



UNIVERSITÀ DEGLI STUDI DI MILANO - BICOCCA

Scuola di Scienze

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di Laurea Magistrale in Informatica

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

Relatore: Dr. Alessandro Giusti

Co-relatore: Dr. Loris Roveda

Co-relatore: Dr. Gianluigi Ciocca

Tesi di Laurea Magistrale di:
Umberto Cocca
Matricola 807191

Anno Accademico 2020-2021

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

Umberto Cocca
Lugano, 23 February 2022

-dedica-

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

Abstract

Robotic systems are increasingly being adopted for filming and photography purposes. By exploiting robots, in fact, it is possible to program complex motions, 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, we capture 3D movements, using a hand tracking system, after building a hand gesture recognition and setting up a solid pipeline to detect 3D trajectories obtained from 2D pixel landmarks, hence estimating the orientation and distance from the camera. Trajectories are interpolated and smoothed with ridge regression. As a proof of concept, the captured trajectory was launched first on a drone, in simulation, implemented in the ROS framework, and later on a real drone called DJI Ryze Tello. The whole pipeline can be easily translated into another kind of task, e.g. interaction and communication in AR / VR.

Acknowledgements

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

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

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

Contents

List of Figures	ix
List of Tables	x
List of Equations	xi
List of Listings	xii
Introduction	1
1 Literature review	4
2 Background	7
2.1 Regression	7
2.1.1 Linear Models for Regression	7
2.1.2 Normal equations	8
2.2 The Problem Of Overfitting	11
2.2.1 Regularization	11
2.2.2 Cost Function	12
2.2.3 (Batch) Gradient Descent	13
2.3 Artificial Neural Networks	15
2.3.1 Feedforward Fully-Connected Neural Networks	16
2.3.2 Neural Network (NN) Setup for Classification	20
2.3.3 Optimisation algorithms	22
3 Tools	24
3.1 Tello	24
3.1.1 Tello Command Types and Results	25
3.2 Gazebo	26
3.3 Frameworks	26
4 Methodologies	28
4.1 Hand Gesture Recognition	28
4.1.1 Data Acquisition and Description	29

4.1.2 Pearson Correlation	30
4.1.3 PCA	33
4.1.4 Model	36
4.2 3D trajectory detection	37
4.2.1 Orientation estimation	38
4.2.2 Camera-hand distance estimation	40
4.2.3 Errors	43
4.2.4 Smoothing	43
4.3 Drone controller	48
4.3.1 Simulation	48
4.3.2 Real world	50
4.4 Pipeline	51
5 Evaluation	52
5.1 Hand gesture recognition model evaluation	52
5.2 3D trajectory evaluation	58
5.2.1 Fitting trajectory	58
5.2.2 Fitting evaluation	59
5.3 Drone in action	64
5.4 Real	64
Conclusion and perspectives	65
A List of Acronyms	66
Bibliography	68

Figures

1.1	Palm detector model architecture.	5
1.2	Architecture of hand landmark model.	6
1.3	Hand landmark model performance characteristics.	6
2.1	Overfitting and Underfitting.	11
2.2	Gradient descent.	13
2.3	Image of a human neuron.	16
2.4	Feed forward neural network.	17
2.5	Image of a human neuron.	19
2.6	Leaky Rectified linear activation.	20
2.7	Comparison of Adam to other optimization algorithms.	23
3.1	Tello - Aircraft diagram.	24
4.1	Hand Landmarks.	28
4.2	Full list of gestures.	29
4.3	Heatmap.	31
4.4	Feature Selection: Feature Vs Feature.	32
4.5	Principal Component Analysis (PCA) - Biplot.	34
4.6	Scree plot.	36
4.7	Hand gesture recognition NN.	37
4.8	Aircraft principal axes.	38
4.9	Orientation test.	39
4.10	Runge's phenomenon problem.	44
4.11	Smoothing spline 1D.	45
4.12	Ridge regression 1D.	46
4.13	Ridge coefficients as a function of the regularization.	47
4.14	Multiple Ridge regression.	48
5.1	Confusion matrix - Features selected.	53
5.2	Confusion matrix - PCA.	55
5.3	Confusion matrix - All features.	57
5.4	Complete acquisition of a 3d trajectory in all its steps.	60
5.5	RMSE.	63

5.6	R-Squared.	63
5.7	Adjusted R-Squared.	64

Tables

3.1 Tello Python Commands.	25
5.1 Metrics - Feature Selected.	54
5.2 Metrics - PCA.	56
5.3 Metrics - All features.	58

Equations

2.1	Linear combinations of fixed nonlinear basis functions.	8
2.2	Gaussian basis function.	8
2.3	Sum of squares regression.	9
2.4	Sum of squares regression in matrix notation.	9
2.5	Design matrix.	9
2.6	Weights of $M - 1$ features and target of N examples.	10
2.7	Optimization problem for ridge regression.	10
2.8	Normal equations.	10
2.9	Prediction in ridge regression.	10
2.10	Cost function Regularisation term.	12
2.11	Loss Function for Ridge Regression	12
2.12	Ridge regression solution	13
2.13	Mean Squared Error.	14
2.14	Regularisation term.	14
2.15	Gradient Descent.	14
2.16	Gradient Descent in compact form.	15
2.17	Cost function for Gradient Descent.	15
2.18	Forward propagation.	18
2.19	Leaky Rectified linear activation.	19
2.20	Softmax nonlinearity.	21
2.21	Cross-entropy loss.	21
2.22	Softmax cross entropy.	22
4.1	Orientation test.	39
4.2	Z-Rotation Matrix in homogeneous coordinates.	41
4.3	Y-Rotation Matrix in homogeneous coordinates.	41
4.4	X-Rotation Matrix in homogeneous coordinates.	42
4.5	General Rotation.	42
5.1	R-square.	61
5.2	Degrees of Freedom Adjusted R-Square.	61
5.3	R-square.	62
5.4	RMSE.	62

Listings

Introduction

Robots are increasingly taking over from what has traditionally been the preserve of human creative people. Progressively film creators are turning to automated systems for recording moving images. Producing new, targeted, and engaging content can be a stressful, time-consuming endeavor. That 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 allows teams to shoot a variety of angles that would be physically impossible or cost a fortune in labor.

Unfortunately, it is known that robot programming could be a very time-consuming 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). To speed up the process of robot programming different company as KUKA, partner 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 [Ani]: 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. Another company, SISU Cinema Robotics, guide the robot with the motion of the hand using a joystick [SIS]

Drone filming and aerial photography presents a wide range of benefits and opportunities to businesses and brands looking to capture stunning aerial imagery and to produce incredible new perspectives [Top]. There can be quite a steep initial learning curve when it comes to getting to grips with drone technology. Furthermore, it is important to use equipment that is both easy to operate and will aid the goals to capture high-quality content to include in the videos.

As well as becoming a decent drone pilot, to make sure that your equipment is correctly set up to ensure that the footage is correctly exposed is needed. As it is important to understand how to fly smoothly and consistently to 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 is extremely crucial look for a

specific angle that seems unique, cinematic, and suggests 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, potentially the one who shoots could 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 [Wha]. Do not moderate the flight time and do not keep a close eye on the battery levels, can lead to disastrous accidents [Lis].

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 [Oikonomidis et al., 2011] 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 [Ocu] marking another important Virtual Reality (VR) input milestone, in the evolution of VR input, with the announcement of hand tracking on Oculus Quest enabling natural interaction in using the own hands on an all-in-one standalone device, no extra hardware required. 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 [Zhang et al., 2020] 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 has been defined. However, this trajectory g is disturbed by estimation errors. Therefore, two different data fitting algorithms were applied on the trajectory g : first 3 splines and then 3 ridge regression, in both cases for each components x, y and z . This phase allowed to go from trajectory g to g_{smooth} .

As a Proof of Concept (POC), the captured trajectory was launched first on a drone,

in simulation, implemented in the Robot Operating System (ROS) framework. After, 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 is 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 is 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 [Lugaresi et al., 2019] 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 [Zhang et al., 2020]. 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.

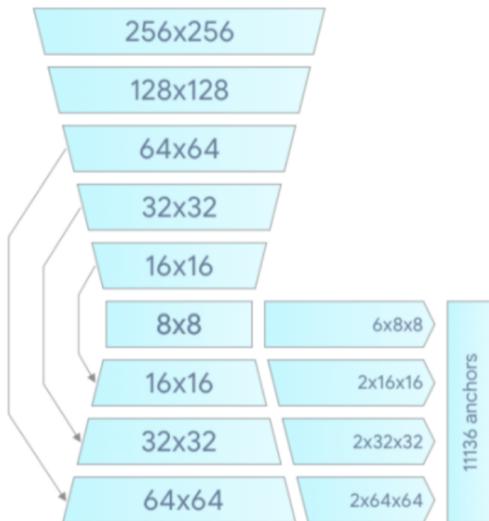


Figure 1.1. Palm detector model architecture.

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.

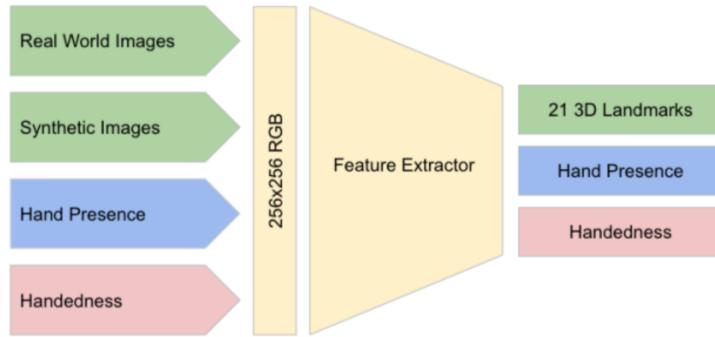


Figure 1.2. Architecture of hand landmark model.

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. [Bishop, 2006]

Given a training data set comprising N observations x_n , where $n = 1, \dots, N$, 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:

$$y(x, w) = w_0 + w_1 x_1 + \dots + w_D x_D \quad (2.1)$$

where $x = (x_1, \dots, x_D)^T$. 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:

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

where $\phi_j(x)$ are known as basis functions. The total number of parameters in this model will be M .

The parameter w_0 is called bias parameter. It is often convenient to define an additional dummy "basis function" $\phi_0(x) = 1$, so that:

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

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

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

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:

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

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

where u_j regulates the locations of the basis functions in input space, while the parameter s is their spatial scale. 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 [Valchanov, 2018]), so that the following is minimised:

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

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

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

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:

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

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

where ϕ is an $N \times M$ matrix, so that:

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

Equation 2.5. This is called the design matrix whose 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 $E(w)$:

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

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

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \frac{1}{2} (\phi \mathbf{w} - \mathbf{t})^T (\phi \mathbf{w} - \mathbf{t}) \quad (2.9)$$

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

The gradient is computed and solving for \mathbf{w} the result obtained is:

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

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

This is the real minimum because if the second derivative $\nabla^2 E(\mathbf{w}) = \phi^T \phi$ 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 \mathbf{w}^* is found and a new data point that has never been seen during training is received, the new target t^* is predicted as:

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

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

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)).

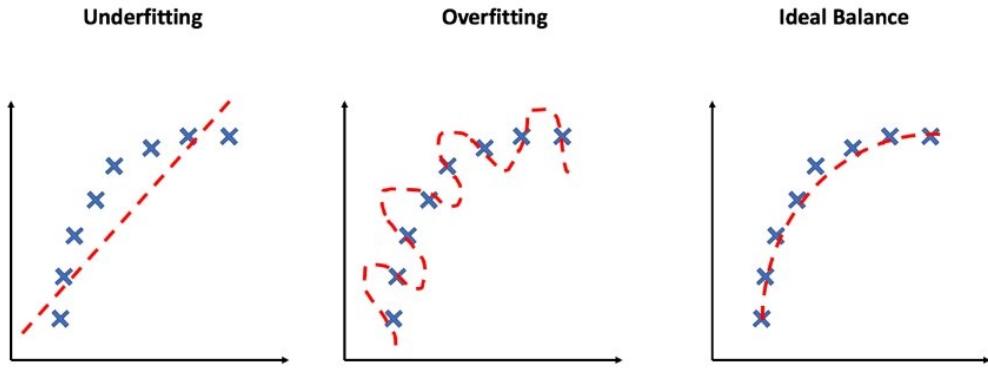


Figure 2.1. Image by "Applied Supervised Learning with R" book ^a. It is possible to see high bias resulting in an oversimplified model (that is underfitting); high variance resulting in over-complicated models (that is overfitting); and lastly, striking the right balance between bias and variance.

^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 overfitting, 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 error function to minimized takes the form:

$$E(w) = E_D(w) + \lambda E_W(w) \quad (2.12)$$

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 is the error based on dataset, while E_W is based on weights.

2.2.2 Cost Function

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

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

Equation 2.10. Cost function | Regularisation term.

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

This is also called ridge regression, a method of estimating the coefficients of multiple-regression models in scenarios where independent variables are highly correlated. If the sum-of-squares error function given by the following formula is considered:

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

then the total error function becomes:

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

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

Setting the gradient of $E(w)$ w.r.t. w to zero, and solving for w , the result obtained is:

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

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

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

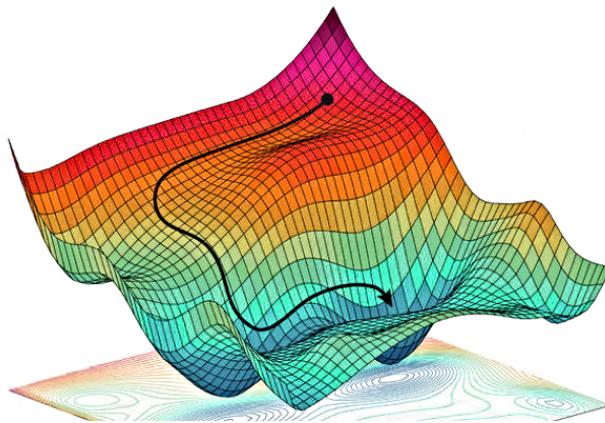


Figure 2.2. Gradient descent is based on the observation that if the multi-variable function $E(w)$ is defined and differentiable in a neighborhood of a point w_0 , then w_0 decreases fastest if one goes from w_0 in the direction of the negative gradient of $E(w)$ at w_0 , $-\nabla E(w_0)$.

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.

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

Equation 2.13. Mean Squared Error. $1/2$ is added, as in SSE, so the derivative doesn't need a constant out front. 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:

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

Equation 2.14. Regularisation term.

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

Combining together following (2.12)

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

Equation 2.15. Gradient Descent.

Have some function $E(w)$

Want $\arg \min_w E(w)$)

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

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

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

where in compact form is:

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

Equation 2.17. Cost function for Gradient Descent.

2.3 Artificial Neural Networks

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

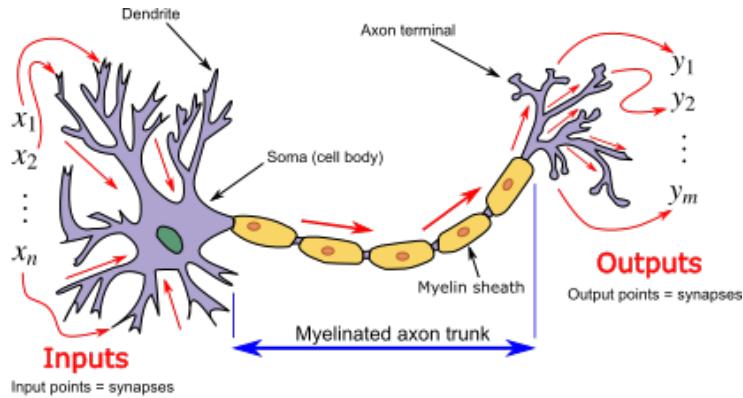


Figure 2.3. Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals. The signal represents a short electrical pulse called 'spike'.

Neurons have 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.

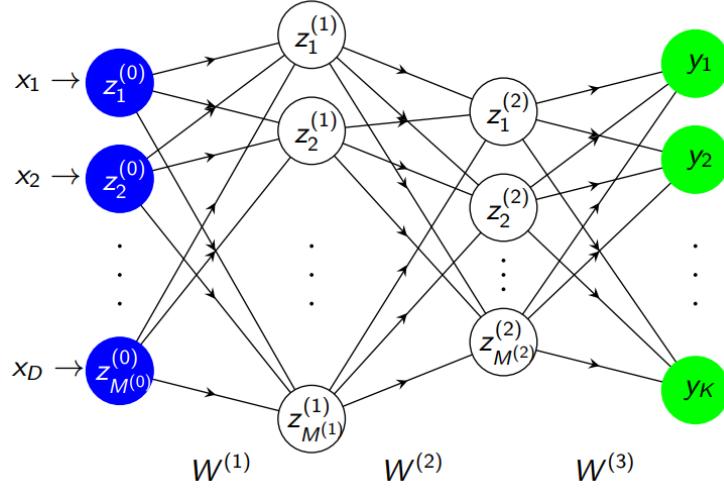


Figure 2.4. The image graphically shows a NN with three layers (or two hidden layers). The input layer has $M^{(0)}$ neurons and the output layer has K outputs. Neurons are organized in layers to allow parallel computation to avoid cyclic dependencies. The process of computing NN outputs from inputs 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 $1, \dots, L$ and usually the starting input is considered the 0th layer. Inputs x_1, \dots, 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 $l \in \{1, \dots, L\}$ has $M^{(l)}$ neurons.

$M^{(1)}$ neurons perform a perceptron-like computation:

$$u_m^{(1)} = (w_m^{(1)})^T x + b_m^{(1)}, \quad z_m^{(1)} = f(u_m^{(1)}), \quad m = 1, \dots, M^{(1)} \quad (2.22)$$

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

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

as input of:

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

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

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

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

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

which can be written as a matrix multiplication:

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

Equation 2.18. 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 $w_{mn}^{(2)}, m = 1, \dots, M^{(1)}, n = 1, \dots, M^{(2)}$. 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)

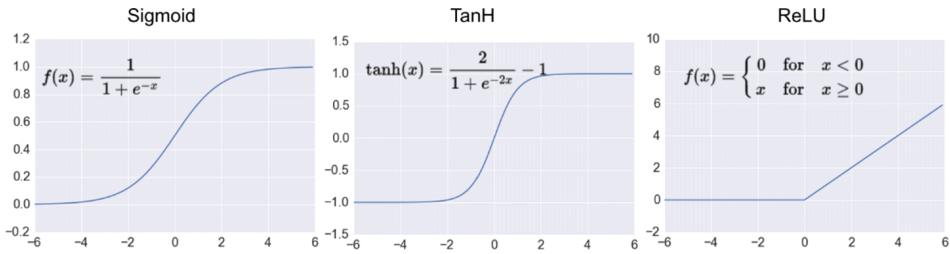


Figure 2.5. Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals. The signal represents a short electrical pulse called 'spike'.

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" [Bengio et al., 1994], in particular on the difficulty of training Recurrent Neural Network (RNN) [Pascanu et al., 2013], while the second is to accelerate the convergence speed. More sophisticated activation function as the "leaky ReLU" try to solve the dying ReLU problem [Lea]. 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) [Xu et al., 2015].

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

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

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

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

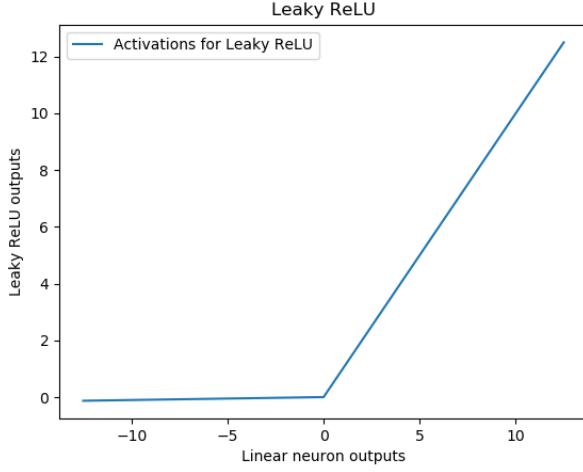


Figure 2.6. Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals. The signal represents a short electrical pulse called 'spike'.

2.3.2 NN Setup for Classification

For a classification task with K classes, 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:

$$c^* = \arg \max_k y_k \quad (2.29)$$

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

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

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:

$$S(y) = \left(\frac{e^{y_1}}{\sum_k e^{y_k}}, \dots, \frac{e^{y_K}}{\sum_k e^{y_k}} \right) \quad (2.31)$$

Equation 2.20. Softmax nonlinearity.

In contrast to other activation functions, it is applied to the full last layer of the NN, not to each independent 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:

$$t = (0, \dots, 0, 1, 0, \dots, 0) \quad (2.32)$$

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:

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

Equation 2.21. Cross-entropy loss.

The intuition about the cross-entropy corresponds to the number of additional bits needed to encode the correct output, given that 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:

$$E_{CE+SM} = - \sum_k (t_k \log S_k(y)) \quad (2.34)$$

Equation 2.22. 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 Optimisation algorithms

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 [Hager and Zhang, 2006], BFGS, the improved version to decrease memory usage L-BFGS [Saputro and Widyaningsih, 2017], 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 [Kingma and Ba, 2017]. It is an extension to stochastic gradient descent that has recently seen broader adoption for Deep Learning (DL) applications in computer vision and Natural Language Processing (NLP).

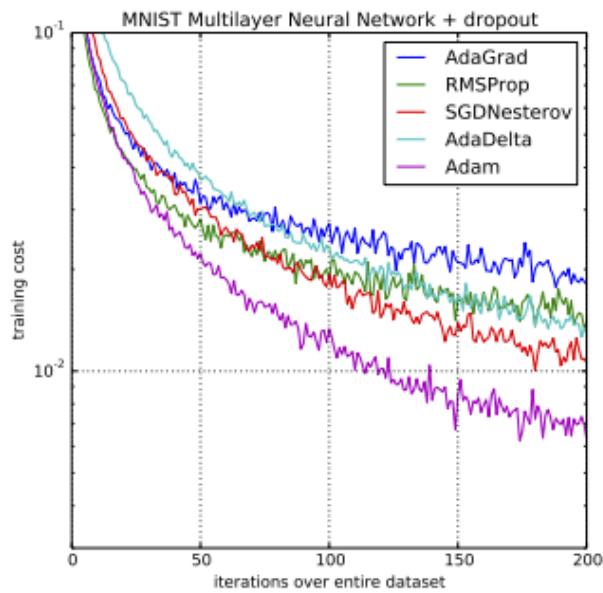


Figure 2.7. Comparison of Adam to other optimization algorithms training a Multilayer Perceptron^a.

^aTaken from Adam: A Method for Stochastic Optimization, 2015

Chapter 3

Tools

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

3.1 Tello

Tello is a small quadcopter that features a Vision Positioning System and an onboard camera. Using its advanced flight controller, it can hover in place and 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 [Tech., 2018].

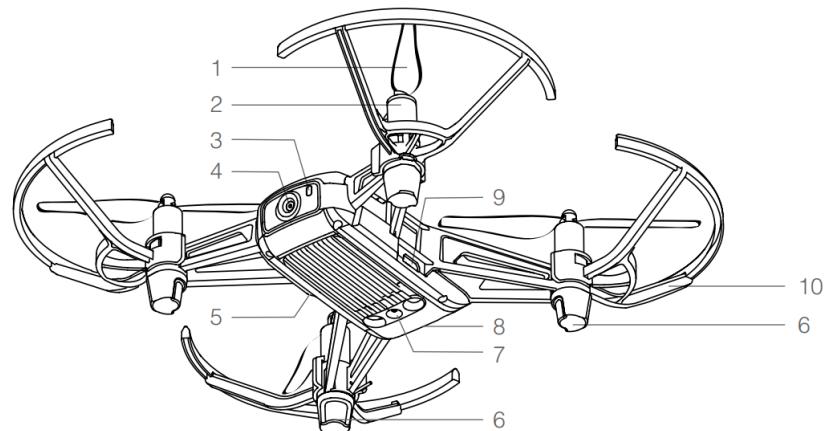


Figure 3.1. 1.Propellers; 2.Motors; 3.Aircraft Status Indicator; 4.Camera; 5.Power Button; 6.Antennas; 7.Vision Positioning System; 8.Flight Battery; 9.Micro USB Port; 10.Propeller Guards.

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

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

3.2 Gazebo

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

3.3 Frameworks

DJITelloPy DJI Tello drone python interface 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 ML².

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. manipulation³.

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 tools⁶.

Matplotlib is a comprehensive package for creating static, animated, and interactive visualisations in Python⁷.

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 Matplotlib⁹.

⁸<http://seaborn.pydata.org/>

⁹<https://scikit-learn.org>

Chapter 4

Methodologies

This chapter illustrates the methodology used to approach the development process of this work. The application is divided into three main independent instances: gesture recognition, 3D trajectory detection and drone controller. First of all, the Section 4.1 to create a Deep Neural Network (DNN) model to recognize hand gestures, followed by Section 4.1.1 and 4.1.4 which describes the type of data used for this purpose and how they are generated. After that, it will be shown how orientation and camera-hand distance has been 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.

4.1 Hand Gesture Recognition

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

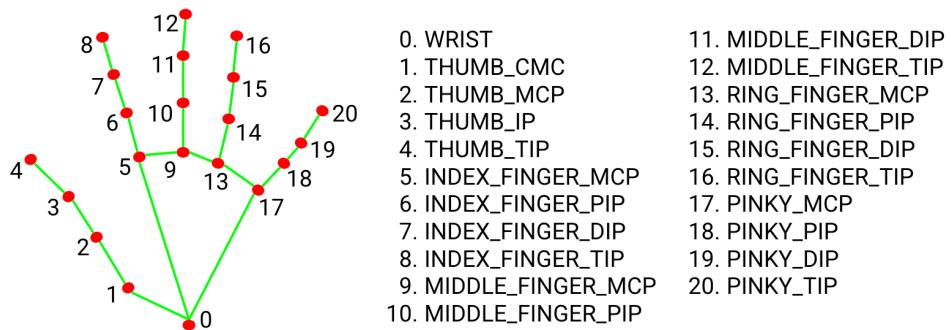


Figure 4.1. Image from the open MediaPipe repository.

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 in order to match mean with origin. Note that shifting the data did not change how the data points are positioned relative to each other. Finally, each point has been scaled 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

Following the normalization of the data, gestures that would allow to acquire a 3D trajectory and to interact directly with the drone have been thought.

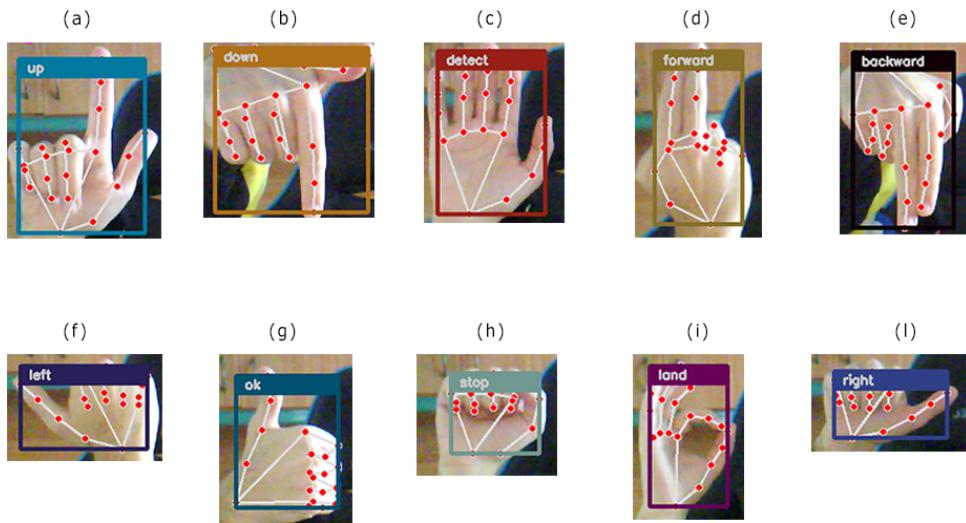


Figure 4.2. Full list of gestures that are available.

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 possibility 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 is 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.

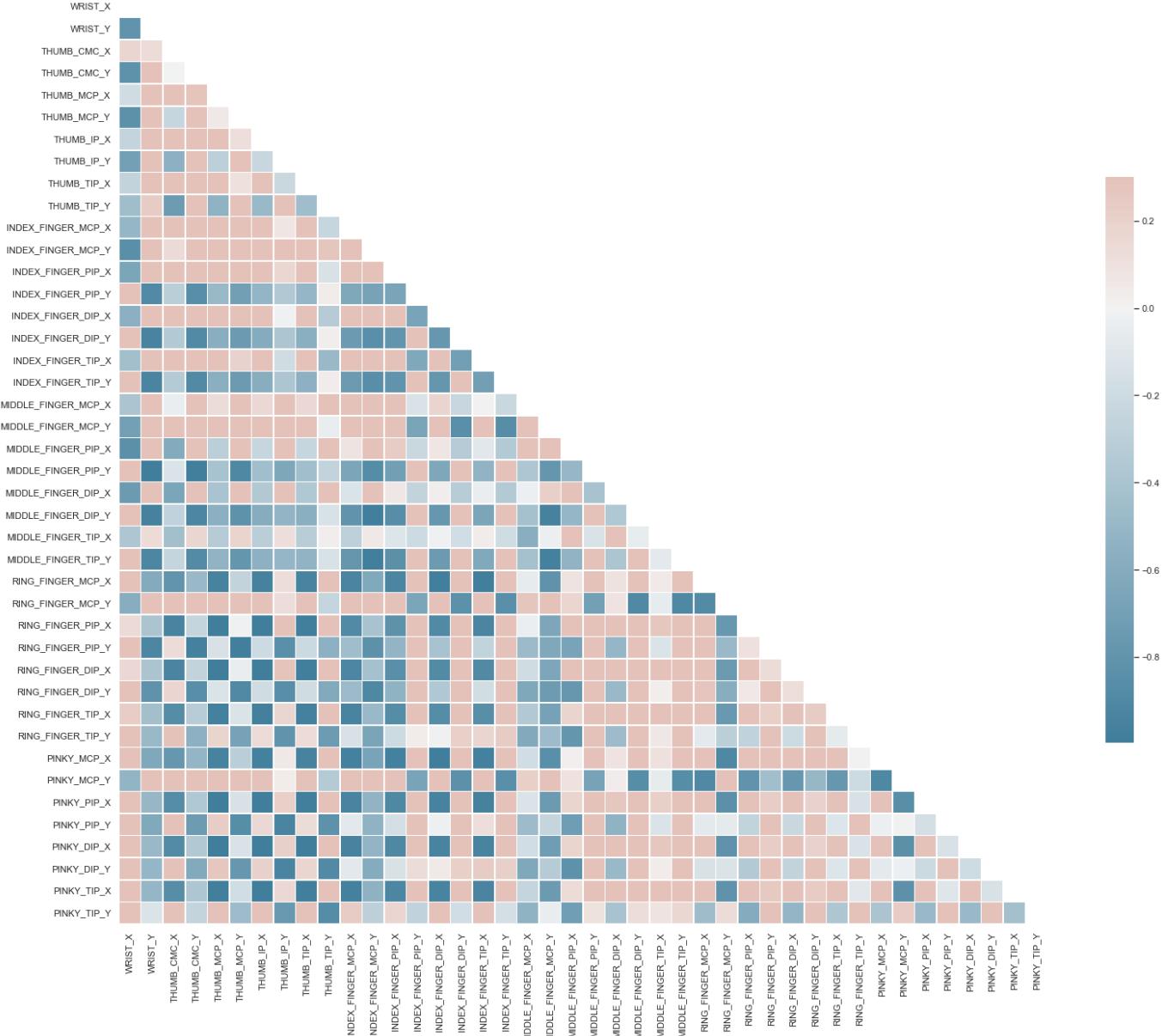


Figure 4.3. Heatmap is really useful to display a general view of numerical data, not to extract specific data point. It represents the correlation matrix and it is done using Pearson correlation.

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:

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

- 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 PCA

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

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 [Cangelosi and Goriely, 2007].

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 for the optimal performance of many machine learning algorithms. An alternative standardization is scaling features to lie between a given minimum and maximum value,

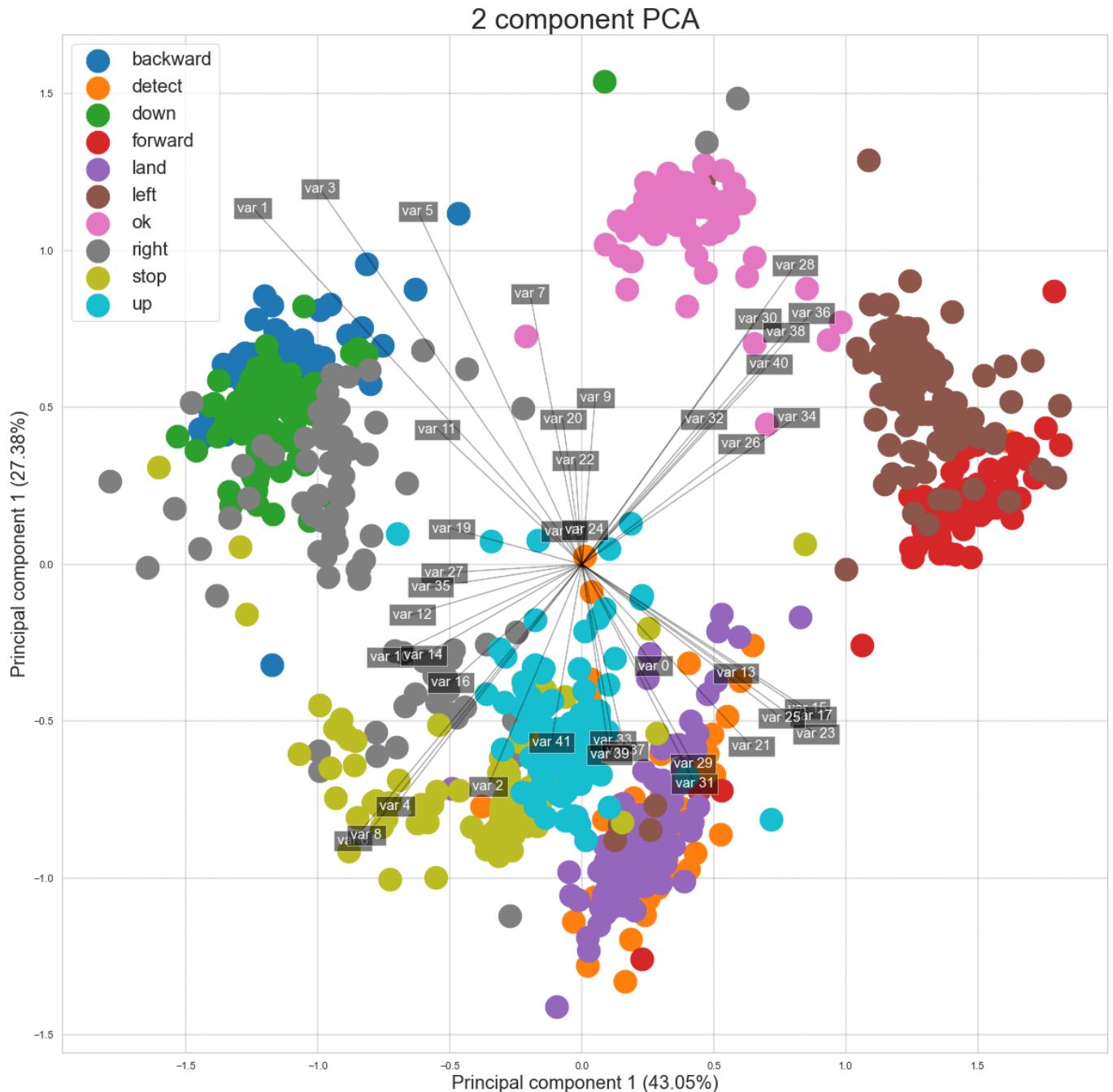


Figure 4.5. After dimensionality reduction, there usually is not a particular meaning assigned to each principal component. The new components are just the two main dimensions of variation.

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

Scree plot (see Fig. 4.6) is another graphical technique useful in PCs retention. 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 .

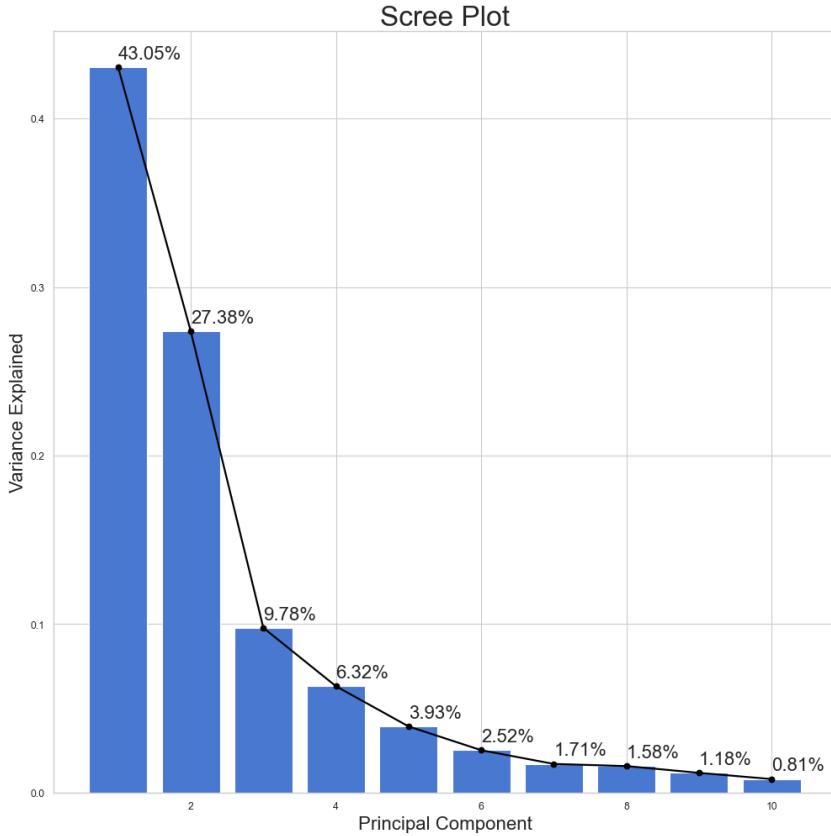


Figure 4.6. Scree plot.

4.1.4 Model

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 has been used, with two hidden layers composed of 30 and 10 neurons each. Input layer is composed of 42 elements and the output layer of 10 classes (see Fig. 4.7). The model must choose among ten classes, so the gestures described in 4.1.1. The activation function is the Leaky ReLU and Adam the optimizer with learning rate 0.1, decay steps 10000 and decay rate 0.96. About the steps, for the training phase 3000 as value has been chosen. Loss is calculated by using softmax cross entropy.

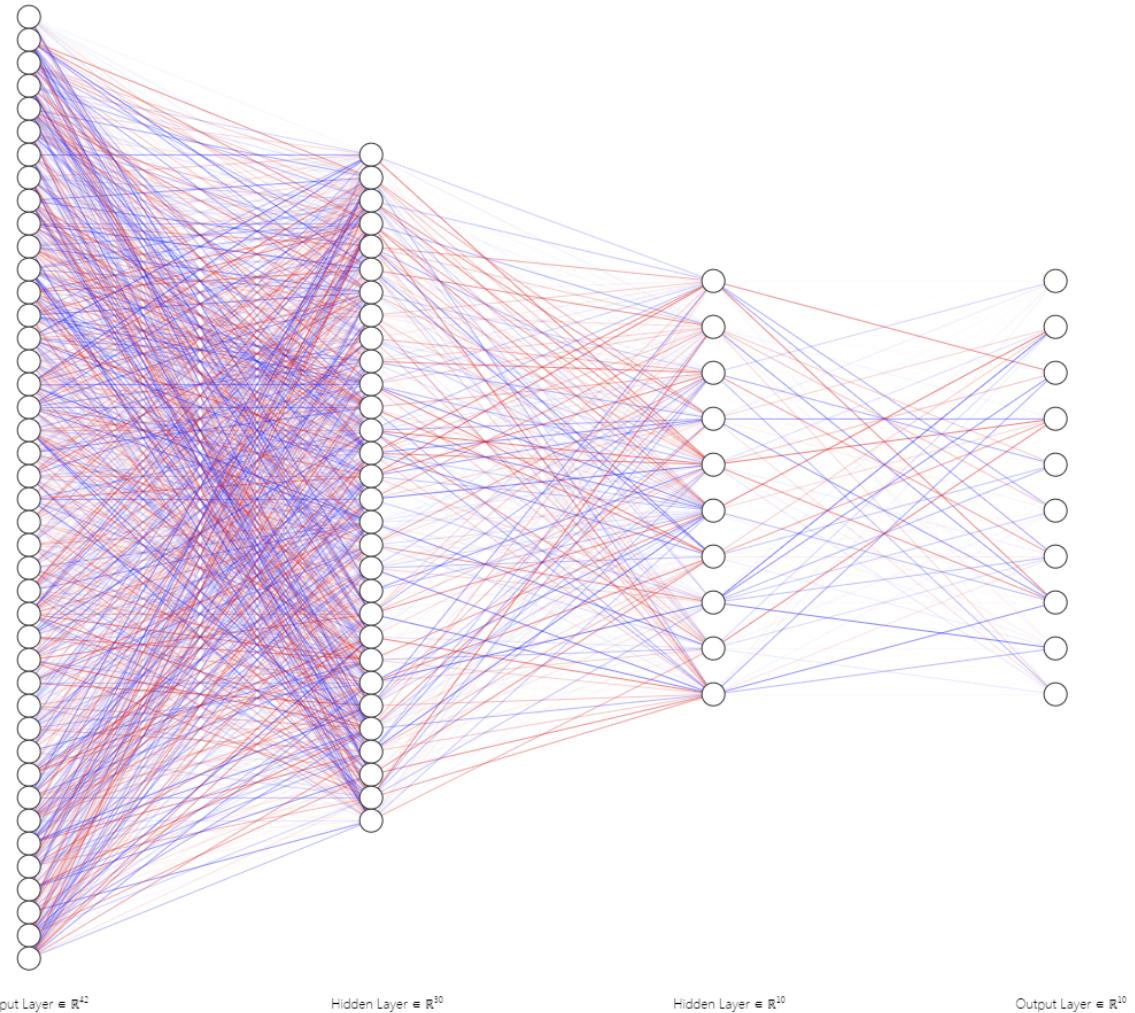


Figure 4.7. DNN image of the hand gesture recognition, generated by NNSVG¹. This is the specific case when all the 42 variables were given as input.

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

4.2 3D trajectory detection

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

4.2.1 Orientation estimation

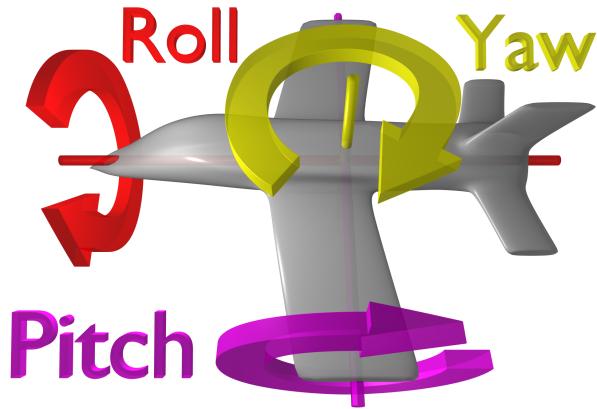


Figure 4.8. Aircraft principal axes.

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 roll is computed at the time of normalization, as described in. The normalization in this phase occurs by first of all making the translation so that the origin of the axes matches the midpoint of the hand, then all the points of the hand are scaled so that the maximum distance between them and the origin has a value of 1. Finally, 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 [CMS]. 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.

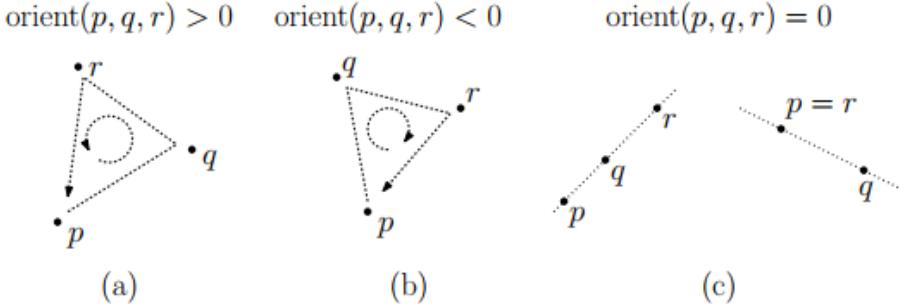


Figure 4.9. Orientations of the ordered triple (p, q, r) .

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

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

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 vertical axis passing over the wrist and the tip of the middle finger has been found (see fig. 4.1).

4.2.1.3 Yaw estimation

First of all, it is necessary to fix a hand to estimate the yaw. This project will be carried on the left-hand, so each comment will be in that direction. Despite this, the same can be said for the right hand. Then, the orientation test (see Eq. 4.1) of three points

$\langle p, q, r \rangle$ is computed. As already said, the order here is important to find the direction of orientation, therefore to know if the left-hand points left or right and with which intensity. Empirically, if $roll < -5^\circ$ or $roll > 5^\circ$ it is better to consider $p = \text{index_finger_mcp}$, $q = \text{index_finger_pip}$ and $r = \text{index_finger_dip}$. Otherwise, if $-5^\circ < roll < 5^\circ$ results are good if $p = \text{middle_finger_mcp}$, $q = \text{middle_finger_pip}$ and $r = \text{middle_finger_dip}$. These hand points were chosen because they scale quite well with hand orientation.

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

4.2.1.4 Pitch estimation

To compute pitch estimation, the middle point m between `index_finger_mcp` and `index_finger_pip` were computed and then the difference between m and the `thumb_tip` w.r.t. vertical axis was done (see fig. 4.1). As for yaw estimation, the value obtained is elevated to the second power, in order to be more stable around 0° . Then, the result was divided by 31 (empirically computed), to bring the results to the range of -90° to 90° . Since the quadratic form was used, if the value is greater than 90° or smaller than -90° the results tends to be really big, therefore the part the exceeds has been reduced to be just the 10% of it.

4.2.2 Camera-hand distance estimation

In the initial phase of identifying a 3D trajectory, the palm of the hand is perpendicular to the range of action from the camera lens. At the beginning, the hand is still and in the acquired image the same hand has a certain area h . If the central point p of the hand is considered as the average distance between 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.

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 can be altered due to the orientation of the hand. 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.

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. 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 3×3 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 [Yaw].

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

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

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

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

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

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

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

$$R(\alpha, \beta, \gamma) = R_X(\alpha) \cdot R_Y(\gamma) \cdot R_Z(\beta) =$$

$$\begin{pmatrix} \cos\alpha \cos\beta & \cos\alpha \sin\beta \sin\gamma - \sin\alpha \cos\gamma & \cos\alpha \sin\beta \cos\gamma + \sin\alpha \sin\beta & 0 \\ \sin\alpha \cos\beta & \sin\alpha \sin\beta \sin\gamma + \cos\alpha \cos\gamma & \sin\alpha \sin\beta \cos\gamma - \cos\alpha \sin\gamma & 0 \\ -\sin\beta & \cos\beta \sin\gamma & \cos\beta \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.5)$$

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 4.10), in which oscillation can occur between points when interpolating using high-degree polynomials.

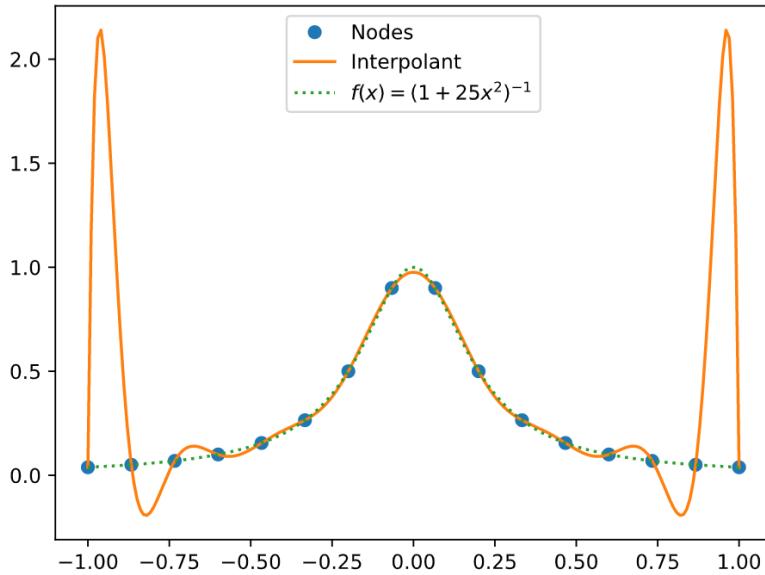


Figure 4.10. Runge's phenomenon problem.

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 .

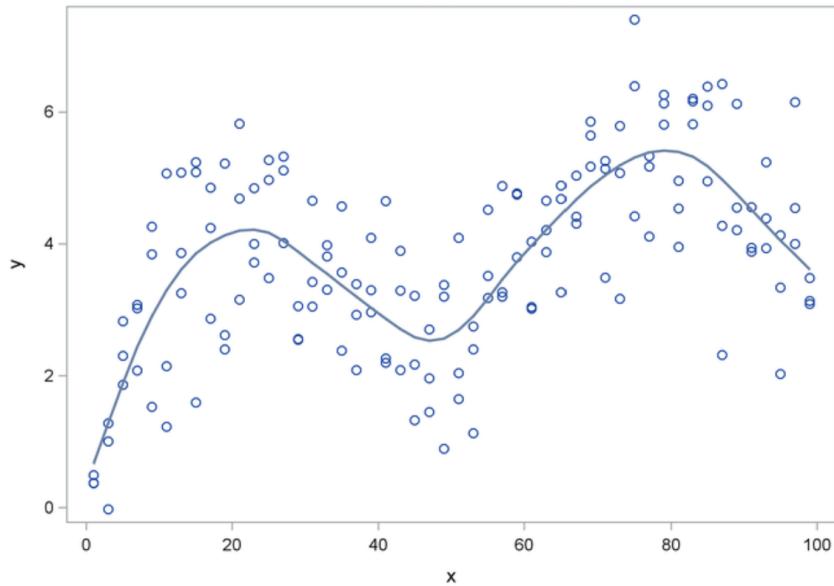


Figure 4.11. Smoothing spline 1D

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 L_2 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 L_2 -norm.

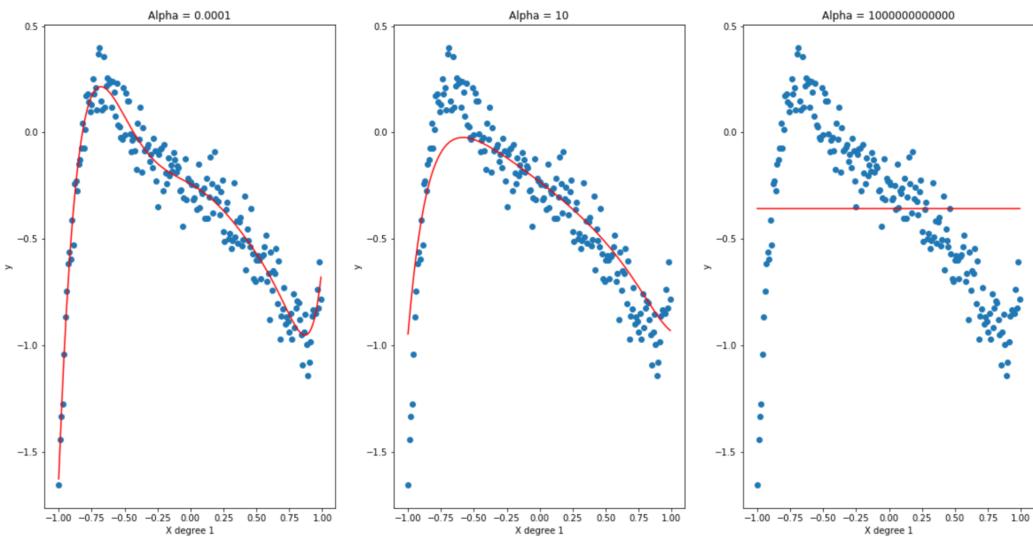


Figure 4.12. Sample fits on a high degree polynomial function with some random noise. As alpha grows larger the model will overgeneralize (under-fit), while at lower values of alpha, the model may be too specific (overfit).

Another way to see the effectiveness of this method is plot the ridge coefficients as a function of the regularization (see Fig. ??). 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.

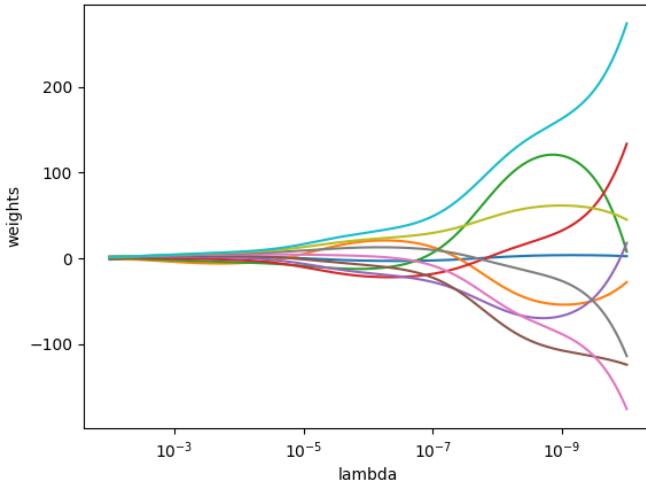


Figure 4.13. Ridge coefficients as a function of the regularization, image from scikit-learn website ^a.

^ahttps://scikit-learn.org/stable/auto_examples/linear_model/plot_ridge_path.html

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 , $X = w^2, w^3, \dots, w^k$ 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 $X_{new} = r^2, r^3, \dots, r^k$ 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 visible in 4.14.

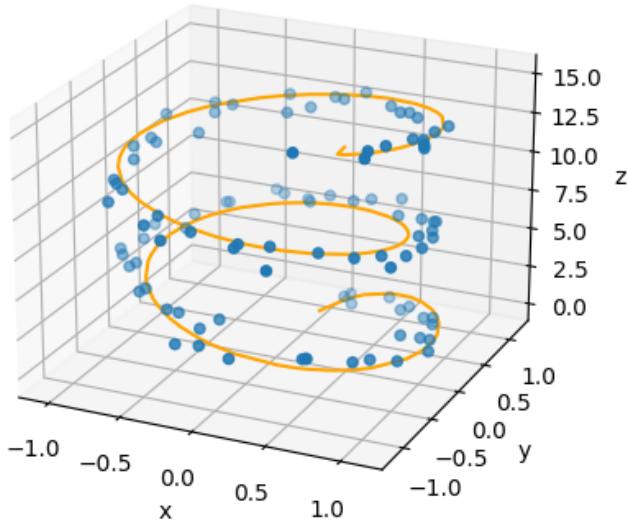


Figure 4.14. Multiple ridge regression with $\lambda = 1e - 10$ as hyperparameter for all the three ridge regression to compute x, y, z values.

4.3 Drone controller

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, simulated air vehicle designed and developed for ROS is ideal. 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 search, a project was found named "tello_ros_gazebo" by the author "bingyo"². The project requires the following dependencies to be executed:

²https://github.com/bingyo/tello_ros_gazebo

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

To remotely control the mobile robot, we have to run several ROS nodes: The first node is ROS master node. The second node is a standard node called `teleop_twist_keyboard`³. This node is constantly checking which keys are pressed on a PC keyboard and 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. This is helpful because Tello is possible drive it in holonomic mode.

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

³https://github.com/ros-teleop/teleop_twist_keyboard

looping at the desired rate. With its argument of 20, we should expect 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, what will be the velocity at that specific time that the robot will assume at the current instant + step, using Inverse kinematics, can be computed. More precisely, if we know the Euclidean distance d between the current and the target position, 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 our 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 [Int].

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 $50cm/s$ [Tech., 2018]. 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 $\times 100$, 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

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. 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 will be displayed and commented first and then those obtained for the estimates of the 3d trajectories.

5.1 Hand gesture recognition model evaluation

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

Metric is different from loss function. Loss functions are functions that show a measure of the model performance and 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.

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. This calculation will be performed on all three models generated. The feature selection model achieved small percentage points higher in accuracy than PCA.

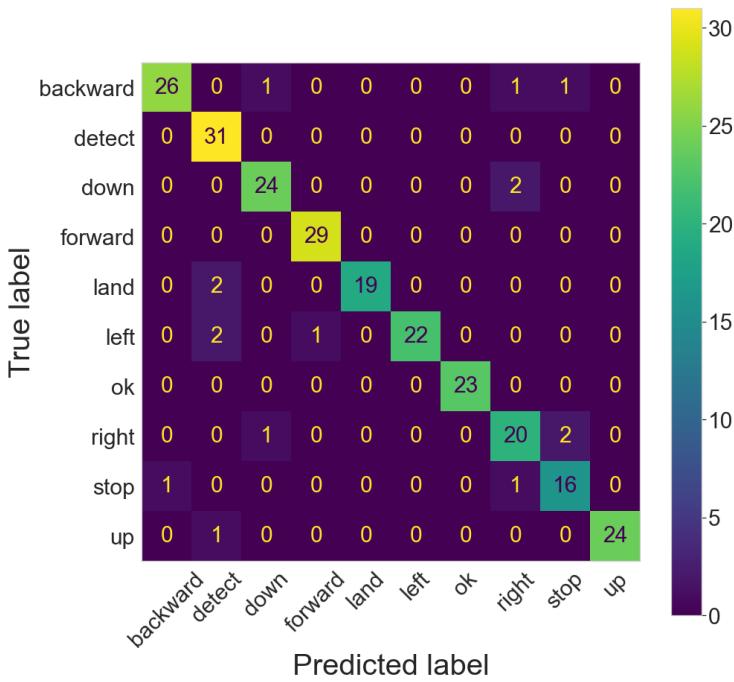


Figure 5.1. Confusion matrix - Features selected.

High accuracy in the "features selected" model was encountered, and then excellent prediction results were found. In fact, in the confusion matrix (see Fig. 5.1) 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.

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. Optimize 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. You may have

	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
<hr/>				
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.

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.

In 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.

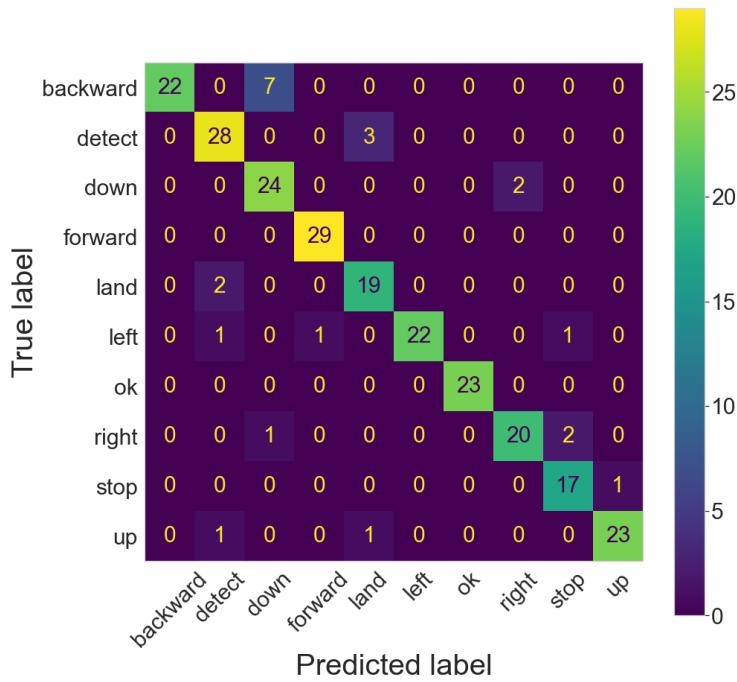


Figure 5.2. Confusion matrix - PCA.

High accuracy in the "PCA" model was encountered, although only the first six principal components have been selected. Here we have slightly less satisfying predictions. In the confusion matrix (see Fig. 5.2) for "forward" and "ok" gestures all the corresponding tests have been correctly predicted, but seven examples labeled as "backward" were instead predicted as "down". The results of the predictions related to the other gestures are better.

	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.

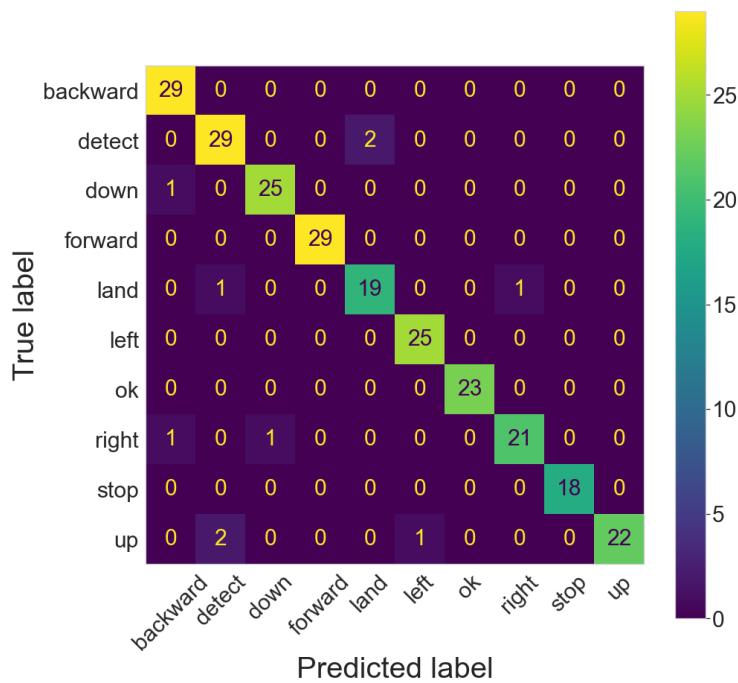


Figure 5.3. Confusion matrix - All features.

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. 5.3) 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.

	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
<hr/>				
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.

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.

5.2 3D trajectory evaluation

After fitting data, is important evaluate the goodness of fit. A visual examination of the fitted curve should be the first step. Beyond that, plenty methods to assess goodness of fit for both linear and nonlinear parametric fits are provided in literature.

5.2.1 Fitting trajectory

In 5.4 are visible nine different frames, 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 yellow the color tends, 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 (given the assumptions of least-squares fitting), in which the model coefficients can be estimated with little uncertainty, or that explains a high proportion of the variability in your data, and is able to predict new observations with high certainty.

A particular application might dictate still other aspects of model fitting that are important to achieving a good fit, such as a simple model that is easy to interpret. It is possible group into two types the methods of analysis [Eva]: 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 are therefore 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, was evaluated:

- **R-square**

This statistic measures how successful the fit is in explaining the variation of the data. Put another way, R-square 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-square is defined as the ratio of the sum of squares of the Regression (SSR) and the Sum of Squares Total (SST). R-square is expressed as

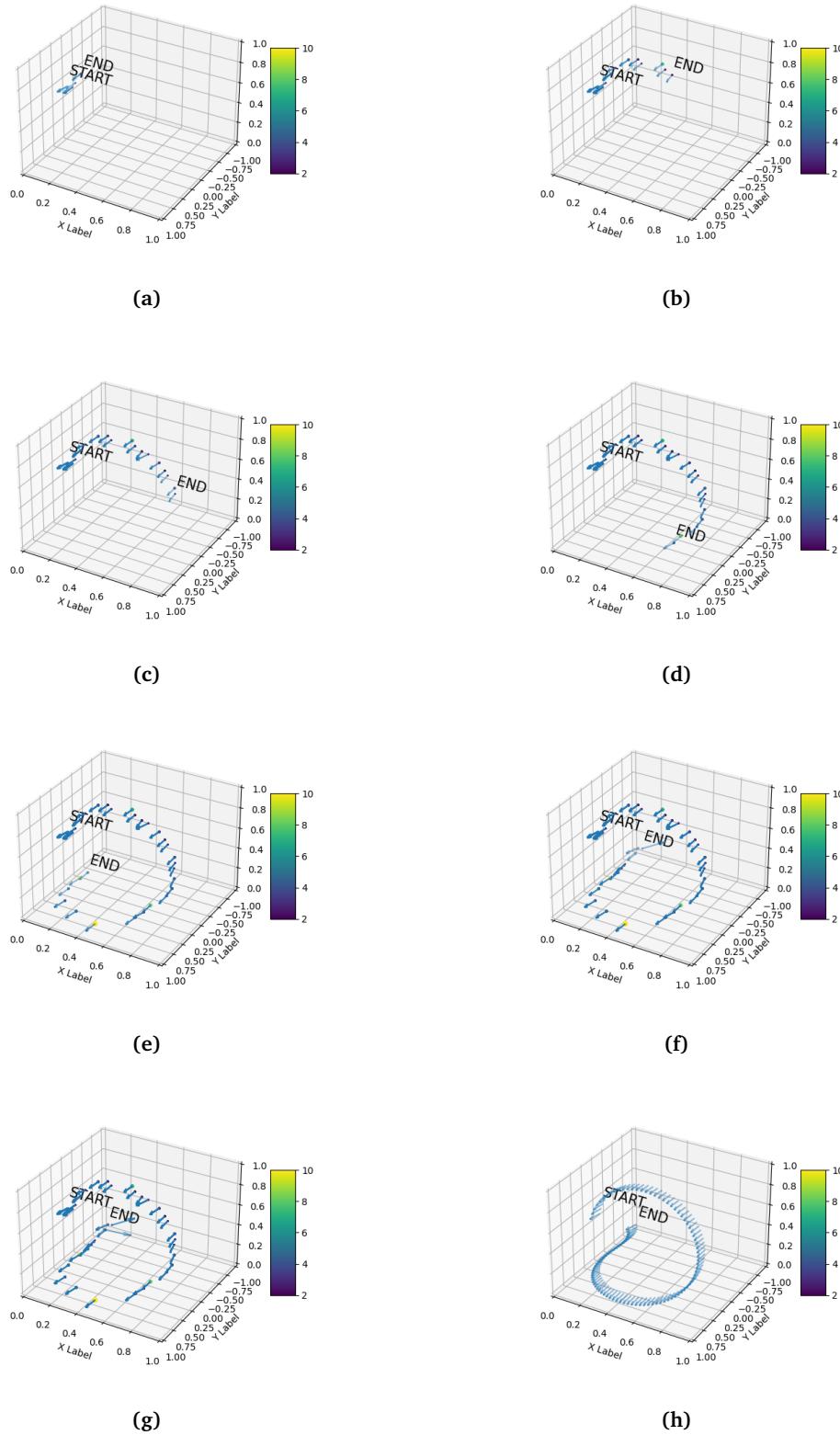


Figure 5.4. Complete acquisition of a 3d trajectory in all its steps.

$$\begin{aligned}
 R^2 &= \frac{SSR}{SST} = \\
 &= \frac{\sum_{i=1}^n w_i (\hat{y}_i - \bar{y})^2}{\sum_{i=1}^n w_i (y_i - \bar{y})^2}
 \end{aligned} \tag{5.1}$$

Equation 5.1. R-square

R-square 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-Square is a good measure to determine how well the model fits the dependent variables. However, it does not take into consideration of overfitting problem. If your regression model has many independent variables, 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-square for equations that do not contain a constant term. Because R-square 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-square is negative. In this case, R-square cannot be interpreted as the square of a correlation. Such situations indicate that a constant term should be added to the model;

- **Adjusted R-square**

This statistic uses the R-square 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-Square.

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-square 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.

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

Equation 5.3. R-square.

The adjusted R-square 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

$$\begin{aligned} RMSE &= \sqrt{MSE} \\ &= \sqrt{\frac{\sum_{i=1}^n w_i (y_i - \hat{y})^2}{v}} \end{aligned} \quad (5.4)$$

Equation 5.4. RMSE.

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 5.4 were calculated on it 3 ridge regression for x , y and z component, as explained in 4.2.4, increasing the degree of the fitting polynomial from 2 to 9.

In 5.5 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.

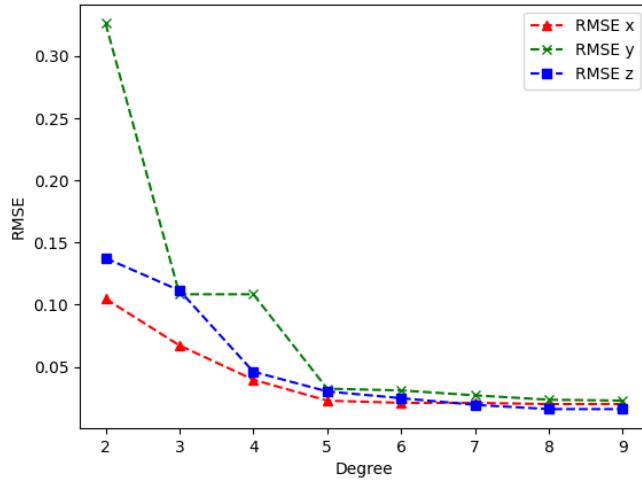


Figure 5.5. RMSE plot.

In 5.6 is present the R-square plot. 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 we take a grade higher than four then we get excellent values capable of capturing more than 90% of variance.

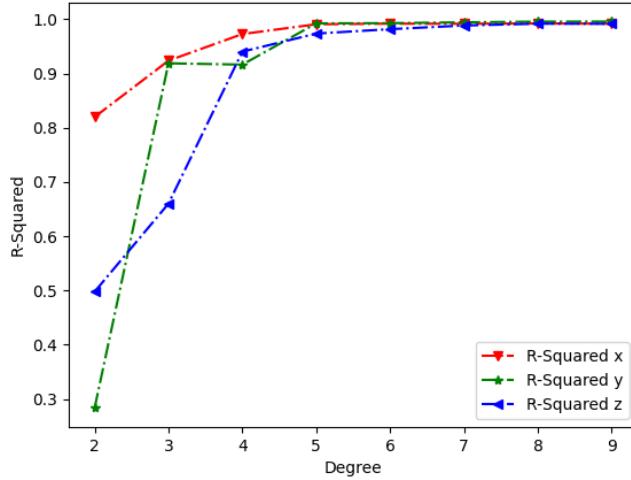


Figure 5.6. R-Squared plot.

In 5.7 is present the Adjusted R-squared plot. 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 similar to the previous one (see Fig. 5.5), 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 we can perfectly explain the trajectory.

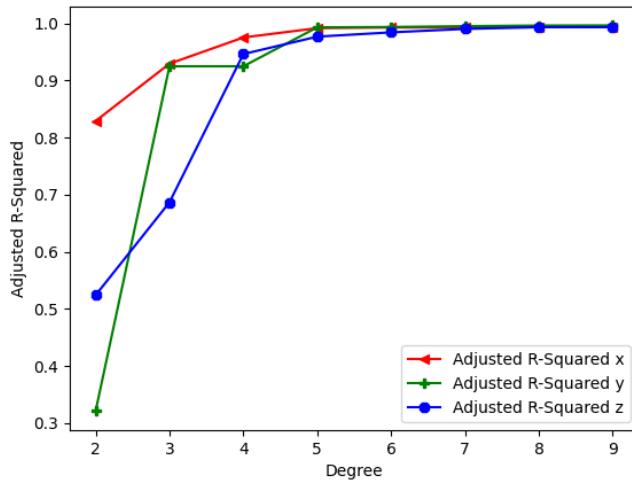


Figure 5.7. Adjusted R-Squared plot.

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 Drone in action

5.4 Real

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.

Conclusions and Future works

Appendix A

List of Acronyms

1D	one-dimensional
2D	two-dimensional
3D	three-dimensional
ANN	Artificial Neural Network
AR	Augmented Reality
CNN	Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
ML	Machine Learning
MSE	Mean Squared Error
NLP	Natural Language Processing
NN	Neural Network
PC	Principal Component
PCA	Principal Component Analysis
POC	Proof of Concept
RGB	Red Green Blue
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network

ROS	Robot Operating System
SSE	Sum of Squared estimate of Errors
SSR	sum of squares of the Regression
SST	Sum of Squares Total.....
UAV	Unmanned Aerial Vehicles
URDF	Unified Robot Description Format
VR	Virtual Reality

Bibliography

Animating robots instead of programming them | andyrobot + kuka - youtube. <https://www.youtube.com/watch?v=Cd0aGU04MyM>. (Accessed on 02/06/2022).

Cmsc 754. <https://www.cs.umd.edu/class/spring2020/cmsc754/lectures.html>. (Accessed on 02/01/2022).

Evaluating goodness of fit - matlab & simulink - mathworks italia. <https://it.mathworks.com/help/curvefit/evaluating-goodness-of-fit.html>. (Accessed on 02/05/2022).

Interpolation - wikipedia. https://en.wikipedia.org/wiki/Interpolation#cite_note-1. (Accessed on 02/02/2022).

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

List of uav-related incidents - wikipedia. https://en.wikipedia.org/wiki/List_of_UAV-related_incidents. (Accessed on 02/06/2022).

Oculus connect 6: Introducing hand tracking on oculus quest, facebook horizon, and more. <https://www.oculus.com/blog/oculus-connect-6-introducing-hand-tracking-on-oculus-quest-facebook-horizon-and-more/>. (Accessed on 02/06/2022).

Sisu cinema robotics. <https://www.sisucinemarobotics.com/>. (Accessed on 02/06/2022).

Top tips for drone cinematography. <https://dakona.co.uk/top-tips-for-drone-cinematography/>. (Accessed on 02/06/2022).

What happens when your drone runs out of battery – drone tech planet. <https://www.dronetechplanet.com/what-happens-when-your-drone-runs-out-of-battery/>. (Accessed on 02/06/2022).

- Yaw, pitch, and roll rotations. <http://planning.cs.uiuc.edu/node102.html>. (Accessed on 01/31/2022).
- Diego Andina, D. Pham, D. Andina, Antonio Vega-Corona, Juan Sejas, and J. Torres-García. *Neural Networks Historical Review*. 02 2007. ISBN 978-0-387-37450-5. doi: 10.1007/0-387-37452-3_2.
- Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.
- Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- Richard Cangelosi and Alain Goriely. Component retention in principal component analysis with application to cdna microarray data. *Biology direct*, 2(1):1–21, 2007.
- W. W. Hager and H. Zhang. A survey of nonlinear conjugate gradient methods, 2006. Pacific journal of Optimization, vol. 2, no. 1, pp. 35–58.
- Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- Camillo Lugaressi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Ubweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, et al. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172*, 2019.
- Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. Citeseer.
- Iason Oikonomidis, Nikolaos Kyriazis, and Antonis A Argyros. Efficient model-based 3d tracking of hand articulations using kinect. In *BmVC*, volume 1, page 3, 2011.
- Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.
- Dewi Retno Sari Saputro and Purnami Widyaningsih. Limited memory broyden-fletcher-goldfarb-shanno (l-bfgs) method for the parameter estimation on geographically weighted ordinal logistic regression model (gwolr). In *AIP Conference Proceedings*, volume 1868, page 040009. AIP Publishing LLC, 2017.
- Ryze Tech. Tello user manual v1.2, 2018. Available online: <https://www.ryzerobotics.com/> (accessed on 21 January 2022).
- Iliya Valchanov. Sum of squares total, sum of squares regression and sum of squares error, 2018. URL <https://365datascience.com/tutorials/statistics->

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

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

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