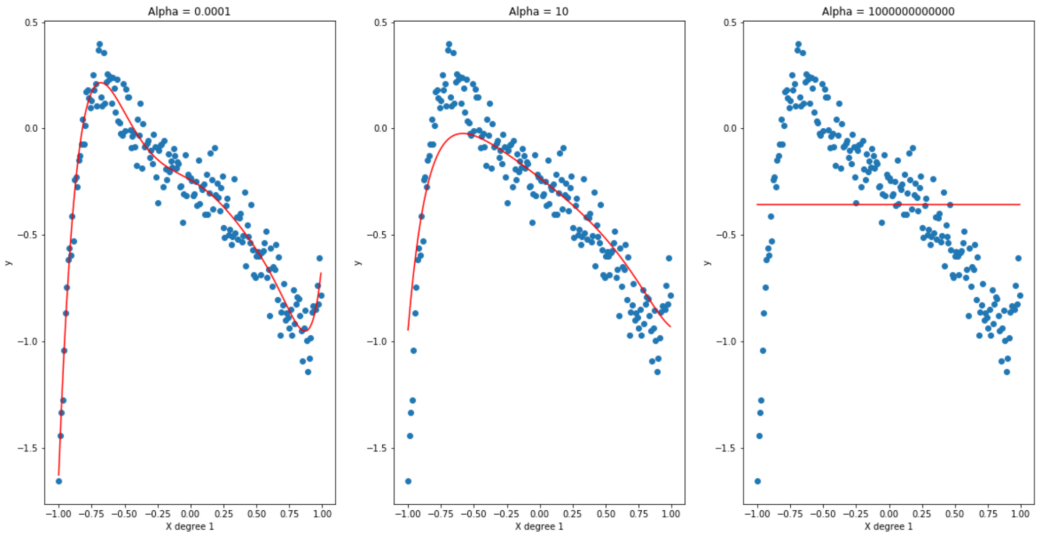


Figure 4.15. Sample fits on a high degree polynomial function with some random noise. As alpha grows larger the model will overgeneralize (under-fit), while at lower values of alpha, the model may be too specific (overfit).

Another way to see the effectiveness of this method is plot the ridge coefficients as a function of the regularization (see Fig. 4.16). Each color represents a different feature of the coefficient vector, and this is displayed as a function of the regularization parameter. A slight change in the target variable can cause huge variances in the calculated weights. In such cases, it is useful to set a certain regularization (lambda) to reduce this variation (noise). When lambda is very large, the regularization effect dominates the squared loss function and the coefficients tend to zero. At the end of the path, as alpha tends toward zero and the solution tends towards the ordinary least squares, coefficients exhibit big oscillations. In practise it is necessary to tune alpha in such a way that a balance is maintained between both.

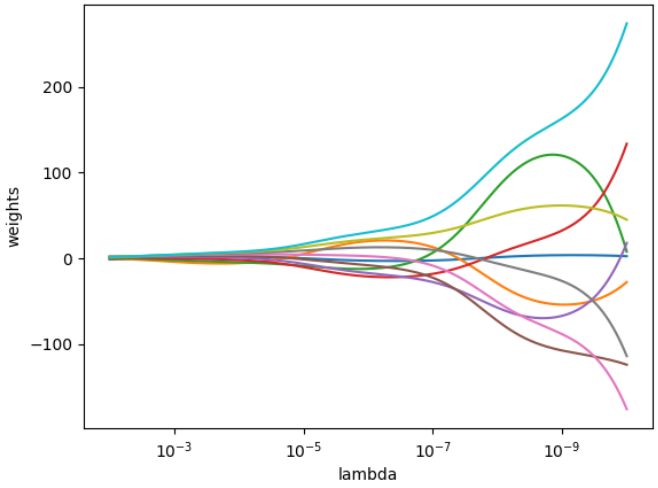


Figure 4.16. Ridge coefficients as a function of the regularization, image from scikit-learn website [a].

[a] https://scikit-learn.org/stable/auto_examples/linear_model/plot_ridge_path.html

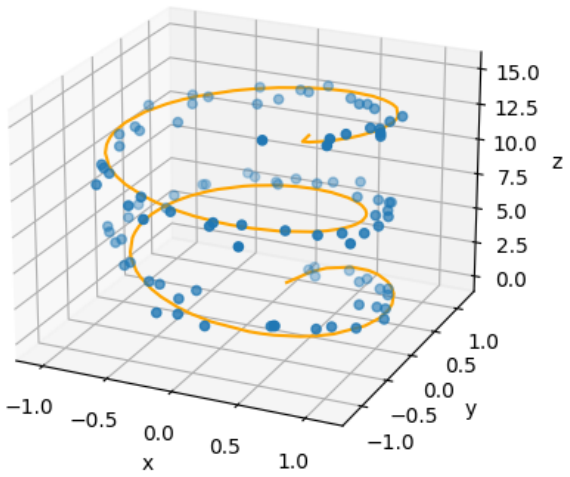


Figure 4.17. Multiple ridge regression with $\lambda = 1e - 10$ as hyperparameter for all the three ridge regression to compute x, y, z values.

At the implementation level, given the trajectory $G(x, y, z)$ the cumulative sum of distances between each pair of adjacent points was calculated. This gives a vector w. Given a degree k, $X = w^2, w^3, ..., w^k$ is computed. After that, 3 ridge regression (or 3 splines, approach is the same) were calculated, where for the first the inputs given are X and the vector of all the components x of the trajectory $G()$, for the second the inputs X and the vector of all the y components of the trajectory $G()$, and for the last the inputs are X and the vector z of all the components of the trajectory $G()$.
Given a set of evenly spaced numbers over a specified interval r, the $X_{new} = r^2, r^3, ..., r^k$ was computed, and then it was used as input for each ridge (or spline) previously computed, thus obtaining $x_{pred}, y_{pred}, z_{pred}$. An example of the result is visibile in 4.17.

4.3 Drone Controller

Once the trajectory was captured, the next step was first to launch it in simulation as described in 4.3.1 and then on a real drone as illustrated in 4.3.2.

4.3.1 Simulation

The hardest part of learning about flying robots is the constant crashing. From learning flight control for the first time to testing new hardware or flight algorithms, the resulting failures can have a huge cost in terms of broken hardware components. To avoid such costs, it is ideal to simulate air vehicles designed and developed for ROS. Many of the most useful capabilities of ROS already exist somewhere in its community: often, stable resources exist. Alternately, some resources are less tested or more “cutting edge” and have not reached a stable release state.

In order to perform the simulation in Gazebo, Github was looking for a ROS package that had the 3D model of the Tello drone. Following the research, a project was found named "tello_ros_gazebo" by the author "bingyo" [2]. The project requires the following dependencies to be executed:

- `geographic_info`;
- `hector_gazebo`;
- `hector_localization`;
- `hector_models`;
- `hector_quadrotor`;

[2] `https://github.com/bingyo/tello_ros_gazebo`

- `hector_slam`;
- `orb_slam_2_ros`;
- `unique_identifier`;
- `ros_noetic-octomap`.

These packages among various things, allow to model, control and simulate quadrotor Unmanned Aerial Vehicles (UAV) systems. A simulated quadrotor UAV for the ROS Gazebo environment has been developed by Team Hector of Technische Universitat Darmstadt. This quadrotor, called Hector Quadrotor, is enclosed in the hector_quadrotor metapackage. This metapackage contains the Unified Robot Description Format (URDF) description for the quadrotor UAV, its flight controllers, and launch files for running the quadrotor simulation in Gazebo. Advanced use of the Hector Quadrotor simulation allows the user to record sensor data such as Lidar, depth camera, and so on. The quadrotor simulation can also be used to test flight algorithms and control approaches in simulation.

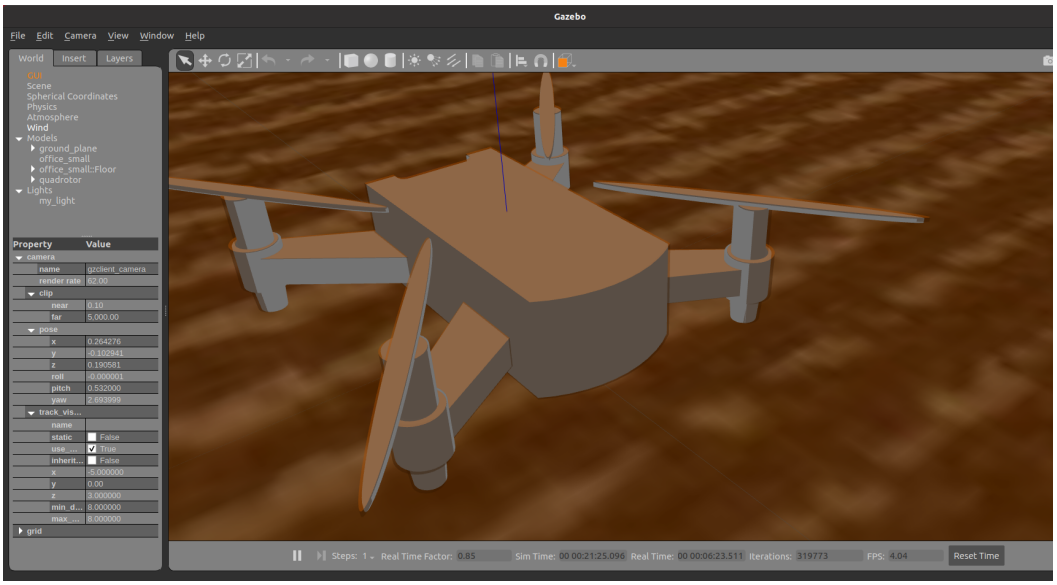


Figure 4.18. Drone in Gazebo.

To remotely control the mobile robot, several ROS nodes need to be run: the first node is ROS master node. The second node is a standard node called teleop_twist_keyboard [3]. This latter node is constantly checking which keys are pressed on a PC keyboard and

[3] `https://github.com/ros-teleop/teleop_twist_keyboard`

based on the keys pressed, publishes twist messages on /cmd_vel topic. Twist message defines what should be the linear and rotational speeds of a mobile robot.
Rospy provides a rospy.Rate convenience class which makes a best effort at maintaining a particular rate for a loop. The Rate instance will attempt to keep the loop at N Hz by accounting for the time used by any operations during the loop. Rate.sleep() can throw a rospy.ROSInterruptException if the sleep is interrupted by shutdown. Basically, it allows loops to run at (nearly) the exact rate (in Hz) that has been specified. It's a pretty standard way of looping with ROS. The line "rate = rospy.Rate(20)" creates a Rate object rate. With the help of its method sleep(), it offers a convenient way for looping at the desired rate. With its argument of 20, it is expectd to go through the loop 20 times per second (as long as our processing time does not exceed $1/10th$ of a second).

Given the $G_{smooth}(x, y, z)$, this trajectory has been scaled twice and the horizontal component ultimately scaled $x1.5$. The time related to each point of $G_{smooth}()$ was halved because rospy.Rate (20) was used.
Knowing that Rate (20) has been set, then each $step = 0.05s$ a cycle is performed. So, the speed that the robot will take at that specific moment plus the next step can be calculated, using Inverse Kinematics. More precisely, if the Euclidean distance d between the current and the target position is known, then the speed s (approximately instantaneous) will be $s = d/step$. But how is it possible to calculate the goal position at time t_1 if there is no such point t_1 in the trajectory $G_{smooth}()$? This is possible through interpolation: a type of estimation, a method of constructing new data points based on the range of a discrete set of known data points [**?**].

The velocity is computed for each x, y, and z component. At first the drone is made to take off, when it reaches a certain height velocity is repeatedly set through the self.vel_msg command.

4.3.2 Real World

After carrying out the simulation experiments in the Gazebo environment, the DJI Ryze Tello drone (see Sec. 3.1) was used to test the application in real life. DJITelloPy is the DJI Tello drone python interface, used in this project to make the application communicate with the drone. To create a connection with the drone, first connect the application to the Tello network is required. Then, programmatically, the me = Tello() object is created, and finally the me.connect() function is called to actually make the connection.

At the beginning, the tests were carried out at home, with the intention of becoming familiar with the drone at the programming level. It was soon realized that some functions of the Framework, even if they allowed a type of communication with response, with the certainty that the command had been received and executed by the robot,

could not really be used because such communications would actually have made the system too much slow. The functions whose problems have just been described are for example $up(x)$, $down(x)$, $left(x)$, $right(x)$, $forward(x)$ or $back(x)$, where x is a value between $20cm/s$ and $50cm/s$ [**?**]. The only function able to keep up with the speed of execution was Send_rc_control. The latter, allows you to set the speed through 4 channels: left-right, forward-backward, up-down and yaw velocity. The possible input values for each channel range from $-100cm/s$ to $100cm/s$.

The acquisition time of each point has been shifted so that the first point of the trajectory has time equal to zero. Let $p = (x, y, z) \in G_{smooth}()$ the x and y points are between 0 and 1. To have a more faithful trajectory scale the horizontal axis so as to be in line with the aspect ratio of the camera is necessary. To do that $ar = width/height$ and then y/ar, for each point. Then, we scale all the points $x100$, so that the trajectories move in a maximum range of $0cm-100cm$. In order to calculate velocities, space from a point a to b(in cm) divided the time spent switching from a to b was executed for each pair of consecutive points. At this point, it would potentially be enough to send the speed to the drone every time the application is updated.

4.4 Pipeline

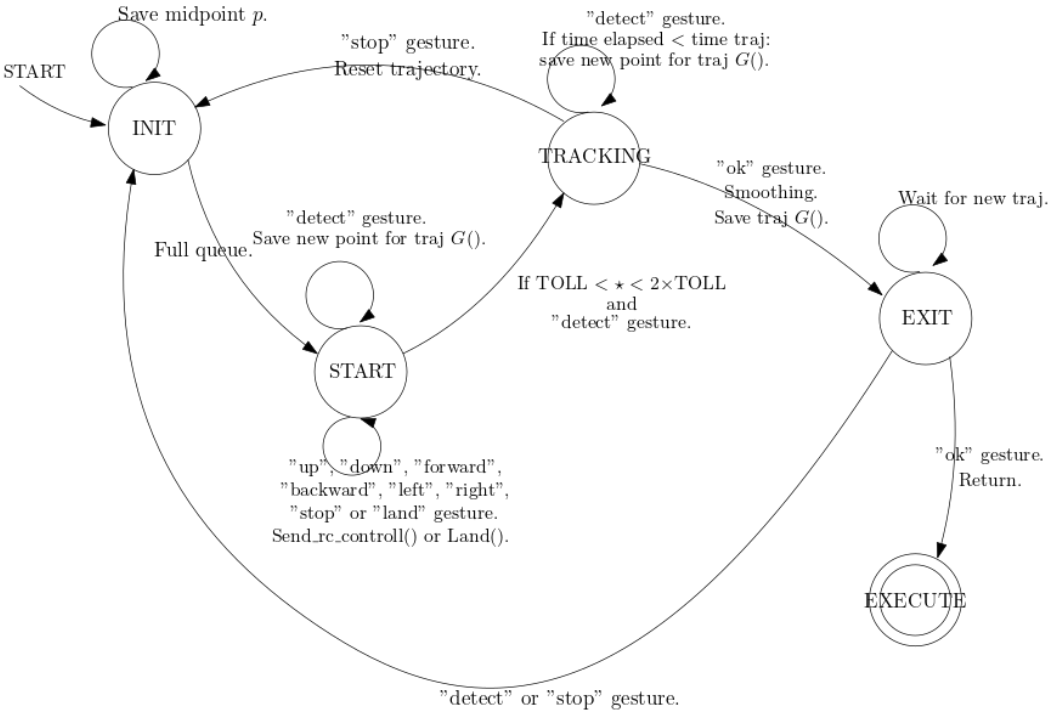


Figure 4.19. Pipeline.

In image 4.19, there is a deterministic finite state automaton. It represents the process of human-drone interaction, both taking into account the direct interaction: sending the left, right, up, down, forward, backward and land commands; and not direct: acquiring a 3d trajectory. It consists of 5 states, an initial "INIT" state and a final "EXECUTE" state.

Before proceeding with the description it is necessary to define what is the circular queue, midpoint p, average midpoint b, $\star$ and $TOLL$. Circular queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called "Ring Buffer". In this queue, each update of the application saves the gesture class, the probability of that gesture and finally the middle point p. To have a more or less precise estimate of the current gesture we proceed by looking at the last n gestures acquired and saved in a circular tail. This queue has a maximum limit of items that can be saved, then new gestures are overwritten on older ones. The midpoint p is the average distance among all the reference points of the hand. The average midpoint b, is the mean of all midpoint p saved in the queue. $\star$ is the distance between the current midpoint p and b. $TOLL$ is computed at the beginning of "START" state. It represents the mean of all distances from the current midpoint p and all hand landmark, plus a little extra value (in the code set as 20). This is the tolerance where as soon as it is overcome, then the change of state takes place.

In the initialization phase, the midpoint p, gesture class and associated probability information are saved in the queue. When the queue fills up completely, that is, it performs its first complete turn, it passes from the "INIT" state to the "START" state. In the latter state, if the "detect" gesture continues, then a new point is computed for the trajectory $G()$. Otherwise, if "up", "down", "forward", "backward", "left", "right", "stop" gestures are performed then it is possible to interact directly with the drone through the Send_rc_controll() function. If the "land" gesture is executed then the drone lands through the land() command. These actions always keeps the "START" status. To get out of this state it must happen that $TOLL < \star < 2 \times TOLL$ and the gesture executed is "detect" gesture. This leads to "TRACKING" status. Here, points can be collected for trajectory $G()$, as long as the time elapsed is less than the set total trajectory acquisition time. After this time, no more points are collected, but the system continues to remain in this state. To exit there are two different ways: eliminate the trajectory by performing the "stop" gesture; perform the "ok" gesture proceeding in the direction of execution of the trajectory $G()$. In the first case there is the transition from the "TRACKING" state to the "INIT" state, in the second case from the "TRACKING" state to the "EXIT" state. Before reaching the "EXIT" state, the trajectory is first cleared of noise, through smoothing, and then saved in memory. In this state the system remains stationary until a fixed time passes (in the code around 1.5 seconds) this because it might be useful to elimi-

nate the trajectory at the last second by making the "detect" or "stop" gesture. If instead passed the waiting time continues to be executed the "ok" gesture, then the trajectory is performed, returning the gesture and launching it on the drone.

In order to have everything safe and secure, as soon as the flight is completed, it is good practice to back up the footage transferring the data to a hard drive. A solution to a problem that can cause a lot of money to rectify. To automate this process, it would be interesting to send videos to the application. DJI Ryze Tello allows video streaming and for the benefits of the application the streaming is recorded. However, streaming results in a loss of quality.

Chapter 5

Evaluation

In this chapter, the results obtained from the three models created for hand gesture recognition and those for the estimates of the 3d trajectories will be displayed and commented in 5.1. Then an analysis is made on the 3D trajectories both in simulation and in real 5.2.

5.1 Hand Gesture Recognition Model Evaluation

As said in 4.1.4 three models were generated using the same hyper-parameters: one using the selected features, one using PCA, and one using all the features.

Metric is different from loss function: loss functions are functions that show a measure of the model performance, are used to train a machine learning model (using some kind of optimization), and are usually differentiable in model's parameters. On the other hand, metrics are used to monitor and measure the performance of a model and do not need to be differentiable. However, if for some tasks the performance metric is differentiable, it can be used both as a loss function and a metric, such as MSE.

When the class distribution is imbalanced, that is one class is more frequent than others, accuracy is not a good indicator of model performance. In this case, even if all samples are correctly predicted does not make sense at all because the model is not learning anything, in fact it is just predicting everything as the top class. Therefore, look at class specific performance metrics too is needed.

Precision is one of such metrics and answers the question of "what proportion of predicted positives are truly positive?" Obviously, this can only answer the question in binary classification. This is why is important to ask the question as many times as the number of classes in the target. Optimizing the model for precision when is fundamental to decrease the number of false positives.

Recall answers the question of "what proportion of actual positives are correctly classified?" Optimize the model for precision when is fundamental to decrease the number of false negatives.

Due to their nature, precision and recall are in a trade-off relationship. It may be necessary to optimize one at the cost of the other. This is where the f1-score comes in: it is calculated by taking the harmonic mean of precision and recall and ranges from 0 to 1. Harmonic mean has a nice arithmetic property representing a truly balanced mean. If either precision or recall is low, it suffers significantly.

Support is the number of actual occurrences of the class in the test data set. Imbalanced support in the training data may indicate the need for stratified sampling or re-balancing.

Confusion matrix, also known as error matrix, is a tabular visualization of the model predictions versus the ground-truth labels. Diagonal elements of this matrix denote the correct prediction for different classes, while the off-diagonal elements denote the samples which are mis-classified.

The Receiver Operating Characteristic (ROC) curves are typically used in binary classification to study the output of a classifier **?**. It is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied. It is created by plotting the fraction of true positives out of the positives (TPR = true positive rate) vs. the fraction of false positives out of the negatives (FPR = false positive rate), at various threshold settings. TPR is also known as sensitivity, and FPR is one minus the specificity or true negative rate. The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes. It is used as a summary of the ROC curve. The higher the Area Under the ROC Curve (AUC), the better the performance of the model at distinguishing between the positive and negative classes. When AUC=1, the classifier is able to perfectly distinguish between all the Positive and the Negative class points correctly. If, however, the AUC had been 0, then the classifier would be predicting all Negatives as Positives, and all Positives as Negatives. When 0.5 <AUC< 1, there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values. When AUC=0.5, then the classifier is not able to distinguish between Positive and Negative class points. Meaning either the classifier is predicting random class or constant class for all the data points. The AUC curve is only for binary classification problems, but multi-class classification problems by using the One-vs-Rest technique can be extended. For each class i, the classification i against all other classes, except i, will be generated.

For each model generated the previous metrics will be calculated.

5.1.1 Model Evaluation - Feature Selection

Not all variables were used in the feature selection model, but only 9, following a correlation analysis as described in 4.1.2. After training the model with 749 examples (using only 9 variables), the test data, composed of 251 examples and obtained from the train test split, to predict the output was used. After, the predicted classes and the actual output classes from the test data to visualize the confusion matrix were used. In the image 5.1, its confusion matrix.

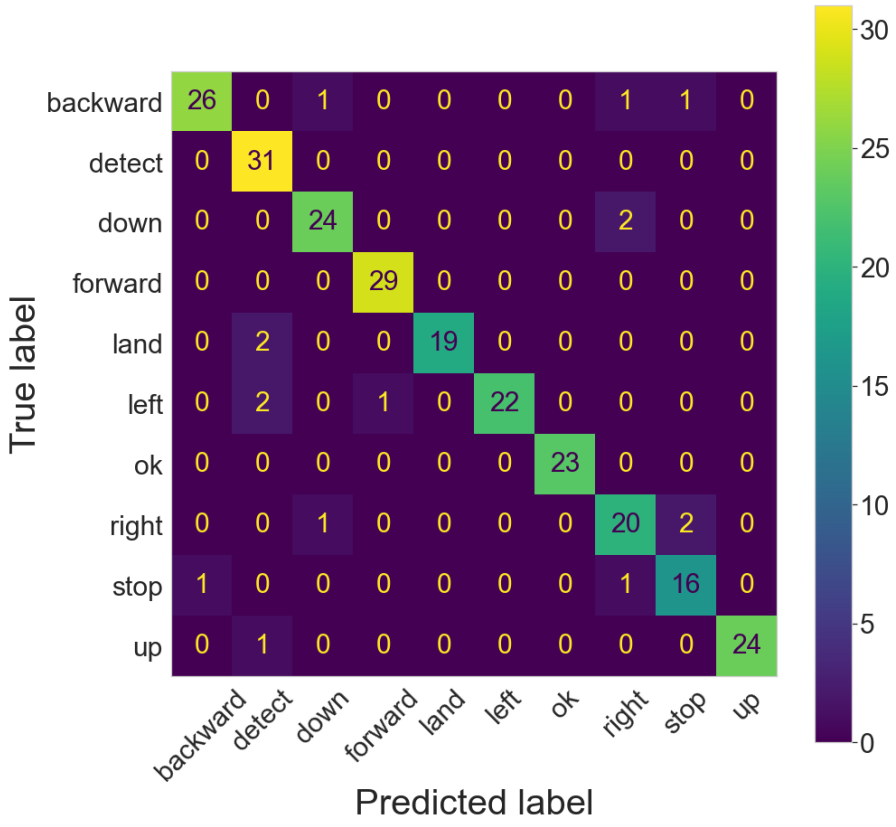


Figure 5.1. Confusion matrix - Features selected.

For "detect", "forward" and "ok" gestures all the corresponding tests have been correctly predicted. In parallel, we noted that for "left" and "right" gestures there were 22 and 20 cases respectively classified correctly. At the same time, for both cases three examples were not classified correctly.

In 5.2 plot there are precision and recall values for each class. Note that the values are rather balanced and generally higher than 80%. This means that both the proportion

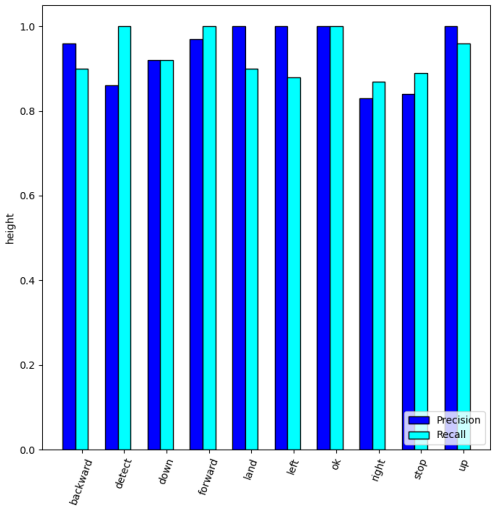


Figure 5.2. Bar-plot for Precision and Recall - Features selected.

of positive forecasts when true positives and the proportion of actual positives correctly classified are high.

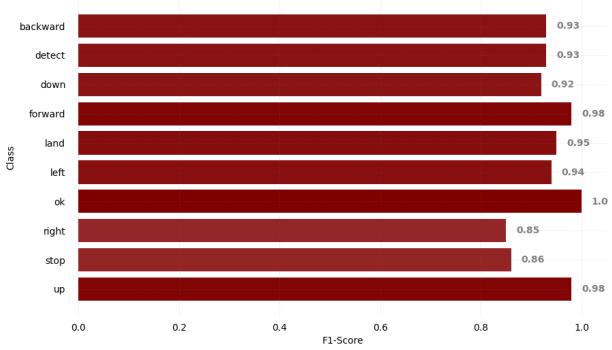


Figure 5.3. Horizontal bar chart for F1 score - Features selected.

From 5.3, it is possible to see that the classifier works better to classify "ok" gesture since it reached 100% as f1-score. In fact this represents the maximum of all the graph. The smallest value achieved is "right" gesture with a value of 85%.

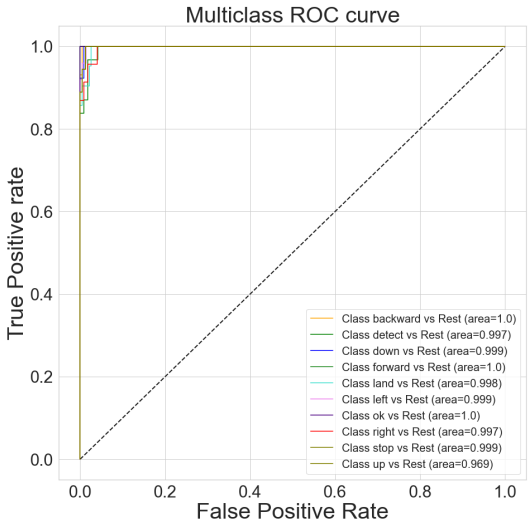


Figure 5.4. ROC curves for the multiclass problem - Features selected.

Through 5.4 is it possible to see 10 ROC curves related to each variable vs all others, hence, to express the behavior between one variable and all the others. Specifically, we note that for each ROC the value of AUC is always higher than 96%, then the classifier is able to perfectly distinguish between all the Positive and the Negative class points correctly. The one-vs-rest AUC score is 0.995 weighted by prevalence.

	precision	**recall**	**f1-score**	**support**
backward	0.96	0.90	0.93	29
detect	0.86	1.00	0.93	31
down	0.92	0.92	0.92	26
forward	0.97	1.00	0.98	29
land	1.00	0.90	0.95	21
left	1.00	0.88	0.94	25
ok	1.00	1.00	1.00	23
right	0.83	0.87	0.85	23
stop	0.84	0.89	0.86	18
up	1.00	0.96	0.98	25
accuracy			0.94	250
macro avg	0.94	0.93	0.93	250
weighted avg	0.94	0.94	0.94	250

Table 5.1. Metrics - Feature Selected.

In table 5.1 precision, recall and f1-score metrics are visible for each classification object. For precision and recall, "right" and "stop" gestures reach the lowest values among all classes. In fact, this is reflected in the value of f1-score. Despite this, the values are still very high.

5.1.2 Model Evaluation - PCA

Here the large dataset has been transformed into a small one through a linear transformation, bringing the old data into a new set of uncorrelated variables. The model with 749 examples was also trained here. After that, the test data composed of 251 examples to predict the output was used. This test set before being input to the model, is first transformed with the same transformation applied for the training set. Then we proceeded to visualize the confusion matrix. High accuracy in the "PCA" model was encountered, although only the first six principal components have been selected. Here there are slightly less satisfying predictions then the feature selected model.

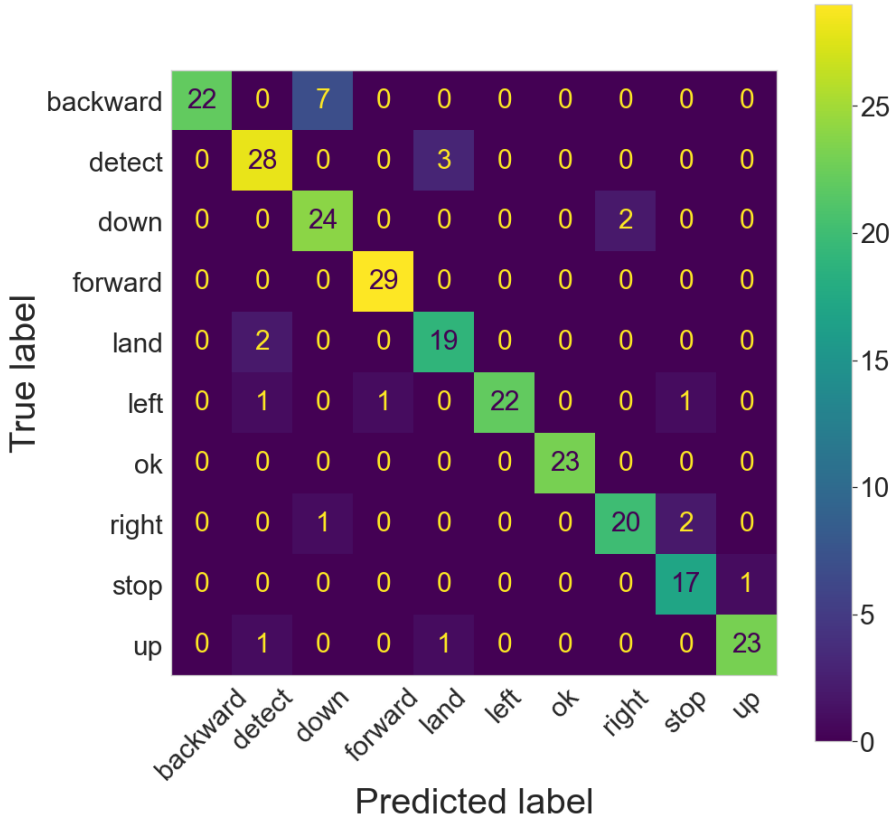


Figure 5.5. Confusion matrix - PCA.

In the confusion matrix (see Fig. 5.5) all the corresponding tests have been correctly predicted for "forward" and "ok" gestures, but seven examples labeled as "backward" were instead predicted as "down". The results of the predictions related to the other gestures are better. The feature selection model achieved small percentage points higher in accuracy than PCA.

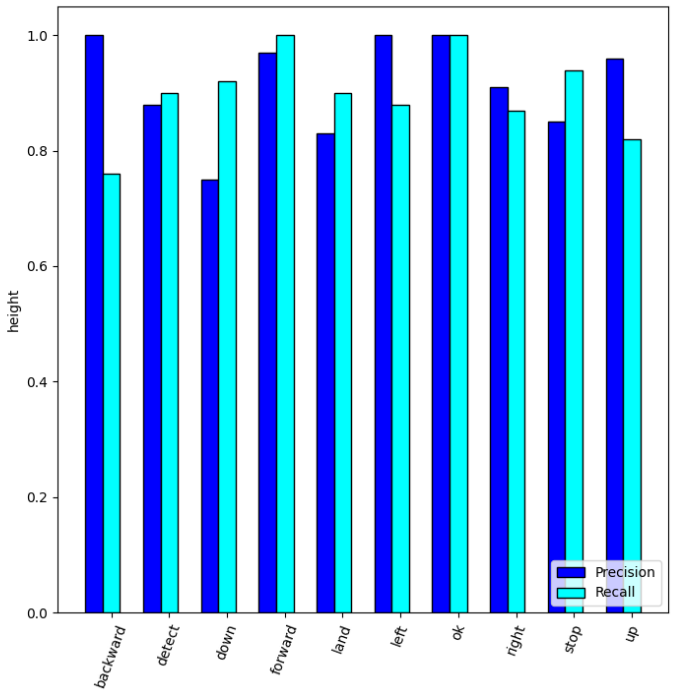


Figure 5.6. Bar-plot for Precision and Recall - PCA.

In 5.6 plot there are precision and recall values for each class. Here is a slight discrepancy between the results obtained. In particular between "backward" and "down" gesture. Despite this, all precision results are above 75% and all results for recall are also greater than 75%. In both cases the maximum possible peaks are reached. This means that both the proportion of positive forecasts when true positives and the proportion of actual positives correctly classified are high.

From 5.7, it is possible to see that the classifier works better to classify "ok" gesture since it reached 100% as f1-score. In fact this represents the maximum of all the graph. This is the same maximum achieved in the model with selected features. The smallest value achieved is "down" gesture with a value of 83%. A minimum slightly lower than

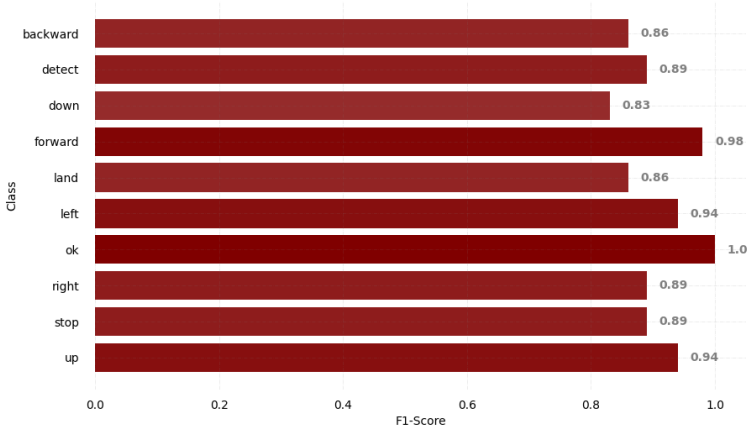


Figure 5.7. Horizontal bar chart for F1 score - PCA.

that in selected features.

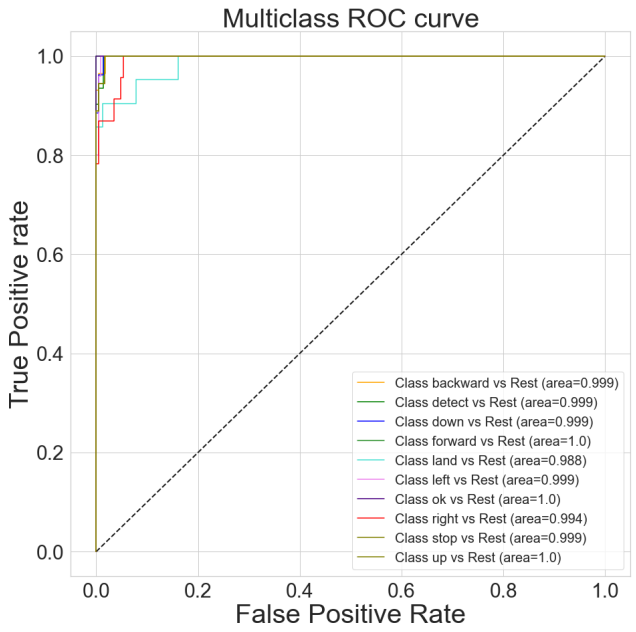


Figure 5.8. ROC curves for the multiclass problem - PCA.

Through 5.8 is it possible to see 10 ROC curves related to each variable vs all others, hence, to express the behavior between one variable and all the others. Specifically, we note that for each ROC the value of AUC is always higher than 98%, then the classifier

is able to perfectly distinguish between all the Positive and the Negative class points correctly. The one-vs-rest AUC score is 0.997 weighted by prevalence.

	precision	**recall**	**f1-score**	**support**
backward	1.00	0.76	0.86	29
detect	0.88	0.90	0.89	31
down	0.75	0.92	0.83	26
forward	0.97	1.00	0.98	29
land	0.83	0.90	0.86	21
left	1.00	0.88	0.94	25
ok	1.00	1.00	1.00	23
right	0.91	0.87	0.89	23
stop	0.85	0.94	0.89	18
up	0.96	0.92	0.94	25
accuracy			0.91	250
macro avg	0.91	0.91	0.91	250
weighted avg	0.92	0.91	0.91	250

Table 5.2. Metrics - PCA.

Also in 5.2 precision, recall and f1-score metrics are visible for each classification object. This time, for precision, "down" and "land" gestures reach the lowest values among all classes with 75% and 83% respectively, even if their recall values are quite close to the average recall. Max precision is reached with the "backward", "left" and "ok" gestures. With "backward" gesture recall is the minimum, while "ok" gesture is always predicted correctly, in fact for the latter f1-score has the maximum value.

5.1.3 Model Evaluation - All Features

Unlike the previous cases, no transformation and no features selection were performed in the dataset. The original dataset was used here. Also here the model with 749 examples was trained. After that, the test data composed of 251 examples to predict the output was used. From the results obtained, the report is extremely positive.
The highest level of accuracy was found using all features. The model is 2% more accurate than feature selection and 5% more accurate than PCA. In the confusion matrix (see Fig. 5.9) for "backward", "forward", "left", "ok" and "stop" gestures all the corresponding tests have been correctly predicted. Problems of little importance in prediction in all other cases.

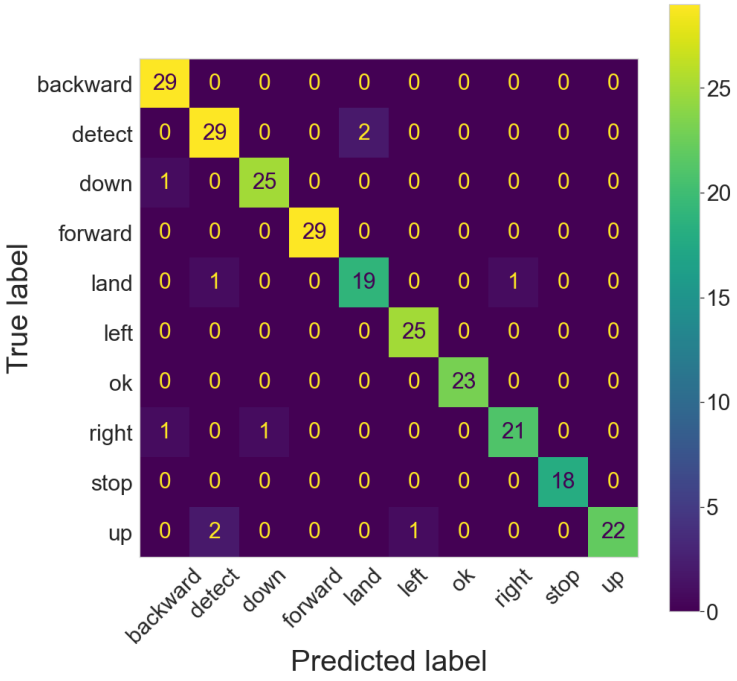


Figure 5.9. Confusion matrix - All features.

In 5.10 plot there are precision and recall values for each class. Both the precision and recall values are 95% higher, and this is true for each class. The model reaches the peak of precision and recall with "forward", "ok" and "stop" gestures. This means that both the proportion of positive forecasts when true positives and the proportion of actual positives correctly classified are high.

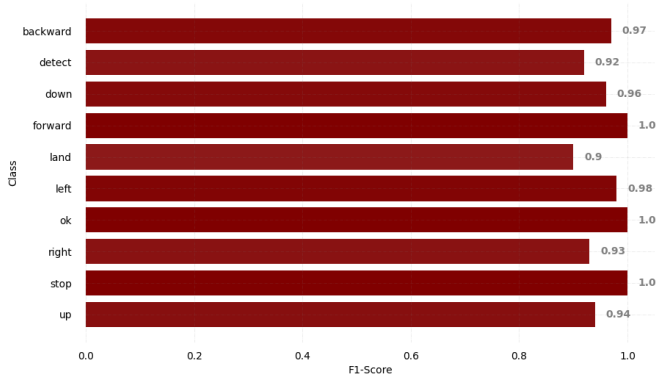


Figure 5.11. Horizontal bar chart for F1 score - All features.

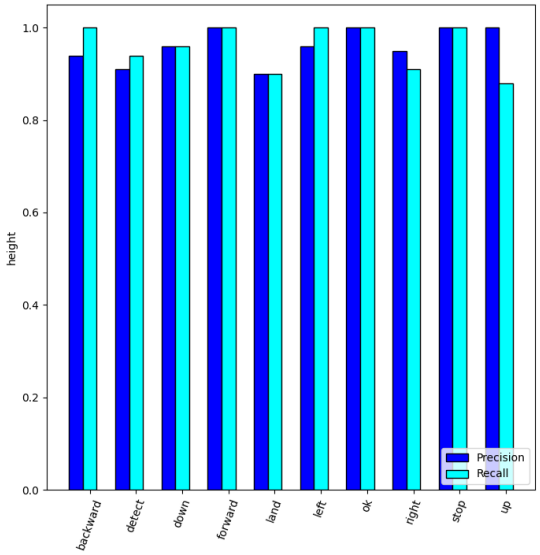


Figure 5.10. Bar-plot for Precision and Recall - All features.

From 5.11, it is possible to see that the classifier works perfectly to classify more than a single gesture. In fact, with "forward, "ok" and "stop" gestures the model reached 100% as f1-score, as they represent the maximum of all the graph. The smallest value achieved is "land" gesture with a value of 90%. All values were satisfactory.

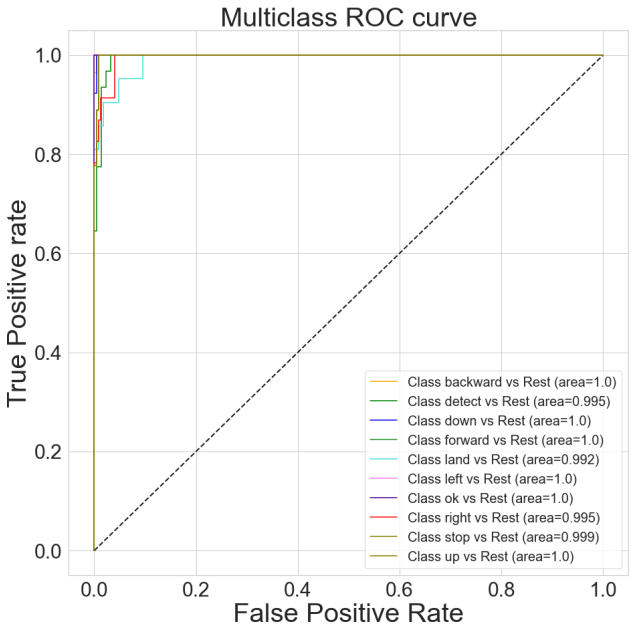


Figure 5.12. ROC curves for the multiclass problem - All featues.

Through 5.12 is it possible to see 10 ROC curves related to each variable vs all others, hence, to express the behavior between one variable and all the others. Specifically, we note that for each ROC the value of AUC is always higher than 99%, then the classifier is able to perfectly distinguish between all the Positive and the Negative class points correctly. The one-vs-rest AUC score is 0.998 weighted by prevalence. High accuracy in the "all features" model was encountered, hence excellent prediction results were found.

In 5.3 for precision, "detect" gesture reach the lowest value among all classes with 91%, even if still remains an extremely positive value. Max precision is reached with the "forward" and "ok" gestures. With "up" gesture recall is the minimum, while "backward", "forward", "left", "ok" and "stop" gesture have the max recall score. In fact, for the latter the f1-score have the maximum value as well.

5.1.4 Choosing the best model

Following the analysis of the three different models, it has been found that each model reaches truly satisfying levels of predictions.

It is not possible to say that one model is actually better than the other because the percentage levels are almost the same. That said, it has to be confirmed that despite the high levels of correlation identified, the model that best captures the internal structure

	precision	recall	f1-score	support
backward	0.94	1.00	0.97	29
detect	0.91	0.94	0.92	31
down	0.96	0.96	0.96	26
forward	1.00	1.00	1.00	29
land	0.90	0.90	0.90	21
left	0.96	1.00	0.98	25
ok	1.00	1.00	1.00	23
right	0.95	0.91	0.93	23
stop	1.00	1.00	1.00	18
up	1.00	0.88	0.94	25
accuracy			0.96	250
macro avg	0.96	0.96	0.96	250
weighted avg	0.96	0.96	0.96	250

Table 5.3. Metrics - All features.

weighted avg	**precision**	**recall**	**f1-score**
Feature Selected	0.94	0.94	0.94
PCA	0.92	0.91	0.91
All features	0.96	0.96	0.96

Table 5.4. Metrics - Best Model.

is the one that makes use of all the variables. Despite this, the model with only 9 variables out of 42 manages to possess an average score of 94% against the 96% of the model with all the features. If the number of selected variables were increased, the value with all the features could have been easily reached. The same applies to the PCA, although in this case the percentage difference is more significant. On one side there is the 91% of the PCA model, on the other there is the 96% of the model with all the features. Here there is a 5% difference, which is not negligible at all. That being said, only 6 main components out of 42 were taken. In fact, increasing the number of selected main components from 6 to 20 has reached a value that was around 97%. In the end, the model with all the variables has been chosen, as it does not require to transform the input of the 42 variables in 20 every time.

5.2 3D Trajectory Evaluation

After fitting data, it is important to evaluate the goodness of fit. A visual examination of the fitted curve should be the first step. Beyond that, plenty of methods to assess

goodness of fit for both linear and nonlinear parametric fits are provided in literature.

5.2.1 Fitting Trajectory

In 5.13 nine different frames are visible, where the first eight are plots of the trajectory captured in live (in different times) and the last one represents the transformation of the 3D trajectory $G()$ in $G_{smooth}()$. Each plot consisting of three axis where the depth is described by the y axis, x describes the camera width and z the camera height. Each point is the position in the space, while the vector on it describes the orientation in that instant. The orientation vector is perpendicular to the palm of the hand. In addition, each point is assigned the mean speed value of the journey between the points at the interval t and $t-1$. The more the color tends to yellow, the faster the speed at that point is. Otherwise, the more the color tends to violet, the slower you are in that instant of time.

5.2.2 Fitting Evaluation

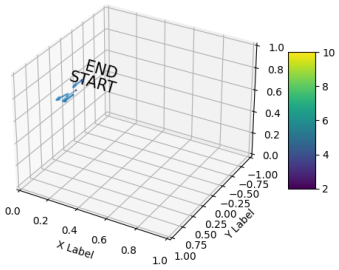
As is common in statistical literature, the term goodness of fit might be a model that the data could reasonably have come from, in which the model coefficients can be estimated with little uncertainty, or that explains a high proportion of the variability in the data, and is able to predict new observations with high certainty.

A particular application might still dictate other aspects of model fitting that are important for achieving a good fit, such as a simple model that is easy to interpret. It is possible to group the methods of analysis [**?**] into two types: graphical and numerical. Plotting residuals and prediction bounds are graphical methods that aid visual interpretation, while computing goodness-of-fit statistics and coefficient confidence bounds yield numerical measures that aid statistical reasoning.

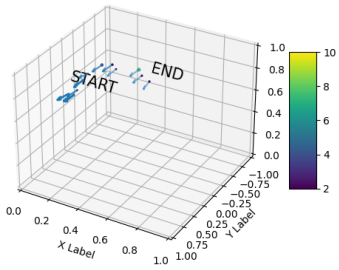
Generally speaking, graphical measures are more beneficial than numerical measures because they allow users to view the entire data set at once, and they can easily display a wide range of relationships between the model and the data. The numerical measures are more narrowly focused on a particular aspect of the data and often try to compress that information into a single number.

Metrics used to evaluate regression models should be able to work on a set of continuous values (with infinite cardinality), and therefore they are slightly different from classification metrics. There are many different evaluation metrics out there but only some of them are suitable to be used for regression. After using graphical methods to evaluate the goodness of fit, the followings were evaluated:

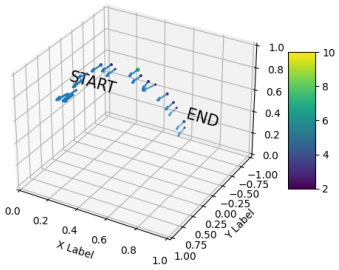
- `R squared`



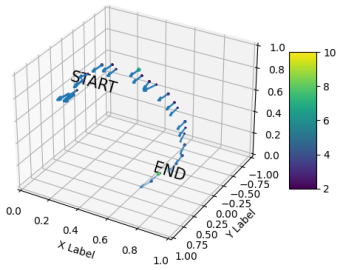
(a)



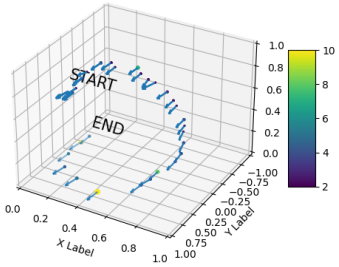
(b)



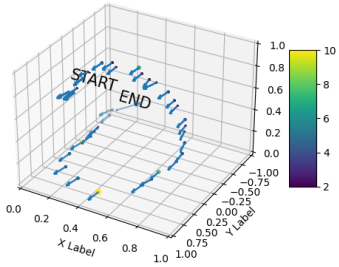
(c)



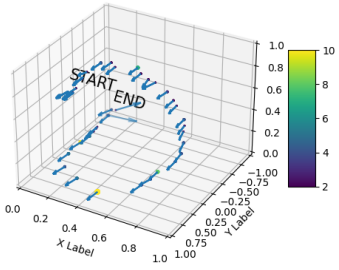
(d)



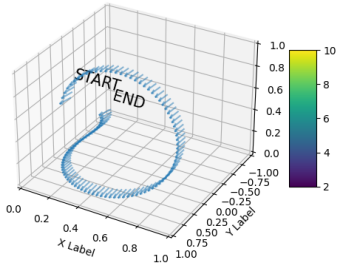
(e)



(f)



(g)



(h)

Figure 5.13. Complete acquisition of a 3d trajectory in all its steps.

This statistic measures how successful the fit is in explaining the variation of the data. Putting in another way, R squared is the square of the correlation between the response values and the predicted response values. It is also called the square of the multiple correlation coefficient and the coefficient of multiple determination. R squared is defined as the ratio of the sum of squares of the Regression (SSR) and the Sum of Squares Total (SST). R squared is expressed as

$$R^2 = \frac{SSR}{SST} = \\ = \frac{\sum_{i=1}^{n} w_i(\widehat{y_i} - \overline{y})^2 \cdot}{\sum_{i=1}^{n} w_i(y_i - \overline{y})^2} \tag{5.1}$$

Equation 5.1. R squared

R squared can take on any value between 0 and 1, with a value closer to 1 indicating that a greater proportion of variance is accounted for by the model.

R squared is a good measure to determine how well the model fits the dependent variables. However, it does not take into consideration the overfitting problem. If the regression model has many independent variables, it is because the model is too complicated. That is why Adjusted R Square is introduced because it will penalize additional independent variables added to the model and adjust the metric to prevent overfitting issues. It is possible to get a negative R squared for equations that do not contain a constant term. Because R squared is defined as the proportion of variance explained by the fit, if the fit is actually worse than just fitting a horizontal line then R squared is negative. In this case, R squared cannot be interpreted as the square of a correlation. Such situations indicate that a constant term should be added to the model;

- `Adjusted R squared`
  This statistic uses the R squared statistic defined above, and adjusts it based on the residual degrees of freedom. The residual degrees of freedom is defined as the number of response values n minus the number of fitted coefficients m estimated from the response values.

$$v = n - m. \tag{5.2}$$

Equation 5.2. Degrees of Freedom Adjusted R squared.

v indicates the number of independent pieces of information involving the n data points that are required to calculate the sum of squares. Note that if parameters are bounded and one or more of the estimates are at their bounds, then those estimates are regarded as fixed. The degrees of freedom is increased by the number of such parameters.

The adjusted R squared statistic is generally the best indicator of the fit quality when you compare two models that are nested — that is, a series of models each of which adds additional coefficients to the previous model.

$$R^2_{adjusted} = 1 - \frac{(n-1)\sum_{i=1}^{n} w_i(y_i - \widehat{y})^2}{v\sum_{i=1}^{n} w_i(y_i - \overline{y})^2} \quad . \tag{5.3}$$

Equation 5.3. R squared.

The adjusted R squared statistic can take on any value less than or equal to 1, with a value closer to 1 indicating a better fit.

- `Root mean squared error.`
  This statistic is also known as the fit standard error and the standard error of the regression. It is an estimate of the standard deviation of the random component in the data, and is defined as

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} w_i(y_i - \widehat{y})^2}{v}}. \tag{5.4}$$

Equation 5.4. RMSE.

A value closer to 0 indicates a fit that is more useful for prediction. It is used more commonly than MSE because firstly sometimes MSE value can be too big to compare easily. Secondly, MSE is calculated by the square of error, and thus square root brings it back to the same level of prediction error and makes it easier for interpretation.

With reference to the trajectory 5.13, 3 ridge regressions for x, y and z component were calculated on it, as explained in 4.2.4, increasing the degree of the fitting polynomial from 2 to 9. In 5.14 it is possible to envision how Root Mean Squared Error (RMSE) behaves as the degree of the fitting polynomial increases. In the specific, it is evident

that RMSE decreases dramatically when we take a grade greater than four, and is true for all three components. The value definitely stabilizes when the degree is equal to seven. In fact, a value of less than 5% is reached.

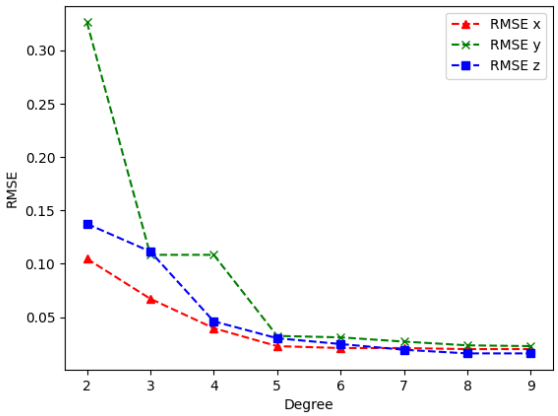


Figure 5.14. RMSE plot.

In 5.15 the R squared plot is present. Here, unlike the previous chart, the closer the value is to 1, the better the result. A value of 0.91 means that the fit explains 91% of the total variation in the data about the average. If a grade higher than four is taken, then excellent values capable of capturing more than 90% of variance are obtained.

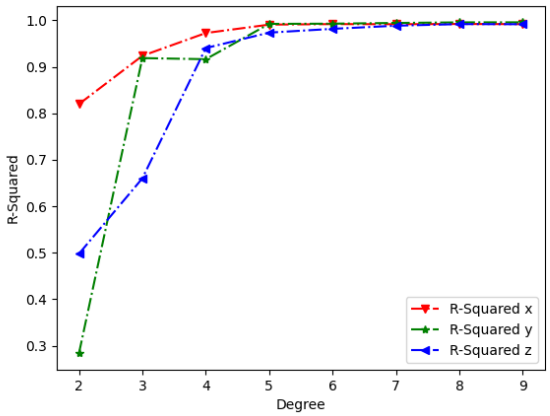


Figure 5.15. R-Squared plot.

In 5.16 the Adjusted R-squared plot is shown. Since R^2 always increases as more predictors are added to the model, adjusted R^2 can serve as a metric that tells how useful a model is, adjusted for the number of predictors in a model. The graph is very sim-

ilar to the previous one (see Fig. 5.14), but it's slightly different. Since the Adjusted R-squared has high values despite the growth of the degree, it explains that does not occur overfitting, then it can perfectly explain the trajectory.

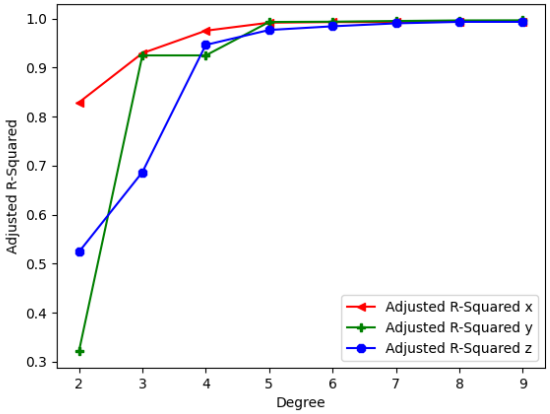


Figure 5.16. Adjusted R-Squared plot.

Following the study by RMSE, R-squared and Adjusted R-squared it has come to the conclusion that a degree of 7 is more than enough to describe a trajectory as necessary.

5.3 Simulation Evaluation

To have a clearer view of the trajectory, a sphere SDF model has been created . An SDF model refers to the <model> SDF tag, and it is essentially a collection of links, joints, collision objects, visuals, and plugins.

```
<?xml version="1.0" ?>
<sdf version="1.5">
    <model name="my1stmodel">
    <static>true</static>
    <link name="link">
        <gravity>0</gravity>
        <visual name="sphere">
            <geometry>
                <sphere>
                    <radius>0.02</radius>
                </sphere>
            </geometry>
            <material> <!-- Wheel material -->
```

```
                <ambient>0.1 1 0.1 1</ambient>
                <diffuse>1 1 1 1</diffuse>
                <specular>0 0 0 0</specular>
                <emissive>0 0 0 1</emissive>
            </material> <!-- End wheel material -->
        </visual>
    </link>
    </model>
</sdf>
```

Listing 5.1. Sphere SDF model.

To prevent the sphere from colliding with the drone, the static tag has been set to "true". To prevent the sphere from falling, the gravity tag has been set to 0.

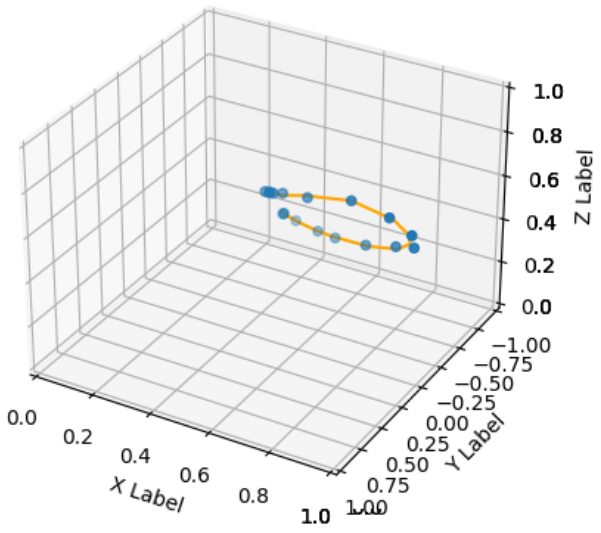


Figure 5.17. Trajectory in Gazebo plot.

During the simulation of the trajectory, each tot seconds a sphere is spawned to indicate the position of the drone in space at that precise moment. In 5.17 the acquired trajectory and in 5.18 execution of the trajectory on a drone that is coming to an end are reported.

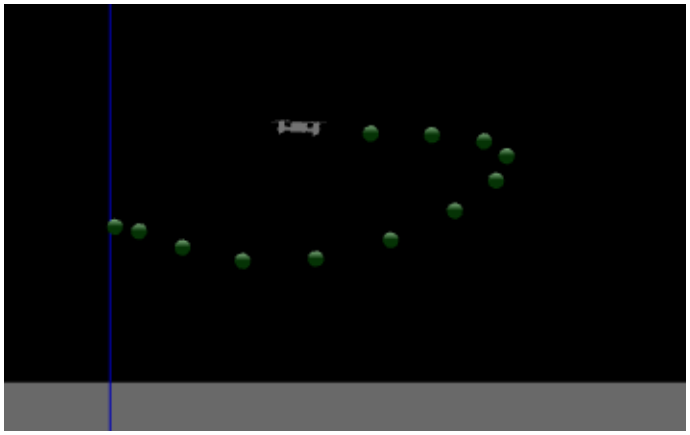


Figure 5.18. Trajectory example in Gazebo.

The intersection between the blue vertical axis and the green ball represents the point of origin of the trajectory.

5.4 Real World Evaluation

The human-drone interaction takes place by means of commands. These are gestures that after processing are sent as a packet to the drone. The connection is UDP. In the figure 5.19, it is possible to see two consecutive moments, in which the drone moves from left to right.

(a) Screenshot at the instant t_1. **(b)** Screenshot at the instant t_2.

Figure 5.19. Human-drone interaction with the "right" gesture. The larger image is captured directly from the front PC camera, while the bottom-right image represents the captured video of the drone, at the same time.

In figure 5.20, it is possible to see two consecutive moments, in which the drone moves from right to left. Note that, in images 5.20a and 5.20b it actually seems that the drone

is moving from left to right, but this is simply due to the fact that the video from the PC camera to the drone are captured, hence it is mirrored.

(a) Screenshot at the instant t_1.

(b) Screenshot at the instant t_2.

Figure 5.20. Human-drone interaction with the "left" gesture. The larger image is captured directly from the back PC camera, while the bottom-right image represents the captured video of the drone, at the same time.

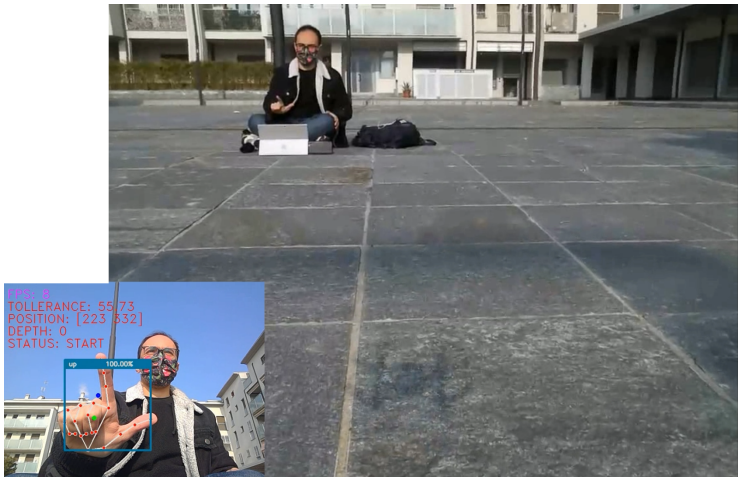
In figure 5.21, it is possible to see two consecutive moments in which the drone moves from top to bottom.

(a) Screenshot at the instant t_1.

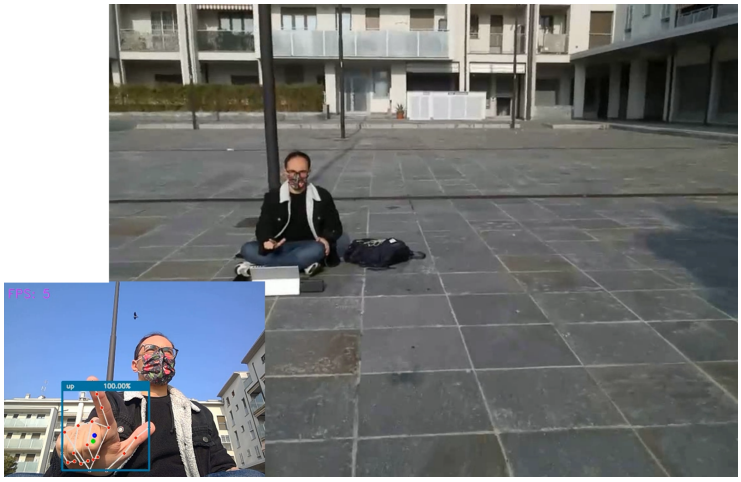
(b) Screenshot at the instant t_2.

Figure 5.21. Human-drone interaction with the "down" gesture.

In figure 5.22, it is possible to see two consecutive moments in which the drone moves from bottom to top.



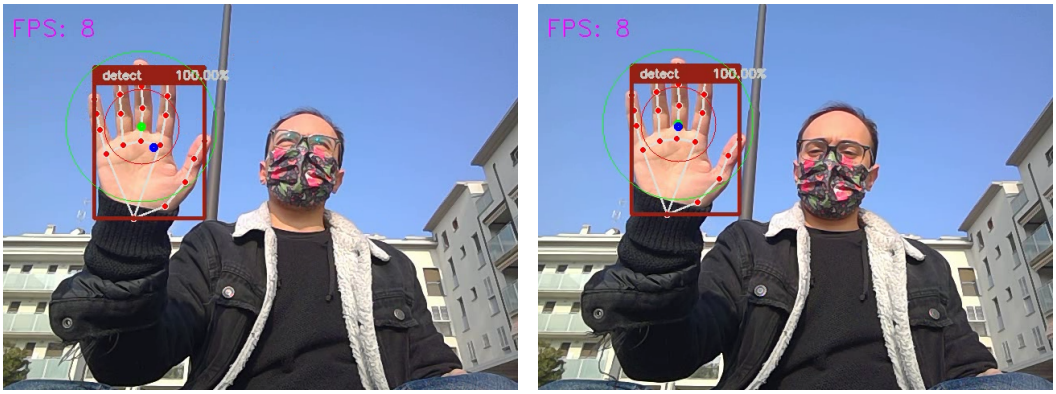
(a) Screenshot at the instant t_1.



(b) Screenshot at the instant t_2.

Figure 5.22. Human-drone interaction with the "up" gesture. The larger image is captured directly from the drone camera, while the bottom-left image represents the captured video of the front PC camera at the same time.

When using the "detect" gesture, the middle point p at each frame is saved in a circular tail composed of n elements (in the figures 5.23a and 5.23b is the green dot, in the center of the hand). In the image 5.23, the blue dot represents the midpoint b of the n points added to the tail. If it coincides with the green point p it means that the hand has been stationary in space for a short time, otherwise it means that it has moved. There are two circles: one red of radius r_1 and one green of radius r_2, where $r2 > r1$. When the current midpoint p is inside the red circle, the points p in the tail are saved. When the blue point b is outside the red circle, the last 5 points are saved in such a way that they represent the first points of the trajectory $G()$.



(a) Screenshot at the instant t_1.

(b) Screenshot at the instant t_2.

Figure 5.23. Middle point p and average of middle points b of the hand. Images captured from the front PC camera.

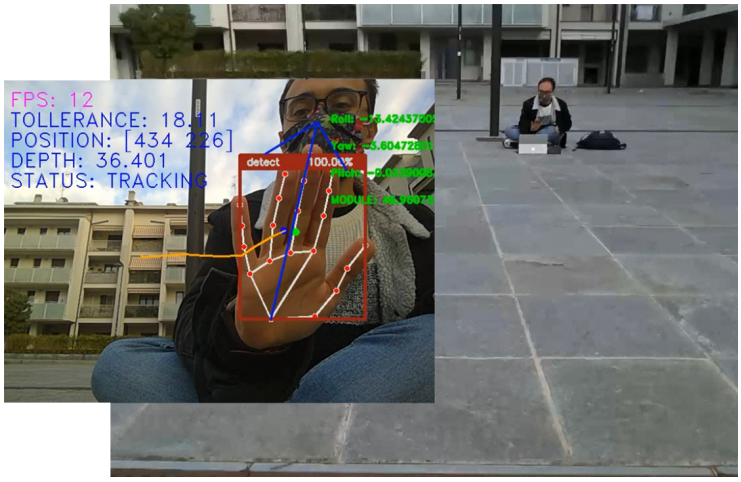
In the image 5.24, the 3D trajectory becomes active. There are only 3 instants of the acquisition. The trajectory starts with the "detect" gesture. Afterwards, it is processed by moving the hand in space. At the end it is necessary to close the trajectory with the "ok" gesture. Specifically, the trajectory acquired has the objective of allowing the drone to make a slight bending forward, to the right.

After acquiring the 3D trajectory, the latter is launched on the drone. In the image 5.25, it is observable how the camera follows the trajectory of 5.24 in 3 different moments.

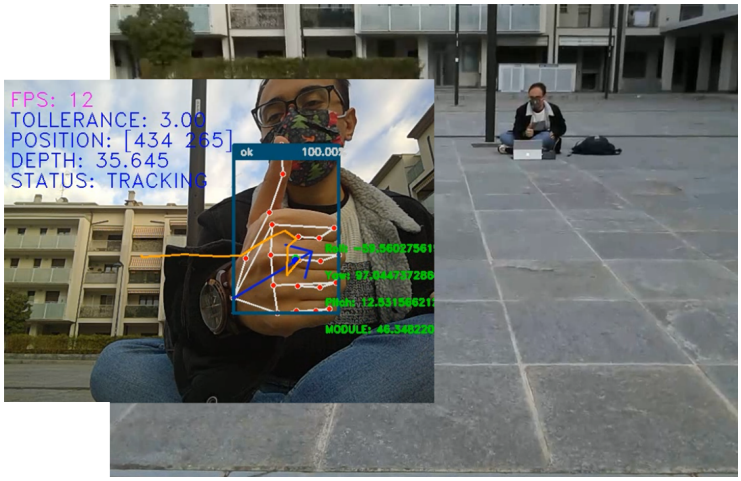
In 5.26, there is the 3D trajectory w.r.t. XY axis and XZ axis in the first two images at the top, whose values are in the range from -1 to $+1$. In the XY axis, the x axis the width of the camera and in y the depth are found. Moving to the right means that the drone will move from left to the right, similar to the left. When at the y-axis, the more the value is 1 the more the drone will go forward. Instead, if y assumes values close to -1 then the drone goes backwards. When the drone is at $y = 0$, it means that the drone does not go forward and backward w.r.t. the starting position. Each trajectory always starts from the origin. In this case that the trajectory expands from left to right and continues going forward all the way is shown. In the XZ axis, the z axis, the camera height, is found. When at the z axis, the more the value is 1 the more the drone will rise higher. Instead, if z assumes values close to -1 then the drone will target the floor. When the drone is at $z = 0$ it means that the drone does not go up and does not go down w.r.t. the starting position. In this case the trajectory expands from left to right and rises slightly towards the end is shown.

In 5.26, there is always the same trajectory, in the last two images at the bottom, but scaled w.r.t. a fixed value. This will tell how much the trajectory approximately will

(a) Screenshot at the instant t_1.



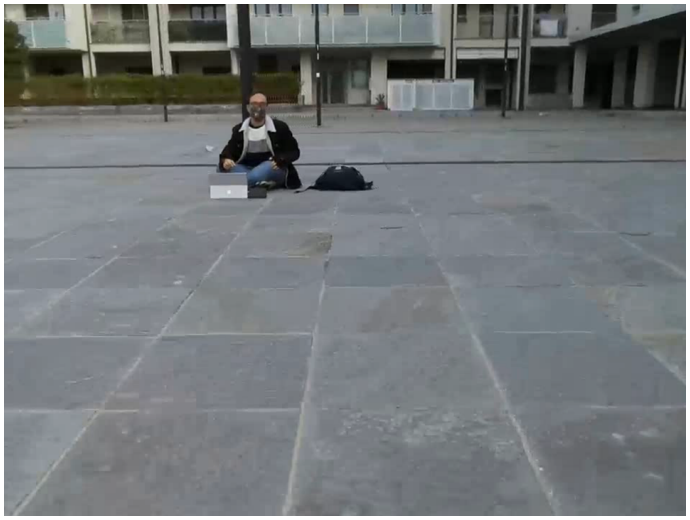
(b) Screenshot at the instant t_2.



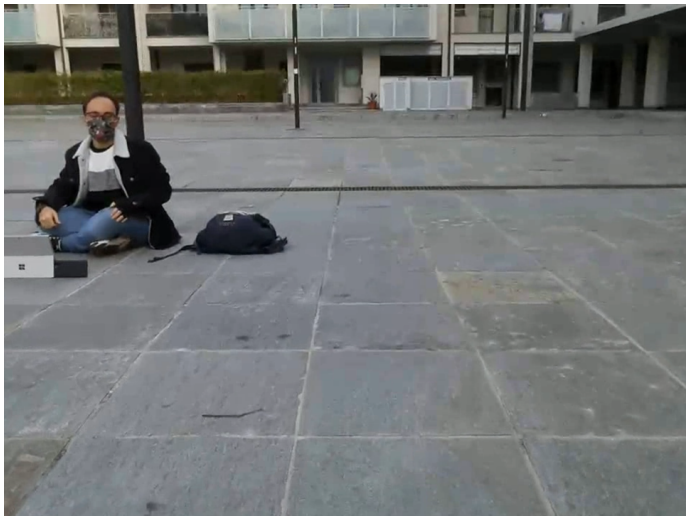
(c) Screenshot at the instant t_3.

Figure 5.24. Capture of a 3D trajectory in real. In the images on the left the front camera coming from the PC and pointing the hand of the user, in the images on the right is the camera of the drone pointing the scene.

(a) Screenshot at the instant t_1.



(b) Screenshot at the instant t_2.



(c) Screenshot at the instant t_3.

Figure 5.25. Execution of the acquired 3D trajectory.

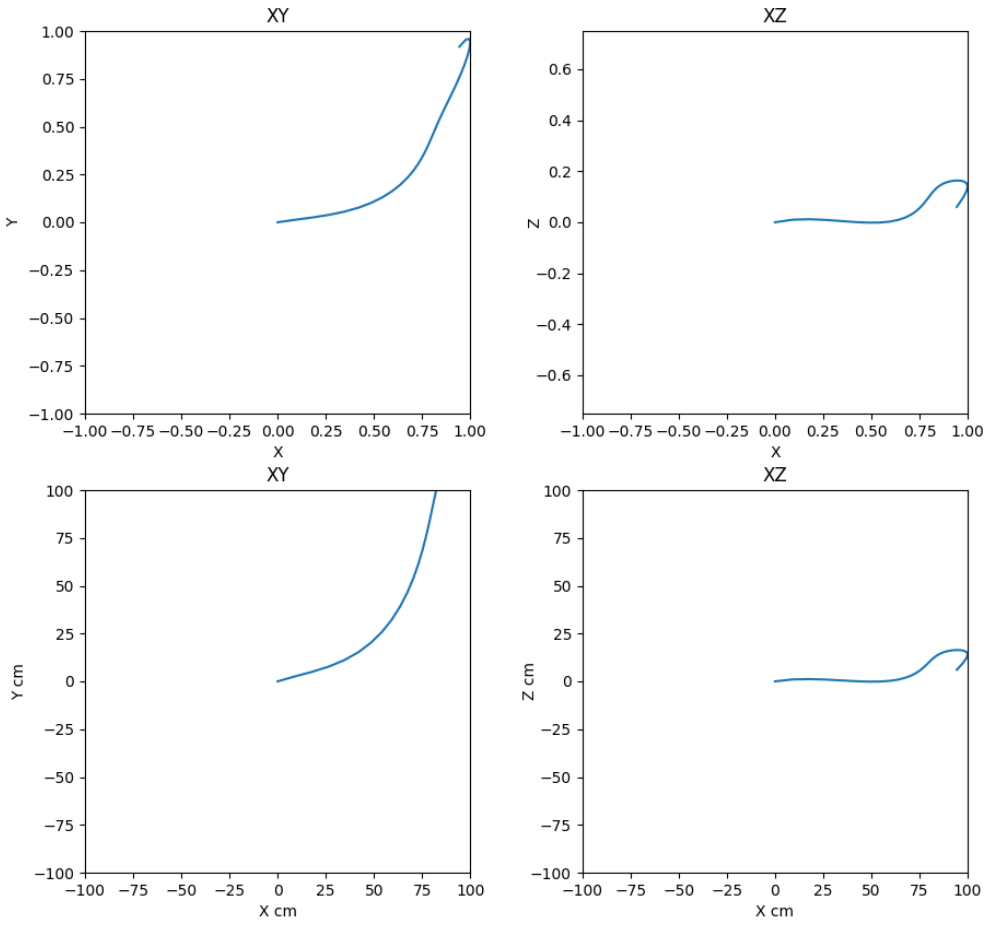


Figure 5.26. Trajectory example in Gazebo.

move in the surrounding space. In the image the set value is 100, so that the trajectory moves in a range of 100cm.

The trajectory veracity is visible to the naked eye. For the project purpose, the perfection of the trajectory execution was not a requirement, but rather that it reflects the correctness of the fixed dimension of the trajectory. This point is important as if the range of action in which the drone operates is uncontrolled then it could be dangerous for those around it. It was soon noticed that the drone does not perform trajectories in a range of about 1m, but these appear to be smaller in size. In fact, following several tests, it was observed that by setting a travel time of 3 seconds, a 0.25s signal sending interval and a drone speed of 15cm/s, the total distance traveled was not correct. Taking the average over all the test attempts it was concluded that 15cm/s actually

corresponded to about $11.7cm/s$. Then, a map was built that gave an idea of the space traveled along the XY and XZ axes.

The acquisition can also be captured directly from the drone camera. It is important to keep in mind that this process can be poor in very strong wind conditions, because the built algorithm does not have ways to stabilize the image dynamically. This is especially true when the trajectory is launched on a lightweight drone like the DJI Ryze Tello.

Figure 5.27. Human-drone interaction directly from the drone camera.

Conclusion and Perspectives

This chapter presents first, in 5.5, concluding thoughts and finally suggestions for possible future research lines in 5.6.

5.5 Concluding Thoughts

Our work makes a further step in the direction of Motion Tracking approaches, and aims to find feasible solutions in a practical scenario, that of drone filming. The hand gesture 3D tracking technology provides a detailed view of digitizing the measurement data, positional movement (tracked on the X, Y and Z axes of a 3D coordinate system) and orientation data calculated through Rotation (roll, pitch and yaw). This movement is reported in relation to the "detect" gesture.

3D tracking technology bridges the gap between static images and dynamic movements and brings the physical world into digital interfaces (and vice-versa). This system is tied to its accuracy. In fact, the difference between arriving exactly at the destination or being off by miles (or millimetres in surgical navigation applications) is a requirement to keep absolutely in mind. Our system tries to establish itself not in an environment of absolute precision, but instead it tries to capture the user's idea by helping him, for the success of the shooting. In this meaning, the original goal is reached with success.

The methods applied have been suitable and have led to the construction of a useful and working system. The main difficulties were those related to the estimation of orientation to obtain information on three-dimensionality, because it was necessary to reason on unconventional approaches.

The Tello is a mini drone that, despite containing several advanced features is mainly dedicated to novice users. Weighing only $80gr$, it does not remain stable in case of a very wind day. During the test phase, it was verified that trying to capture a trajectory directly from the drone's camera in a day with a wind travelling over $10km/h$ is unfeasible. The drone is constantly pushed by the wind, making it unstable and generating too much noise on the trajectory not allowing the correct reading. Otherwise,

with a wind slower than $5km/h$ the tests were carried out without any problem. This leads to the conclusion that it would be ideal to use a heavier drone, or at least a more wind resistant one, so that the latter can generate not too much noise to overpower the acquisition process of the 3D trajectory.

5.6 Future Works

Our contribution gives space to numerous developments: it might be interesting to think of a gestures combinations system, thus establishing a real language. Obviously, this would make user-drone interaction more complex. Even though benefits are more interesting types of applications as for example combinations of parameterizable programmed actions. In fact, if two different gestures with the same hand (at different times) are done, then two actions are performed whose intensity could be determined by the speed of passage from one gesture to another or by the direction in which this passage is made.

Since the orientation estimation is performed, it can be also exploited in such a way that this information is effectively used to make shots. This would permit the drone not only to respect the position in space, but also the orientation. In the case of Tello, is it possible handle only the yaw. However, there are plenty robots (like mechanical arms) capable to control also the row and pitch. Mediapipe gives a way to locate more than one hand in the scene. This means that it may be possible to combine not only different gestures with the same hand as time goes further, but also multiple hands with different gestures in the same instant or as time goes further.

Appendix A

List of Acronyms