



CENTRO DE INVESTIGACIÓN Y DE
ESTUDIOS AVANZADOS DEL INSTITUTO
POLITÉCNICO NACIONAL

DEPARTAMENTO DE COMPUTACIÓN

**Diseño y Desarrollo de un Sistema de Reconocimiento de Gestos
Manuales para el Control de un VANT**

Tesis que presenta

Dulce Adriana GÓMEZ ROSAL

para obtener el grado de

**Maestra en Ciencias
en Computación**

Directora de Tesis:

Dra. Xiaoou LI

Ciudad de México

Septiembre de 2016



CENTER FOR RESEARCH AND ADVANCED
STUDIES OF THE NATIONAL
POLYTECHNIC INSTITUTE

COMPUTER SCIENCE DEPARTMENT

**Design and Development of a Hand Gesture Recognition System for
UAV Control**

by

Dulce Adriana GÓMEZ ROSAL

Thesis submitted in partial fulfillment of the requirements for the
degree of

**Master of Science
in Computer Science**

Advisor:

Dr. Xiaou LI

Mexico City

September 2016

Dedicated to my family, in every form.

"Nullius in verba. "

Royal Society of London

Agradecimientos

Todo lo mostrado y no mostrado en éste documento, producto de intensos meses de continuo trabajo, fue posible gracias a diversas formas de ayuda que recibí en éste tiempo. Por ello, dedico un inicial agradecimiento al CONACyT, quien posibilitó económicamente éstos estudios de posgrado.

Las labores administrativas pueden llegar a ser un dolor de cabeza, y por ello agradezco enormemente las facilidades que el Dr. Amilcar Meneses Viveros, Sofía Reza Cruz y Erika Ríos me proporcionaron.

Por su lado, al ser la médula espinal de éste trabajo, las labores académicas requieren de un agradecimiento particular. A la Dra. Xiaou Li por su apoyo en diversos aspectos, al Dr. Jiacun Wang por su disponibilidad y atención hacia mi persona y todo el proyecto, a la Dra. Sonia Mendoza por su cortesía y favor hacia todo el proceso documental, al Dr. Jair Cervantes por su consideración y miramiento hacia ésta tesis, y finalmente al CINVESTAV y a la Universidad de Monmouth, por la presteza de los recursos tecnológicos e instalaciones.

A mi familia sanguínea, Lic. María del Rosario Rosal Montiel, Biól. Nancy Vianey Gómez Rosal e Ing. Sergio Gerardo Gómez Rosal, hállese aquí el producto de todo su infinito amor y paciencia, a los cuales las palabras le quedan cortas.. pero, ¡lo logramos!

A mi familia de elección, amigos que me permiten no salir de mis cabales, entienden mis momentos de lejanía y sobretodo, estaban siempre prestos con su afecto y lealtad a cuando el proyecto me regalaba micromomentos de distensión: a Berenice Jimenez, Álvaro Becerril, Mario Peña, Ariana Cortés, Elizabeth Martínez, Daniela Brindis, Mauricio Sánchez, Alicia Cruz, Fany Ramírez, Elia Peredo, David Quevedo, Uriel Silva, Kim y Frannie Festejo y varios (¡muchos!) más, que aún sin nombrarlos, se saben parte de mi vida y éste proyecto.

Un agradecimiento especial merecen quienes con al menos una palabra o un momento, me regalaron un poco de su experiencia y sapiencia en las artes de la Computación: al Dr. Francisco Rodríguez Henríquez, Dr. Gabriel Ramírez Torres, Dr. Luis Gerardo de la Fraga, Dra. Dolores Lara, Axel Salazar, Guillermo Monroy y Daniel Torres, por mencionar algunos.

Sería imposible listar todas las personas que merecen al menos una pequeña mención en éste producto, es por ello que en atención a ésta falta, extiendo un penúltimo agradecimiento a todos los involucrados de una u otra forma no mencionada hasta el momento.

Y finalmente, gracias a usted, estimado lector, por dedicar minutos de su tiempo a éste documento. En ello va la motivación real de todo éste trabajo.

Resumen

Al comunicarnos con las personas, usamos, consciente o inconscientemente, gestos. ¿Es posible que los humanos se comuniquen con un dispositivo inteligente, como un VANT (Vehículo Aéreo No Tripulado), a través de gestos manuales? Éste trabajo de tesis se enfoca en responder ésta pregunta, y su objetivo es diseñar y desarrollar un sistema de reconocimiento de gestos manuales para controlar un VANT cuadrirrotor.

En el sistema, 13 gestos fueron diseñados (5 estáticos y 8 dinámicos) para controlar la operación y dirección de un VANT. Para entrenar al sistema a reconocer los gestos, se usaron datos esqueléticos de la mano recolectados a través del dispositivo Leap Motion (6 características puras en total). El Clasificador Adaptativo Ingenuo de Bayes (ANBC por sus siglas en inglés *Adaptive Naïve Bayes Classifier*) y las Máquinas de Soporte Vectorial (SVM, *Support Vector Machines*) son seleccionados como los algoritmos base de aprendizaje para el método de ensamble desarrollado; ésto fue requerido para mejorar la precisión en el reconocimiento. Para reconocer gestos dinámicos utilizando los clasificadores estáticos, se presenta un *enfoque temporal*, en el cual se crean características a través de la composición de datos secuenciales de la mano, ésto generó en total 8 características compuestas. Con ello, los clasificadores son entrenados utilizando datos de 14 características (6 puras y 8 compuestas) en vez de 6 características puras para reconocer gestos dinámicos a través de clasificadores estáticos.

Los componentes de software son tres nodos del Sistema Operativo Robótico (ROS, por sus siglas en inglés, *Robot Operating System*) que generan las funciones de *creación del vector de características, reconocimiento de gestos y control del VANT*. La plataforma ROS se ha vuelto el estándar para la robótica industrial, de investigación y entretenimiento. Por tanto, ésta implementación puede ser utilizada por cualquier otro dispositivo compatible con ROS.

Para probar el sistema, se usó el software Gazebo (un simulador de ROS) así como un VANT emulado (con el software ArduCopter). Los resultados indican que el sistema desarrollado es capaz de controlar un VANT de manera estable.

Keywords: Reconocimiento de Gestos Manuales, Leap Motion, ROS, control de VANT.

Abstract

When communicating with each other, we use, conscious or unconsciously, gestures. Can human communicate with an intelligent device, such as an UAV (Unmanned Aerial Vehicle), through hand gestures? This thesis work focuses on answering this question, and its aim is to design and develop a hand gesture recognition system to control a quadrotor UAV.

In the system, 13 gestures were designed (5 statics and 8 dynamics) to control the UAV operation and direction. In order to train the system to recognize these gestures, hand skeleton data were collected from the Leap Motion device (6 raw features in total). Adaptive Naïve Bayes Classifier (ANBC) and Support Vector Machines (SVM) are selected as the base learning algorithms for the ensemble method we developed; this was required to improve the recognition accuracy. To recognize dynamic gestures using the static classifiers, we introduce the *temporal approach* in which the features are created by composing sequential hand data; this provided in total 8 composed features. Therefore, the classifiers are trained using data of 14 features (6 raw and 8 composed) instead of 6 raw features in pursuance of dynamic gestures recognition through static classifiers.

The software components are three nodes of the Robot Operating System (ROS) that accomplish *feature vector creation*, *gesture recognition*, and *UAV Control* functions. ROS platform has become the standard for industrial, research and entertainment robotics. Therefore, this implementation can be used by any other ROS compatible device.

To test the system, the Gazebo software (a ROS simulator) was used along with an emulated UAV (with the ArduCopter software). Experimental results indicate that the developed system is able to control the UAV stably.

Keywords: Hand Gesture Recognition, Leap Motion, ROS, UAV Control.

Contents

Resumen	vii
Abstract	ix
Contents	xi
List of Figures	xiii
List of Tables	xv
1 Introduction	1
1.1 Motivation	1
1.2 Problem Statement	2
1.3 Project Description	4
1.4 Contributions	5
1.5 Thesis organization	5
2 Hand Gesture Recognition	7
2.1 Human Machine Interface	7
2.1.1 Real-Time Control Systems	8
2.1.2 Quadrotor UAV Control	9
2.2 Hand Gesture Recognition Systems	10
2.3 Hand Gesture Recognition	12
2.3.1 Recognition Devices	12
2.3.2 Other Tools and Applications	13
2.4 Pattern Recognition	14
2.4.1 Recognition Process	16
2.4.2 Gesture Recognition	18
2.5 Machine Learning Fundamentals	19
2.5.1 Preliminaries	19
2.5.2 Types of Learning	20
2.5.3 Stages of the Machine Learning Process	22
Training, Testing and Validating sets	23
The Peaking Phenomenon	24
2.5.4 Validation Methods	25
Test Sets and Errors	25
The Confusion Matrix	25
2.6 Supervised Learning Algorithms	26
2.6.1 Bayesian algorithms	26
2.6.2 Decision Trees	27

2.6.3	K-Nearest Neighbors	28
2.6.4	Minimum Distances	29
2.6.5	Support Vector Machine	29
2.6.6	Hidden Markov Model	31
2.6.7	Dynamic Time Warping	32
2.6.8	Artificial Neural Networks	32
2.6.9	Algorithms Comparison Chart	33
2.6.10	Ensemble Methods	36
2.7	Tested Algorithms	36
3	Hand Gesture Recognition System for UAV Control	39
3.1	Architecture of the System	39
3.2	Gesture Recognition subsystem	40
3.2.1	The Leap Motion Controller	41
3.2.2	Data Filters	42
3.2.3	The Feature Vector	43
3.2.4	Classification	47
3.2.5	Base Learners	48
3.3	UAV Control Subsystem	49
3.3.1	UAS Elements	51
3.4	Implementation	52
3.4.1	The Robotic Operating System (ROS)	52
3.4.2	ROS Nodes	52
3.4.3	System Pseudocodes	55
3.4.4	Simulation on Gazebo	58
4	Experiment	63
4.1	Experiment Design	63
4.2	Evaluation Description	65
4.2.1	Training Data	66
4.2.2	Feature Combinations	67
4.2.3	Predominant Features	70
4.3	Static Tests: <i>Test Subset</i> Accuracy Response	71
4.3.1	Class Accuracy	71
4.4	Dynamic Tests: Live Data Accuracy Response	76
4.4.1	Multi-classes Confusion Matrixes	76
4.5	Dynamic Tests: Live Data Time Response	82
5	Conclusions and further work	85
5.1	Conclusions	86
5.2	Further Work	88
	Bibliography	89

List of Figures

1.1	<i>Telekinesis</i> , a dream cherished for a long time.	3
2.1	A Diagram of a Closed Loop Control System.	8
2.2	A quadrotor UAV, <i>Parrot</i> brand.	9
2.3	Quadcopter attitude axes.	10
2.4	Diagram of a Hand Gesture Recognizer used in a closed loop control system . . .	11
2.5	Examples of approaches: a) vision based b) data-glove based c) colored glove . .	12
2.6	A sample of the astrology across the mayan, egyptian and greek cultures.	15
2.7	The precolombian cultures shared the myth referring a rabbit in the moon.	15
2.8	A simple classification process.	17
2.9	Workflow of a conventional pattern recognition implementation with a supervised learning algorithm.	18
2.10	Supervised Learning.	22
2.11	Overview of applying a Supervised Learning Algorithm to a Problem.	23
2.12	The Peaking Problem.	24
2.13	Example of the probability density functions for a three classes Naive Bayes classifier, the class with greatest likelihood is selected as the output class.	27
2.14	Example of a 2-Classes case, where KNN provides a different class accordingly to the number of surrounding samples.	28
2.15	Example of a 2-Class Minimum Distances classifier.	29
2.16	Example of the <i>kernel trick</i> . Since the data is not linearly separable, it is transformed to a higher space where it becomes separable.	30
2.17	A representation of the HMM.	31
2.18	A graphical representation of how DTW obtains the distance in time of two out-of-phase signals.	32
3.1	A logic view of the system.	40
3.2	Gesture recognizer subsystem architecture.	41
3.3	Leap Motion controller, cover removed	41
3.4	Leap Motion coordinate system and its correspondence with a screen.	42
3.5	Difference across Frames	44
3.6	Palm Delta vector (Δ Palm).	45
3.7	Leap Motion Palm Vectors	46
3.8	Leap Motion Finger Vectors	46
3.9	Hand angular movements.	47
3.10	Hand Gesture Recognizer node.	48
3.11	UAV Control Architecture.	50
3.12	Erle Brain 2 and Erle Copter.	51
3.13	Interaction between ROS nodes.	53

3.14	Graph with nodes and topics used in the recognition phase and the consequent UAV control.	55
3.15	Communication between virtualized components and nodes in the simulation. .	60
3.16	Node graph of simulated system.	61
3.17	Simulation of the system in Gazebo.	62
4.1	Workflow of current supervised learning algorithm implementation.	64
4.2	Data captured for Up gesture.	70
4.3	Data captured for left rotation gesture.	71
4.4	Comparison between average accuracy for each classifier in the test set validation.	75
4.5	Comparison between average accuracy for each classifier in the live validation. .	81

List of Tables

2.1	Example of a confusion matrix	26
2.2	First part of a qualitative comparison of some Supervised Learning algorithms	34
2.3	Second part of a qualitative comparison of some Supervised Learning algorithms	35
3.1	Relation between target class c_j and the redognized hand gestures.	49
3.2	Relation between target class c_j , hand movements and UAV behaviour.	50
4.1	Sample number per training data set and per class c_j	67
4.2	Total sample number per training data set.	67
4.3	Feature vector combinations.	69
4.4	A sample of the results in the static validation: Test set accuracy for dynamic gestures with the Combination D.	73
4.5	Average of accuracy results in the static validation.	74
4.6	A sample of the results in the dynamic validation: Live data accuracy for dynamic gestures with the Combination B for algorithm ANBC.	77
4.7	Average of accuracy results for the main diagonal in the live validation.	79
4.8	Average time response (in miliseconds) of classifiers across combinations in the live validation.	82

Chapter 1

Introduction

Humans love to communicate. Communication is the channel through which we make ourselves noticeable to others and in general, to the world. Despite the fact that each region of the world has their own communication codes, it is undeniable that there is still an universal language: **gestures**. Most communication is nonverbal, gestures is a basic nonverbal communication tool. Among gestures, hand gestures plays an important role in our life. Considering the current technological development, why do not we extend gesture communication to devices that we want to communicate with? This question has been partially answered by the development of the research field named *Gesture Recognition*, a subfield of the *Pattern Recognition*, branch of *Machine Learning*.

This introductory chapter explains the motivations of this work, plots the problem statement, presents the project description, lists the contributions of this thesis and finally outlines the organization of the rest of the document.

1.1 Motivation

Since all humans are different, it is not odd to suppose that we have different communication capabilities and/or preferences. This leads us to argument that: **the interaction possibility through diverse devices, with different technology and sensing channels, widens the public to which a certain application or device is focused**. With this spirit, hardware development enthusiasts have provided us in recent years with novel contact-free input systems based on different methodologies, e.g. speech input, body gestures recognition and eye-gaze control, just to mention a couple.

The introduction of these technologies enables to diversify the application focus. Let us mention a sterile environment (clean room) for instance; in such a place, a standard input device like a regular mouse or keyboard would become devices that violate the sanity terms of the place and therefore the introduction of certain technology is improper. Under the aforementioned scenarios, and moreover, thinking of persons who do not feel either comfortable or capable of interacting with the regular input devices (such as a mouse or keyboard), we can state that:

Different human abilities require different input device features, and consequently such different input device features opens the possibility to alternative software application targets.

Nowadays the alternative interacting channels research and the gesture recognition field has become a very promising field, and this is the project's catalyst: the seek and development of software tools that, making use of alternatives hardware devices and gesture recognition, provide a reliable response to a certain task. In this project, this reliable response will be focused on the movement command of a quadrotor unmanned aerial vehicle (UAV) as an application case.

The hand gestures recognition is a research field that has been addressed with many Machine Learning algorithms, however its response has been evaluated most of the times only in terms of its certainty and still depends on the standard devices used for the information retrieval such as regular cameras. Since the intention of this project is to apply the hand gesture recognition algorithms to the control of a mobile robot, it is mandatory to think in terms of real-time response which generates the need of evaluating the algorithm response not only in certainty terms but in real-time requirements.

Gathering the above mentioned, the main motivation of this project resides in the proposal of new technology usage combined with a comprehensive set of hand gestures recognition algorithms, guided under the control real-time system in order to manipulate mobile hardware (the UAV). The belief is that this proposal gains attention in order to integrate new technologies with solid algorithms for some other non-standard devices.

1.2 Problem Statement

A world where the *Telekinesis* (production of motion in objects without contact or other physical means, but induced by mental or spiritual power) is a reproducible and scientifically verifiable phenomenon still looks like a distant reality, although many characters claim to have this ability such as Édouard Isidore Buguet in the eighteenth century (Figure 1.1). Inspired by this, some enthusiastic technologists have developed hardware, through which human may express commands we want to order to some other objects. Gesture communication is a very natural language, learned since the beginning of mankind, therefore is valid to think that this communication may be the (so far) closest attempt to a command induction for mobile objects.



FIGURE 1.1: *Telekinesis*, a dream cherished for a long time.

However, attempts to control objects through hand gestures have been as varied since the first attempts that several techniques (for hardware and software) were developed; above all, depending on the purpose for which the gestures are used, the techniques greatly vary.

Meanwhile, UAVs have become an interesting subject not only for technologists and amateurs, but for large companies and government sectors that foresee in these devices, a very versatile pioneer. Today the most common control of the UAVs is through explicit commands via a remote control or a control station that generates coordinates and order them where to move. Would not it be interesting to have capabilities of control over the UAV in a gesticulated way?

Combining both topics, on remote handling through hand gestures movement of a UAV, it is that the problem arises to resolve. How to ensure that hand gestures can intuitively control the movement of a UAV?

In recent years, a great interest emerged in the supply of devices that transduce information from manual movements to computer data; enhanced by this, study techniques of these data got refined. Nonetheless, these techniques prove to be more suitable for static hand gestures while for the study of dynamic gestures, there are still problems in terms of hardware requirements or simply the recognition of a right gesture is low.

Most of the people used to think that regular recognizers could deliver a stable recognition independently of the application or entered data, however many researchers who use the traditional techniques have fallen into the notion that there is not ideal and universal recognizer approach and for every desired application, a complete study should be driven in order to propose a suitable approach.

Back when gesture recognition was a field of study in its infancy, people talked about a rigid classification; now, thanks to the development of more flexible mathematical models, the great stride of Machine learning (Computer Science subfield dedicated to creating programs able to generalize behaviors from an unstructured information, usually provided as examples; is a

process of knowledge induction, responsible for developing techniques that allow computers to learn) emerged, which allowed to expand the type of data and applications that the recognizers may cover.

In first instance, it was thought that a recognizer from a particular methodology was sufficient. And it was indeed! although just for simple applications such as only static gestures cases. Later on, the recognizers employee grew more complex, which forced to revise alternative approaches; hence, a drive lane proliferated towards more complex models making use of diversity. Thus, the so-called Ensemble Methodology was developed and its focus is to combine recognition techniques using different learning paradigms. By recent experiments it was found that these methodologies deliver better results than the traditional single recognition approach.

The problem attacked in this project is to control an UAV through hand gestures, both static and dynamic, using an Ensemble Methodology with Supervised Learning algorithms (retrieved from Machine Learning) and it is of great interest because the control of an UAV through different media may contribute to creativity on their applications, while recognizing both types of gestures (static and dynamic) in a single meta-classifier potentiates the uses it may provide and extends the range of approaches to generate a pattern recognition.

1.3 Project Description

This work develops a control system that recognizes hand gestures making use of the information retrieved by the commercial device named *Leap Motion*. The recognizer was designed based on the data generated by the Leap Motion controller, user's hand gestures, an ensemble learning methodology, recognition algorithms that solve the task in a real-time manner and a proposed temporal approach; all this, tested and evaluated with a modified version of the regular comparison tools. A combination of static and dynamic gestures is recognized and mapped to specific label commands, and in order to select the two most suitable base learners, four popular algorithms were tested. As an application case, the usage of the recognized gestures is the control of a quadrotor UAV through the Robot Operating System (ROS) platform.

The gesture recognition phase starts with the hand skeleton data processing provided by Leap Motion, and the implemented approach is through the usage of a control algorithm which makes use of *Base Learners*. The Base Learners belong to the *Supervised Learning* field, which is a *Learning Paradigm* of the *Computing Methodology* called *Machine Learning* according to the Association for Computing Machinery (ACM) classification [1].

With these ideas, we summarize the main objective of this project in the following question: Is it possible to develop software that recognizes hand gestures and controls certain actions for a mobile robot in a real-time manner and incorporates a learning approach?

This question is worth to be answered from (at least) two points of view:

- **Control:** To control a device in just one way may be a challenge or even uncomfortable for some people. Therefore, to investigate alternative approaches for common actions becomes an important field of research.

- **Gesture recognition:** This research field still has plenty of room for improvements and technical proposals with the development of the latest sensors and composed devices.

This project makes use of a proposed approach that allows the incorporation of **static** classifiers into **temporal** gestures recognition, we call it: the *temporal* approach. Afterwards, in order to validate the response of the recognizer algorithms, their performance and accuracy responses are compared in the classical way, by making use of the classic Test Set, and in a *modified* version of the binary confusion matrix. This allowed us to extend its comparison usage from the *binary* to the *multiclass* classifiers.

1.4 Contributions

- A *Real-Time Control System* for the command of a *Quadrotor Unmanned Aerial Vehicle*, which retrieves the orders from the static and dynamic hand gestures recognition. This will be obtained as the outcome of the control algorithm under the philosophy of the *Ensemble Methods* of base *Supervised Learning* algorithms. The complete system runs over the Robot Operating System.
- A *temporal* approach that enables the usage of static classifiers into dynamic gestures recognition.
- A proposal of the accuracy evaluation based on *live* data.

1.5 Thesis organization

The rest of the document is organized as follows: Chapter 2 contains an overview of hand gesture recognition systems, some similar projects around the State of Art and the theoretical framework presents briefly the tools to understand in a quick glance the world of Pattern Recognition and the Machine Learning techniques that it inherits, which are key to comprehend the algorithms cores and the information displayed in following chapters. Chapter 3 displays the complete system design, unwrapping each subsystem into understandable fragments: the classifiers and the control stage; the *temporal* approach is depicted and the ROS implementation is exhibited. Chapter 4 holds the experiment design, the tests and results where the selection of the most suitable base learners is made; the static and live evaluations are explained along with the introduction of the modified confusion matrixes. Finally, Chapter 5 is made up by the summary of the complete work, where some conclusions and ideas for further work are discussed as well.

Chapter 2

Hand Gesture Recognition

The human recognition and learning capabilities are, without any question, one of the greatest gifts that mankind has ever received. In pursue of transmitting this abilities to their own technological creations, researchers developed some decades ago the Machine Learning and the Pattern Recognition research fields. There is still a long discussion among their bookish, whether which branch was first; what it is known for sure, is that both of them support each other through its principle sharing. Hand Gesture Recognition is a branch of Pattern Recognition and since this is the mechanism that this project counts with, it is wise to take a brief travel to review some important definitions and tools.

This Chapter starts by introducing the components of a Human Machine Interface, explaining what a Real-Time System is and what it takes to control a quadrotor Unmanned Aerial Vehicle (UAV) in a general overview. Following section holds a review of the State of Art around similar developments, introducing related devices and applications, while in next section a proper induction to the Pattern Recognition theory is presented. Since Pattern Recognition can not exist without the Machine Learning tools, a section of their fundamentals is presented. This project makes use of the Ensemble Methodology of Supervised Learning algorithms, hence, a section is devoted to briefly present the working principle of some of them (emphasizing the algorithms to be compared in Chapter 4) followed by a comparison chart where their main characteristics are faced. Finally, section 2.7 contains the list of algorithms selected for the evaluation and the reasons why they were chosen.

2.1 Human Machine Interface

Human Machine Interface (HMI) systems are nowadays a common solution for the problem of communicating technology with its users. An HMI may be defined as the channel that enables the machine control to an operator, therefore the main task of an HMI system resides in its ability of being self-evident to the user, i.e. a well designed HMI system will be the most self-explanatory possible, or will require the less possible training for the user. An HMI system is judged by its usability, which includes how easy is to learn its manipulation as well as how productive its user may be [2].

The design of such a system must take in account several factors such as safety, effectiveness, consistency and intuition. And from these four features, the *intuition* may be the fuzziest and hardest-to-define term. Insight of this, this project emphasizes this factor, and since humans start communicating with gestures prior than words, the project uses hand gestures as input

commands for the system.

In [3] a series of guidelines is provided in order to design an ideal HMI; among them, the key to a successful HMI system is to get to *know the operator*. For any user along the range from *intuitive* to *expert*, interface ergonomic considerations should be taken in account. And this is where Hand Gestures Recognition come to play an important role. But let us get there little by little.

2.1.1 Real-Time Control Systems

It is important to emphasize what *Real-Time* in the Computer Sciences means according to the IEEE Computers Society Technical Committee on Real-Time Systems:

Real-Time System: is a computing system whose correct behavior depends not only on the value of the computation but also on the time at which outputs are produced [4] .

Therefore, a Real-Time system describes hardware and/or software systems subject to a “real-time constraint” [5]. This does not necessarily means “fast”, but **responsive** in terms of a time need. This approach is widely used in control applications and a good reason for this is the following: in a closed loop controlled system, one of the main goals is to reduce the *feedback error* as soon as possible; hence, the response time is one of the most important error collaborators.

Figure 2.1 lets us think about it in this way: when the system is fed with a certain *input*, the system reacts to this signal in such a way that produces an *output* (the controlled variable). Right after the output is generated, it is retrieved as a *feedback signal*, which allows the *error signal* calculation: the difference between the reference input and the feedback signal. Once the error is known, the *controller* will behave in such a way that next time it retrieves a decreased error. It is then logical to think that when the complete system is coordinated and there are no big time loses or significant *delays*, the controller is able to modify its behavior in such a way that next time it gets the signal error, it corresponds to the commanded behavior and not to other one that could have been *delayed* in the path. In such an ideal system, the controller is able to perform its control action trusting completely in the system and its behavior.

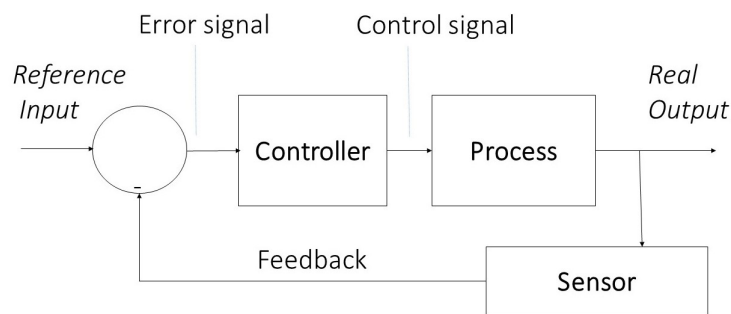


FIGURE 2.1: A Diagram of a Closed Loop Control System.

Real-Time Control Systems are subject to tight time windows: to retrieve data, process it, and update the station. It is easy to notice that if this time window is disturbed, the stability of the system is degraded and may be compromised. Nowadays these systems are found almost in every aspect of human life, since the fundamental components of the CERN's Large Hadron Collider, until the breaking mechanism of a regular car. Its features and advantages make these systems very popular and widely spread. Naturally the Robotics field is not apathetic to this and in response, this scheme is maybe the most used for control applications. Insight of this, let us talk about its application in one of the trending robotics applications: the Unmanned Aerial Vehicles.

2.1.2 Quadrotor UAV Control

If the Robotics of this century could be summarized into one word, possibly that would be the *drone*. This is a fancy word which makes reference to the Unmanned Aerial Vehicles (UAVs), a vast category within the Mobile Robots.

The UAVs are only a member of an Unmanned Aerial System (UAS), being that an UAS consists of one or more UAVs, a communication link, a control station and the ground crew (expert persons who control the complete UAS). Its growing importance and interest are owed to the exponential popularity that the UAVs have gained due to their wide applications field and the fact that the technological cost of its assembly pieces has decreased.

This situation leaded to a wide expansion of the UAVs devices (nowadays practically every mobile robot in the air is named an UAV). Perhaps the most widespread UAV is the *multiple rotating wing* style, where the UAV has from four to eight propellers. The simplest version, which counts with just four rotors, is popularly called the *quadrotor* style, quadrotor UAV or simply *quadcopter* (Figure 2.2). The four light motors hold the UAV in the air, providing the capability of keeping the device statically in the air.



FIGURE 2.2: A quadrotor UAV, Parrot brand.

The quadricopters have special perks since their design allows them to take off and land in a vertical manner, static flying capability, reduced dimensions and an interesting control mechanism. This last feature is better explained as follows:

The quadrotor UAV uses the *pressure difference* as uplift principle, which consists in the creation of a vertical force by an air flow and a counterturned torque around the UAV at the same time. The combined action of the rotors allows the movement of the UAV and because it needs

only four (rotor) inputs to control six individual outputs, this robot style is named a *disengaged controlled device*.

The six individual outputs to be commanded allow this quadcopter model to be a *Holonomic Mobile Robot* since it does not require a reconfiguration in order to follow a different direction, i.e. the current movement direction may be changed immediate and independently, without needing of any extra configuration adjustment beyond the specific one for the new desired direction. Figure 2.3 shows the quadrotor UAV attitude axes.

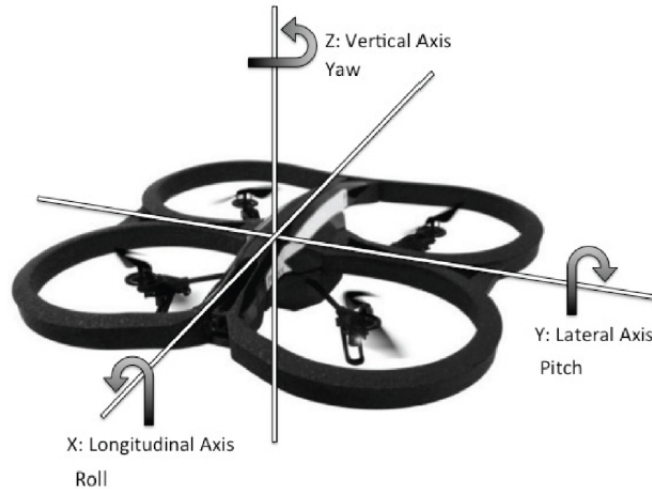


FIGURE 2.3: Quadcopter attitude axes.

2.2 Hand Gesture Recognition Systems

If humans learn to express themselves through gestures prior than through words, does not that make us more experienced using that channel? This question has addressed some research efforts into the conjunction of real time system control and its commandment through gestures. Being precise, such a system requires an interface, through which the natural human gestures are translated into a tag that the control system understands. This translation is the primary job of a *Gesture Recognizer*.

A *Gesture Recognizer* is a system that, making use of hardware and/or software, retrieves an input (may be a video camera frame), pre-process it, and then analyzes the incoming information in such a way that it is capable to deliver a class or label that corresponds to the *recognized gesture*. The mechanism that allows this *deliverance, labeling or categorization* is normally an *algorithm*. Of course, there are plenty of efforts in providing an algorithm that performs this categorization as best as possible, however these proposals are affected by the circumstances of the problem itself.

Depending on the application, the recognition task may have or not time constrains and there must be a trade-off between the time it takes to recognize a certain pattern, and the rate

of the retrieval device. These are very challenging jobs, and being more precise, this is the challenge that Real-Time Control Systems face when they incorporate Gesture Recognizers.

Initial recognizer proposals employed Image Processing tools in its early stages. In these first experiences, the challenge was the usage of 2D models: Since the data is normally gathered by a regular cameras, there was no information about the depth and therefore the the algorithms performed taking into account only 2D models (*Appearance* based). The main problems of this approach is the lack of some others features (like the one that the third dimension would provide). This led to undesired limitations, such as the fact that the algorithms' usage lied right after the image processing, process which would not deliver information straight to the algorithms and consequently did not perform as good as expected. In response to this, some other model estimation variations were proposed. A good example of this is the one presented by researchers in the Nara Institute of Science and Technology in Japan, who in 2001 published a method to estimate the translation of hand 2D information into a 3D model using silhouettes[6].

Figure 2.4 plots in a general diagram, the interaction between the components in a real time system controlled through a hand gesture recognizer. The HMI provides only the connection between the user (and his hand gestures) and the controlled system and must be compliant with the real time constrain, otherwise it could lead to a bottleneck that, even when the controller and the process work in real time manner, would introduce undesired latency and therefore a delay.

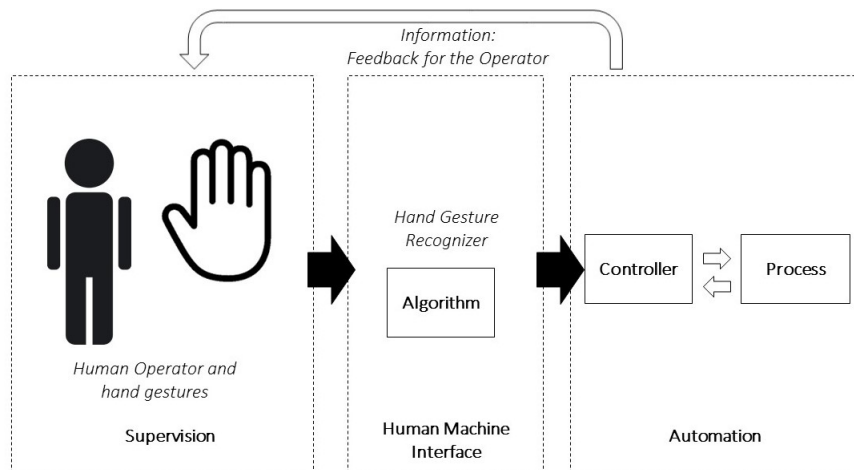


FIGURE 2.4: Diagram of a Hand Gesture Recognizer used in a closed loop control system .

In the pursuit of such a recognizer, an approach is to combine Supervised Learning Algorithms with the HMI development field. Being the HMI natural born's place the industry [7], the HMI applications rapidly grew into more formal and documented examples[8]. Given the fact that a decisive element of the HMI and the recognition process lays on the algorithm itself, several techniques have been developed in order to address this problem on its own. This leads us now to the next big topic: Hand Gesture Recognition.

2.3 Hand Gesture Recognition

The hand gesture recognition problem has been handled traditionally, by conventional cameras which look at the hand(s) from a front view. This is called the *Appearance Based* approach and uses different algorithms in order to recognize the gesture. Even when they still have an ample error range in the recognition task, it is still a popular research field. Example of this is shown in [9], where the authors present an interesting study about the sign language recognition, emphasizing the usage of the three common approaches: Appearance based, Hand Shape and 3D hand models.

The hardware usage is very creative and may vary in styles, for instance, in [10], the author uses a regular stereo webcam to gather the information and then merge the images in order to get a descriptive frame.

Another very interesting, although classical, approach is the creation of the model to be recognized through fixed hardware: gloves full of sensors and gloves with markers on strategic points of the hand [11], this is better expressed in Figure 2.5.

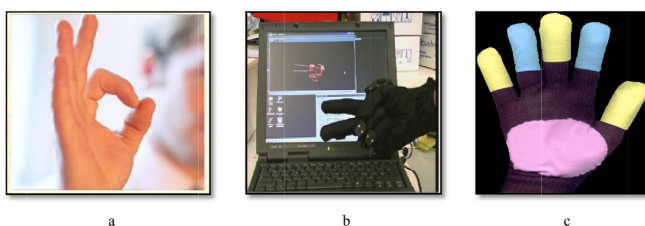


FIGURE 2.5: Examples of approaches: a) vision based b) data-glove based c) colored glove

In [12], an assistive glove is made up by networked sensors which create specific data stream accordingly to the current hand position. The low voltage signals are then provided to the software system that decodes this information into recognizable words and generates a message. In [13], the Cracow University of Technology developed a similar tool (although with much less sensors). In this project, the sensed information is provided to a pre-processor and delivered to three Machine Learning Classifiers.

2.3.1 Recognition Devices

With the introduction of hardware capable of providing more useful gesture information, in lately years a new approach has been developed and reinforced: *the 3D Hand Skeleton Model*, which is a subfield of the more general 3D model based [14]. The Skeleton Model basically makes use of different techniques and sensors in order to calculate the hands' **depth** information. Some examples of hardware devices capable of providing this kind of model are binocular cameras, Kinect and Leap Motion. Since their public availability, many researchers have been attracted to their usage: In 2014, the University of Padova published an application and comparison of the Kinect and Leap Motion devices for the same application[15]. Their results express that both devices have good response despite their physical differences and in conjunction, both of them can achieve a higher hand recognition accuracy. While in 2013 and 2014, Frank

Weichert conducted a series of experiments ([16], [17]) demonstrating the specific capabilities of Leap Motion and Kinect.

Despite of the great perks that the Skeleton Model provides, the inclusion of the following two facts, turns this approach into a very challenging process:

- The multiple hands' degrees of freedom (DOF).
- The hand is an articulated deformable object.

Being the Leap Motion Controller a device which makes use of cameras that look from a down-to-up view, it requires a different approach in order to recognize the gestures. Considering this, the Leap Motion has been used mainly for Computer interaction and for Desktop software control. Since it is a recent device (launched in 2013), its development capabilities still have plenty of room and with this, the recognition implementation tools is a very exploitable field. In combination with this, the proposal of a hardware control makes the gesture recognizer a very attracting and promising research sphere which is enhanced with the addition of a set of effective and efficient recognition algorithms [18].

Nowadays, the Leap Motion APP Store counts with around 200 applications developed for Windows and OS X ranking in categories such as games, virtual reality, desktop environment manipulation, education, music, among some others [19]. This kind of tools attracts the attention of programmers all over the world who, inspired by its flexibility and open SDK write programming codes for hobby, educational and research purposes. Up to now, to the best knowledge of the author, there exist a couple of applications that making use of the Leap Motion, send commands in order to control a quadcopter, nevertheless they don't use a recognizer machine. This leads to the unstable performance that the results show [20].

In this spirit, some other implementation tools have been developed, such as the MATLAB Image Processing Toolbox or its conjunction with the OpenCV Library to implement recognizing systems, despite, they only use 2D images.

2.3.2 Other Tools and Applications

The majority of the tools used in Hand Gesture Recognition are the heritage algorithms of the so-called Artificial Intelligence (better explained in Section 2).

The techniques for dynamic gestures recognition popularly are the Hidden Markov Model, Dynamic Time Warping and Artificial Neural Networks, while for static postures the offer is much larger: Kalman Filter based hand tracking, Condensation Algorithms (based in the principle of Particle Filtering), Finite State Machine, Bayesian Methods, Support Vector Machine, Time Delay Neural Network, among many others [14].

Proof of its wide application are for instance, the project published by the Nanjing University [21] in which, through the Hidden Markov Model, an incremental learning model is implemented and a HCI is enabled. On its side, the NVIDIA corporation developed a recognizer making use of Convolutional Neural Networks [22] in order to get a recognition accuracy

of near 77%. In 2012, the Xian Jiaotong University published a project [23] in which through the Adaboost algorithm, a system is capable to achieve a trustable gesture recognition.

Once the recognition of the gestures is achieved, the application boundaries is merely left to the imagination limits. Proof of this are projects such as the one published by the Ruhr West University [24], in which, through a Multi Layer Perceptron, a learning mechanism is trained for automotive applications. Multimedia applications is also targeted and one attractive employment is the one generated by the Karlsruhe University [25], where the sound and music control of an application is left on charge of an ANN based hand gesture recognizer.

Special attention to this field has attracted even the development of complete hardware-SDK combos, such as the one introduced by Intel with *Intel Real Sense SDK* [26] which is an integrated tool capable of hand/finger tracking, facial analysis, speech recognition, augmented reality and background segmentation, and consists of a specific Intel camera and its SDK. Despite of its interesting capabilities, this tool is a closed-source software that works only with its specific devices and runs only over Windows.

When it comes to robot manipulation, the hand gestures play also a substantial role and although some of the developments have been targeted to trigger behaviours, the recognition may be focused on the robot's movement control. In 2012, the National Taiwan University published a project [27] in which the actions of a service robot were handled by the recognition of the person hand gestures; its task was mainly to provide the cared person with healthing actions. On the other side, the Beijing Institute of Technology achieved to control a virtual robot in 2014 [28] with Neurofuzzy techniques and a SVM classifier.

2.4 Pattern Recognition

We were born into a mystery, one that has haunted us since the very beginning of the human memory. We awakened on this tiny world beneath a blanket of stars like an abandoned baby left in a doorstep, without a note to explain where we came from, who we are or why the world is the way it is. With no idea how to end our lack of knowledge, we had to figure it all out for ourselves. Best thing we had going for us was our intelligence, specially our gift for finding non-apparent clues in clusters of information, or deducting conclusions where there are not straight rules. This is our *pattern recognition* skill, sharpened over eons of evolution. The first humans who were good at spotting prey and predator, telling poisonous plants from the nourishing ones, they had a better chance to live and reproduce, they survived and passed on those genes for pattern recognition with its obvious advantages.

Across the planet, almost every culture looked at the same stars and found different figures there, Figure 2.6 plots this idea. We can see how astonishing our disorder perception and imagination may be. In those early days, we used this gift for recognizing patterns in nature to read the messages written in the stars, they told our forefathers and mothers when the migratory rains and when the rains and the cold would come, or when they would cease for a time. When they observed the right connection between the motions of the stars and the seasonal cycles of life on Earth, they concluded, naturally, that what happens up there must be directed at us down here. And it makes sense, isn't it?

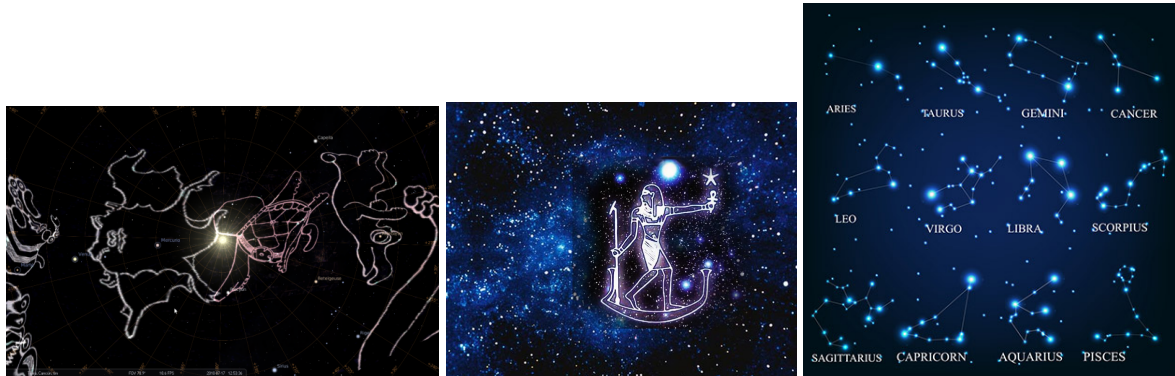


FIGURE 2.6: A sample of the astrology across the mayan, egyptian and greek cultures.

However, the human talent for pattern recognition is a two-edged sword. We are specially good at finding patterns, even when they are not really there, something known as *false pattern recognition* (Figure 2.7). We hunger for significance, for signs that our personal existence is of special meaning to the universe, and this motivation has led us to discoveries of undescrptible value. Let us recall for instance, those ages in which by looking at the stars, Edmond Halley predicted the Halley comet return every 76 years. It is certainly true that our guest has not let us down ever since.



FIGURE 2.7: The precolombian cultures shared the myth referring a rabbit in the moon.

The easiness that we have in order to recognize the world (faces, voices or actions just to mention some examples) is a proof of the astoundingly intricated processes that take place when we need to recognize patterns. Even though, the knowledge about our cognitive process is still small, we know that our pattern recognition is not a straight-forward process. Despite of this lack of awareness, we have tried to transfer this ability to things over which we have power: the machines. In general terms, the artificial pattern recognition feature is nowadays used by

several technologies at very different levels of engagement; nonetheless there still exist some challenging problems that are much more complex than others. This is the case of the *voice* and *visual* pattern recognition (just to mention a couple them)[29], [30]; hence, Pattern Recognition is a branch of *Machine Learning* (Section 2.5).

When it comes to visual patterns, the task implies the selection of specific **features** that the image processing will focus on. The selection of these required *features* provides another degree of complexity, since the determination of the right number of them (keeping the major number of necessary features, ignoring redundant and irrelevant information), is another research field called *Feature Extraction and Selection*. From this research field, we have learned that **conforming features of a model must be robust, relatively insensitive to noise and as most explanatory as possible**.

2.4.1 Recognition Process

The core of a Pattern Recognition process is executed mainly in the *Classifier* stage. This function is performed by certain algorithms, which making an analogy with the human process, receive a certain amount of information without an apparent structure and deduces from it, a hypothesis. The initial provided information is a set of *features*, the deduction process may imply learning or not, and the obtained hypothesis is called the *class*. Due to this last step, these algorithms are called *Classifiers*.

Although the classification task may be based on the structural relationships between various features (Features aggregation), rather than the features themselves, this second approach is the one that has gained popularity and may informally be explained as follows: the set of the selected features will conform a *model*, which may be understood as a mathematical description of the features.

The holly grail in pattern classification is to hypothesize the class number of data clusters, process the sensed data to eliminate noise (not due to the models) and for any new sensed pattern, choose the model that corresponds best. A comprehensive view of this process is shown in Figure 2.8. Incoming data is perceived through a certain port, it may be a virtual *Transducer* or a physical *Sensor*. It delivers a certain data model to a *Preprocessor* stage, which is responsible to adequate the raw data to a more suitable model taking care of not losing any important information. Regularly, this second data model is transferred to a *Segmentation* proceeding. This is a supporting step, responsible of isolating the information into more concise units, and although is not mandatory for every classification system, it is of great help. Following step is to distribute the data model to the *Feature Extraction* phase. This stage is critical, since the overall system performance is bonded to its proper fulfillment; in here, the most descriptive and robust features of the data model are arranged into a structure suitable to the algorithm understanding. Last step is naturally the execution of the *Classifier* algorithm, over the features vector, providing a final decision or **class**.

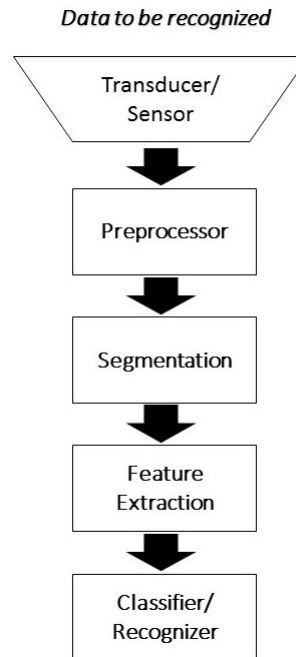


FIGURE 2.8: A simple classification process.

The complete Pattern Recognition process may be listed, but not limited, into the following steps:

1. Data Preprocessing.
2. Dimension Reduction and Feature Extraction.
3. Classification making use of Machine Learning algorithms.
4. Iterative algorithm Training and Testing.
5. Comparison across Classifiers and Feature Subsets.

While **Data Preprocessing**, **Dimension Reduction** and **Feature Extraction** are executed in the same way that Figure 2.8 displayed, following phases are separated for good reason: Classification algorithms may be featured with a *learning* approach or not. Among the many differences between them, the underlying importance of the learning algorithms, is that their output obeys to the way they are **trained**, whereas a non-learning classifier delivers an output based on the way it is **programmed**.

The approach used in a learning classifier differs, since different classification techniques are useful depending on different factors, such as the learning style (supervised or unsupervised), feature selection (how many and what types of features they manage), or the type of candidate models themselves. In general, the algorithms that obey to this behaviour belong to the Machine Learning field and therefore they inherit the name and its tools. Most of the times, it is desirable that the classifier provides a real-time response, however this may be difficult

to achieve, since several experiments have shown that the decisions based on overly complex models, often lead to lower accuracy and time response from the algorithm [31], [32].

As result of a learning classifier election, an iterative training and testing phase is required over the algorithm. This is mandatory because the classifier performance lies explicitly in the training procedure. Figure 2.9[33] shows the workflow of a supervised learning classifier implementation divided in four main steps: sample retrieval, configuration of the learning problem and algorithm, learning phase and the *traditional* static evaluation. The samples retrieval phase consists of the Training Dataset gathering in order to feed the following stage. Configuration of the problem is where it is analyzed and some proposals are introduced, regularly as features selections or creations followed by their employee in the elected algorithm (along with the algorithms parameters tuning). Once a good proposal is retrieved, the algorithm gets trained the consequent static evaluation consists of computing the algorithm over a test dataset, providing decision or *class*. This output is analyzed commonly through binary confusion matrixes.

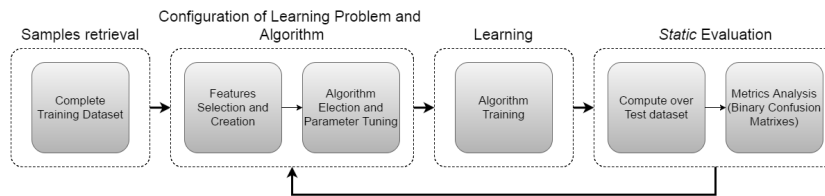


FIGURE 2.9: Workflow of a conventional pattern recognition implementation with a supervised learning algorithm.

Finally, a certain algorithm may, or may not, be the best choice for the specific problem. This is known by the study of the algorithm guidelines and through their performance comparisons; therefore, a test comparison across classifiers is a very useful resource to count with. The same applies when selecting the best features for the feature selection stage. The way the algorithms answer to different features subsets is directly related to its recognition ability.

2.4.2 Gesture Recognition

Gesture Recognition is a remarkable subfield of the Pattern Recognition domain and its importance lies on the variety of applications that a human **gesture** and its interpretation may provide, although this implies one of the main problems of this dilemma: generally there are *many-to-one* mappings from meanings to gestures and vice versa; hence, gestures are *ambiguous* and *incompletely specified*.

A simple and initial classification of human gestures splits them up in two categories: **Static** and **Dynamic**:

- **Static gestures** are non-temporal body positions which express something.
- **Dynamic gestures** refer to a body movement with a related significance.

Among both of them, the dynamic gestures recognition requires special attention due to its necessary temporal segmentation. Dynamic gestures make use of specifications of the start and end points in terms of the movements frames; dynamic frames are often tracked to generate suitable recognitions. This problem is better known as “Gesture Spotting” and could be described as the situation where, when the hand motion changes from each gesture to another, an intermediate movement occurs and most of the times, this *transitional* movement is segmented and matched with some referent pattern. This is obviously a disadvantage and needs to be removed from the model to process. Hence, **dynamic classifiers** are much more complex than **static classifiers**.

2.5 Machine Learning Fundamentals

Machine Learning is the field of study that according to Arthur Samuel “gives computers the ability to learn without being explicitly programmed”[34]. Even when this definition provides the reader with the spirit of what Machine Learning does, there is another (more formal) definition provided by Tom Mitchell: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T , as measured by P , improves with experience E ” [35].

Machine Learning is then, the subfield of the Computing Sciences that develops mechanisms that *adapt* their actions so that these actions improve their *accuracy*, being that **accuracy** is defined as a *measure of how well the selected decision reflects the right decision*. The Machine Learning mechanisms are based in the making of a **model**, which is constructed from examples (experiences). This step is called **Training** and it allows the model to deliver data-driven answers to a certain task (in the *On-line* stage), instead of selecting explicitly a programmed sequence.

2.5.1 Preliminaries

In the simplest approach, a Machine Learning algorithm must be first *trained* with a training dataset, *tested* with a test dataset and finally *on-line executed* with live data.

The nature of the datasets categorize the Learning style of the Machine Learning algorithm, nonetheless, we can make use of a simple example in order to clarify some terms and expressions:

- **Feature, Attribute or Covariate x_i** : are characteristic properties to be measured of a certain object. The feature value should be similar for objects in a particular *class*, and different from the values for objects in another *class*; this is known as distance intra- and inter-class. We can say, for instance, if the object to be classified are human persons, the features could be weight, height, age and gender.
- **Feature vector x** : is the D -dimensional vector that contains the specific *features* for an object. Taking the same example of the human, the feature vector for a certain person with the features mentioned could be : [60kg, 160cm, 28years, male]. The Feature vector are

the incoming data to the algorithm. For n features, its feature vector will be represented by:

$$\mathbf{x} = [x_1, x_2, \dots, x_n] \quad (2.1)$$

- **Class, Target, Label, or Category c_j :** It is the expected category or nominal value, depending on the desired output.
- **Classes or targets vector \mathbf{c} :** will contain j available classes and is denoted by:

$$\mathbf{c} = [c_1, c_2, \dots, c_m] \quad (2.2)$$

The election of the number of features is an important step, since a bad choice could lead the process to suffer what is called the *Curse of Dimensionality*. This is a term used to express the problematic that arises when a high number of features is chosen. It is natural to wonder *why would we want to use many features?*, and the reason is this: a Machine Learning algorithm requires many features in order to describe and generalize the process sufficiently well. Since we have already mentioned, the algorithms are fed mostly through the features, and if we aspire to one that works fine, we would think that as long as we provide more features, its performance will increase. And that is where the curse happens: as the number of input dimensions (the features) gets larger, the algorithm needs more data samples to be able to generalize the recognition. Most of the times, grabbing so many samples and handling a huge amount of data is a remarkable problem. This problem is well explained in [36] and what matters the most, is that the features amount should be selected in such a way that the algorithm has enough information in order to do its job.

2.5.2 Types of Learning

Depending on the nature of the available data to a learning system, the Machine Learning tasks are categorized mainly in the following categories:

- **Supervised learning:** The algorithm is trained with a set of data composed by the feature vector and the corresponding target class. Based on the training set, the algorithm generalizes a model to categorize other possible inputs.
- **Unsupervised learning:** In this case, the training set does not provide target classes to each feature vector, therefore the algorithm tries to identify patterns and similarities between the incoming samples in a way that the samples with similar features are categorized under the same class. Its statistical approach is known as *density estimation*
- **Reinforcement learning:** This approach focuses on the output as a whole. For this algorithms, what is important is a *sequence of actions* that together make up a good policy. A sequence on its own is not *right or wrong*, but the policy (sequence of correct actions to reach a goal) as a whole. It is somewhere between supervised and unsupervised learning: The algorithm gets told when the answer is wrong, but does not get told how to correct it. It has to explore and try out different possibilities until it works out how to get the answer right.
- **Evolutionary learning:** This style is taken from the biological evolution ideas, in which an organism gets adapted by improving their survival rates and chance of having descendants in their environment.

It can be said that Machine Learning is commonly used to solve the following two main problems (among some others such as Segmentation, Clustering, Time series Prediction, Knowledge Discovery, etc):

- **Classification:** This task delivers *Discrete Outputs* and may be understood as the task where a recognition is required for a feature vector in such a way that the response may *take its time*.
- **Regression:** This solution provides *Continuous Outputs* and is also known as *Prediction*. It owes this name to the fact that the correspondence between feature vector and classes must be performed continuously since the system requires a labeling each time frame.

Even when they are displayed as two different tasks, it becomes the same classification problem: In the field, there have been several experiments that use **continuous classification** algorithms in order to solve Prediction tasks, and although the algorithms may be diversified in order to attend one or another problem, it is important to keep this difference in mind.

Therefore, this work makes use of a **Classification** approach, using **Supervised Learning**, since the task is to recognize classes (or targets) and these algorithms have shown great advantages and improved responses as for what real time concerns. In Section 2.5 these algorithms are discussed in greater detail and why they are suitable to this system. Figure 2.10 shows the general process of a supervised learning system, it is composed by two phases that happen **offline** and **online**. The Offline Phase is called **Training** because it is where the algorithm develops a notion model of the system accordingly to what it is taught through the examples. The learning algorithm is fed with examples, known as the Training Dataset, and using diverse procedures, it is able to deliver a model that expresses the relations and knowledge that could retrieve from the examples. The delivered model is then used by the following phase, which happens to be the **Classifier** itself. Its task is to receive the data that need a classification, and making use of the model delivered by the learning algorithm, does its job i.e. makes a classification decision over the incoming data and assigns a **target class** to the processed data.

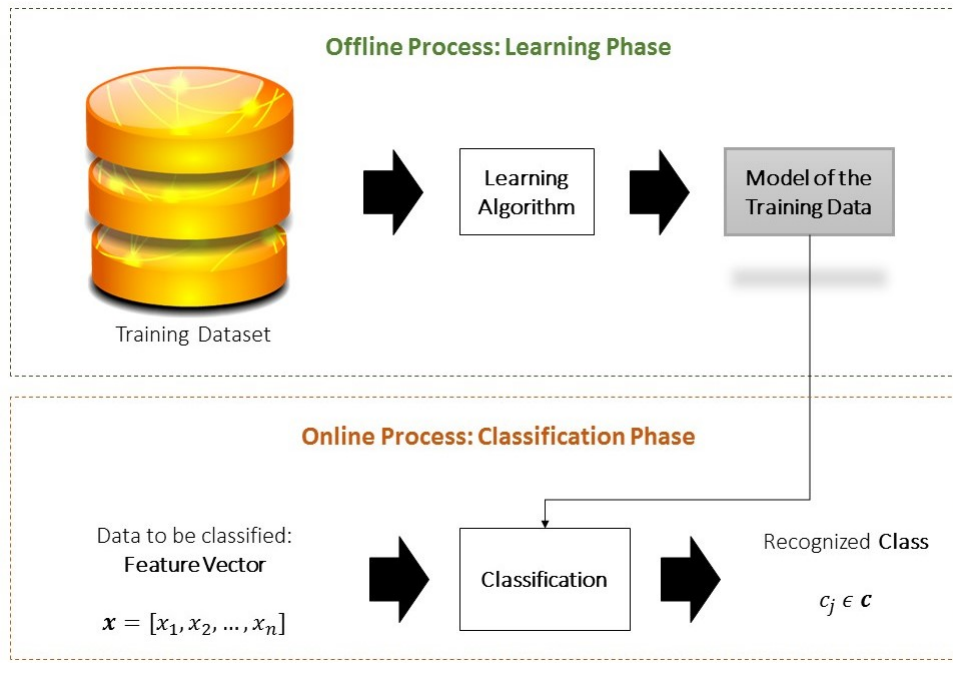


FIGURE 2.10: Supervised Learning.

2.5.3 Stages of the Machine Learning Process

The application of a supervised learning algorithm to a problem is depicted in Figure 2.11. As it can be seen, it consists of two phases separated by the dataflow nature: in the **offline** process, the algorithm is provided with a previously recorded Training Dataset obtaining a model of the training data as result. Afterwards, this model is employed by the same classifier algorithm in order to process the incoming data (the **Feature Vector**) and provide its recognized **Class** as output.

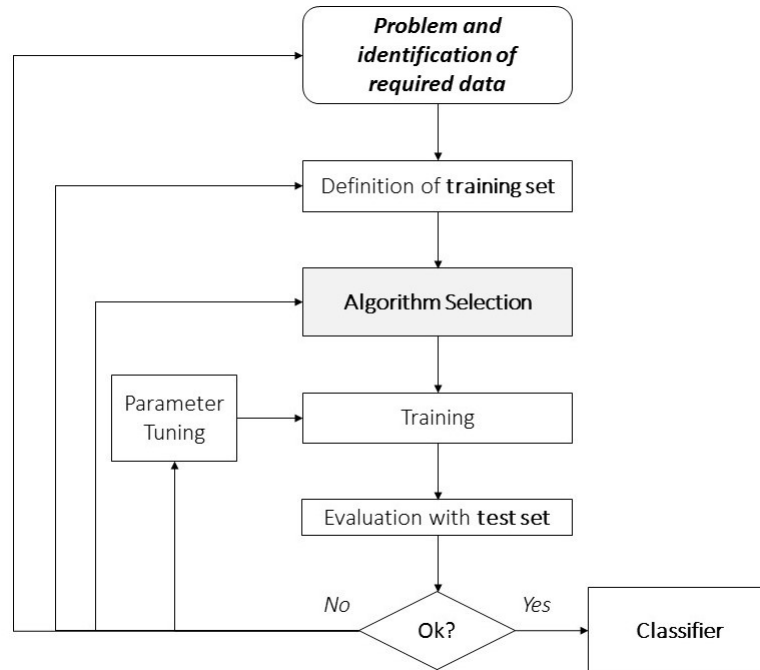


FIGURE 2.11: Overview of applying a Supervised Learning Algorithm to a Problem.

It becomes self evident then, that the choice of the specific learning algorithm is a critical step that directly affects the performance of the classification. This problem is not uncommon and is expressed by the **No Free Lunch Theorem** [37], by Wolpert and Macready in 1997, which relates that *there is no ideal solution to the classification problem*. Therefore, the process of solving a problem is: select an algorithm, apply it to a training set, evaluate it, correct if necessary and then adopt it to for the occasion. This evaluation is most often based on its *prediction accuracy*, however, since the elected algorithm will be a component of the HMI, a very important feature will be its *time response*.

Training, Testing and Validating sets

Retaking the Figure 2.10, it is easy to see that the Training Data may be expressed by the vector:

$$X = [[\mathbf{x}_1, \mathbf{t}_1], [\mathbf{x}_2, \mathbf{t}_2], \dots, [\mathbf{x}_M, \mathbf{t}_M]]^T \quad (2.3)$$

However, the unique real way to know how good an algorithm is to compare its classifications against *known* target classes, and this is how the training stage happens in Supervised Learning. Therefore, in Supervised Learning two data sets are required (at least): the *training* and the *validation* datasets. There is a third set, called the *testing* dataset, and its importance lies in the need of knowing how well the algorithm generalized the classification task for the samples that were not seen (or shown) in the training dataset. This test dataset will be used only to decide how well the algorithm learned and not for modifying any weight or parameter.

Therefore, we could list the three datasets as:

- **Training Data:** To actually train the algorithm.
- **Testing Data:** To keep track of how well it is doing as it learns (to check if the model might be overtrained or undertrained)
- **Validation Data:** To generate the final result.

The Peaking Phenomenon

As it was expressed in 2.5.2, the learning algorithm deduces the model from the provided Training Dataset. The common sense would tell that if we provide *as many examples as possible*, the algorithm would be trained *at its best* and therefore the model would be *nearly perfect*. This is almost true but counterproductive and is known as *Overtraining* the algorithm, or *Overfitting*. The negative effect of this may be understood by thinking that (besides of getting the *curse of dimensionality*) when the algorithm has too many examples, it learns exactly what those examples are, and start to lose *Generality*. On the other side, if the algorithm is provided with very few examples, the generated model could provide decisions that would not obey to a certain pattern and it would not be so different from the decision provided by a coin toast; this is known as *Underfitting*.

This notion may be seen easily in a two classes Classifier as Figure 2.12 shows. Suppose that the algorithm is fed with Training Datasets that deliver two classes (the triangle and the circle class). Since it would be a two-dimensional problem, a simple line marks the limit between the classes. This limit is called the **Decision Boundary** and it is what mainly composes the model delivered by the learning algorithm. Now, if the algorithm is fed with *too many* training data, it will be **overtrained** and deliver the *tight* decision boundary shown in a). This boundary is intricate, generates a complicated mathematical expression and more importantly, **it loses generality** in great manner: during the on-line process, is highly probable that an incoming data would be mistakenly categorized if it is near to the training sample that is *almost* inside the other class field. On the other side, if the algorithm does not receive *enough* training, it may generate a very loose and simple boundary that simplifies the decision way too much. Finally, the third example shows that even when a model is not either Over- or Underfitted, there could be many decision boundaries. This is influenced by the training set and the way the algorithm works.

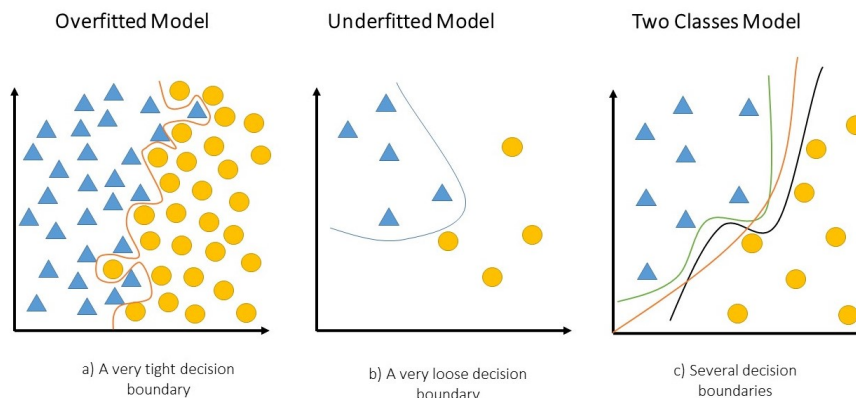


FIGURE 2.12: The Peaking Problem.

As a natural consequence, it is a trade-off between how much training samples the algorithm requires in order to provide a **generalized model**, since the model will trust on the training samples (on the learning phase) and will decide based on how well it **generalizes** the classification problem.

2.5.4 Validation Methods

Insight of what has been described it is required a measure that allows the selection of the best way to proceed taking in account several factors such as the way the whole training dataset is partitioned into the three subsets, the accuracy of the algorithm, the decision boundary form, and of course, the response of the classifier for the system requirements where it will be used.

Test Sets and Errors

As explained earlier, three datasets are required in order to train and confirm if the algorithm is working as expected. It is obvious that the three datasets should be the same nature, i.e. they must have a feature vector and its corresponding class, and normally, having a great amount of such samples is difficult. The way these sets are selected and splitted affects the way the algorithm performs; normally this distribution election is up to the developer, however a good proportion would be 50-60% for training, 25-20% for testing and 25-20% for validation.

We have talked about the error in a model, however we have not defined what an *error* would be, so let us start on that way. An **error** in a classification will happen when the classifier assigns a wrong target class to a certain input sample.

Even when this single term would be useful for most of the classifiers, there are some other tool measures that provide better knowledge on how good a classifier is performing.

The Confusion Matrix

This chart is a tool developed for two-class classifiers (or binary classifiers). For the sake of this implementation, it is required to extend its usage to multi-class classifiers, and in order to understand the basis, the simple binary example is explained:

It consists of a square matrix made up by all the possible target classes. The top of the table contains the predicted outputs (result of classification) and the rows hold the expected output (real class). With this structure, the cell (i, j) contains the number of how many input patterns for class i were in the training set, but were classified as j by the algorithm.

In the example shown in Table 2.1, it is easy to perceive that everything in the **leading diagonal** is correct, while everything outside was miss-classified.

In the Section 4, these matrixes are modified in order to display the output for the multi-class recognizers in percentage values and analyse the classifiers response in a more useful way.

Real Class	Output		
	Class 1	Class 2	Class 3
Class 1	8	1	0
Class 2	2	6	1
Class 3	1	2	5

TABLE 2.1: Example of a confusion matrix

2.6 Supervised Learning Algorithms

The offer of Supervised Learning Algorithms is immense and diverse. Fortunately to us, since the emergence of this discipline, it has gained good attention and therefore several approaches have been proposed. Since this work makes use of a combination of them, it is wise to take a brief look to them. There is an voluminous literature covering the algorithms due to their popularity and extensive use and although most of them have additional subtleties and parameters beyond the mentioned here, they will be discussed just in a simple overview. If further interest, the reader is encouraged to consult [30] and [36] which are authorities in the topic along with the related articles published every now and then.

2.6.1 Bayesian algorithms

The bayesian algorithms imply a whole kind of algorithms of the probability approach which are based on the confidence that, beneath the training samples, there is a probability model. In general, given a number of hypotheses (or *a priori probabilities* with the training examples) the class assignation is performed turning them into *a posteriori probabilities* by selecting the hypothesis with highest likelihood (or probability of being a certain class). This notion is expressed in the Bayes Law (Equation 2.4) which express that the probability of likelihood of event A occurring given the observation of event B. In the algorithms usage, $P(A|B)$ would turn to be the *posteriori*, $P(B|A)$ the likelihood, $P(A)$ the priori and $P(B)$ the evidence.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (2.4)$$

Making use of the notation for the targets classes and the features vector, this Law would be rewritten as:

$$P(c_j|\mathbf{x}) = \frac{P(\mathbf{x}|c_j)P(c_j)}{\sum_{i=1}^m P(\mathbf{x}|c_i)P(c_i)} \quad (2.5)$$

and since \mathbf{x} is a vector with n features:

$$P(\mathbf{x}|c_j)P(c_j) = \prod_{i=1}^n P(\mathbf{x}_i|c_j)P(c_j) \quad (2.6)$$

Equation 2.5 predicts the likelihood of label c_j given the observation of the feature vector x .

This approach requires that the features are independent conditionally to the targets and the delivered model consists of boundaries determined by the probabilities density functions of the classes, which are determined by the probability distributions of the samples. Afterwards the probabilities density function, the class election depends on the class most similar to the input sample.

The **Adaptive Naive Bayes Classifier (ANBC)** [38] is an algorithm based on the Bayes' theory which essentially works by fitting an N-dimensional Gaussian distribution to each class during the training phase. The incoming data is labelled in the prediction phase by finding the gesture that results in the maximum likelihood value (given the new sensor data and each of the Gaussian distributions). Its implementation is convenient due to its prediction time complexity, short computational training time and low variance, which makes it highly recommendable to use in combination with some other classifier and approaches. This may be depicted in Figure 2.13

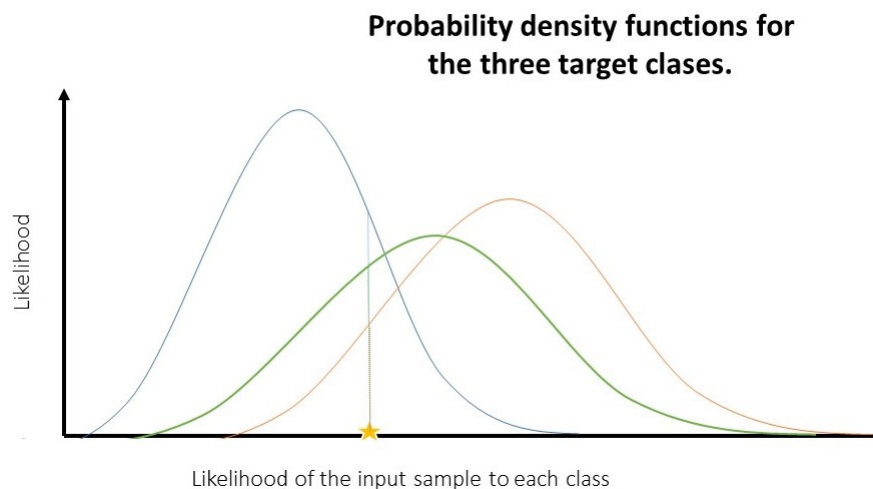


FIGURE 2.13: Example of the probability density functions for a three classes Naive Bayes classifier, the class with greatest likelihood is selected as the output class.

2.6.2 Decision Trees

It is a nonmetric approach that assigns classes using a top-down decision flowchart where each step is divided generally with a *if-else* statement over the input features. It may be understood as a decision graph that recursively partitions the dataset based on a certain feature. When such an algorithm is trained, the model is made up by a tree architecture where each node's decision is structured by the training data, or better said each branching level is a feature-value pair; with this, the lowest levels become the leaf nodes and are labelled with a class name; if a smaller model is required, the non-desired leaves may be *pruned away* in order to have a simpler and quicker model. This enables the option of express the model as a rule set or a decision tree.

2.6.3 K-Nearest Neighbors

The K-Nearest Neighbors is an instance-based method which delivers a target class using a majority vote of the k nearest training examples. k is the primary parameter of this algorithm and defines how *smooth* the categorization will be: if it takes less neighbors, the decision will be taken without much knowledge. The decision boundary between classes is defined using the Euclidean distance d (most of the times) which for an n -dimensional space and two vectors (\mathbf{a} and \mathbf{b}) is computed as:

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \dots + (a_n - b_n)^2} = \sqrt{\sum_{i=1}^n (a_i - b_i)^2} \quad (2.7)$$

This distance is measured from the received data to its k -neighbors, therefore it is composed of sections of straight lines and it can be said that the algorithm does not provide a *global* model but a local one built *on the fly* from the samples. Figure 2.14 displays the way it categorizes in a two-Classes case, if K is selected as 4, the new information will be categorized as the three closer neighbors say, on the other hand if K is larger, it will take in account more opinions from the neighbors.

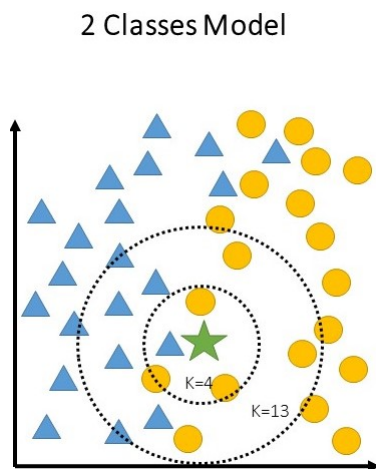


FIGURE 2.14: Example of a 2-Classes case, where KNN provides a different class according to the number of surrounding samples.

Its disadvantages could be named as: the large storage requirements, the sensibility to the choice of the similarity function (for comparing instances) and the lack of a principled way to choose k . This last problem may be better understood by taking a look to some scenarios where the k NN could misclassify a sample: If a query input is located near to a noisy region, the noisy instance could win the majority vote resulting in the incorrect classification (in this case a larger k could avoid the problem); or if the class defining region is very small, instances belonging to the surrounding classes to the small region could win the majority (a smaller k could solve this problem).

Its nature of instance-based provides an easy implementation since *no training is required* given the fact that the learning model is retrieved directly from the raw data. Due to its lack of training, it is also named as a *Lazy Learning* algorithm.

2.6.4 Minimum Distances

The Minimum Distances classifier is a simple algorithm which partitions the space into as many disjoint decision regions as number of known target classes. The equation that generates its decision boundaries is the Euclidean distance (Equation 2.7) between classes *means*. The prediction is based on the computation of the closest mean to the incoming information through its Euclidean distance. It may be easily depicted in a 2-Class case, as Figure 2.15 shows, where the decision boundary is defined by the focus to the hyperplane that is halfway between the two means of each data group and is orthogonal to the line connecting them.

2 Classes Model

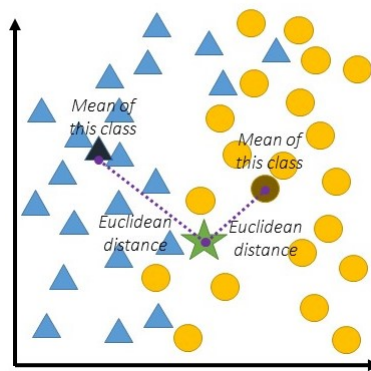


FIGURE 2.15: Example of a 2-Class Minimum Distances classifier.

As it can be seen, it is a very simple approach the one that it uses and enables a quick prediction (compared to KNN) although its reliability lies on the simplicity of the application use, the learning phase, and of course, the distance between the class data.

Some approaches use instead of the mean, a cluster principle that joins the training Samples. This variation sets then the number of clusters as the single parameter of this algorithm.

2.6.5 Support Vector Machine

It is a Kernel Method that performs its classification based on the distance from the instance (to be classified) until the hyperplane on a high-dimensional space.

This hyperplane transformation is called the **Kernel trick** and most of the times produces a non-linear model. It allows to create a decision boundary in a high-dimension space when this

boundary could not be possible in the features vector space. This idea is displayed in Figure 2.16.

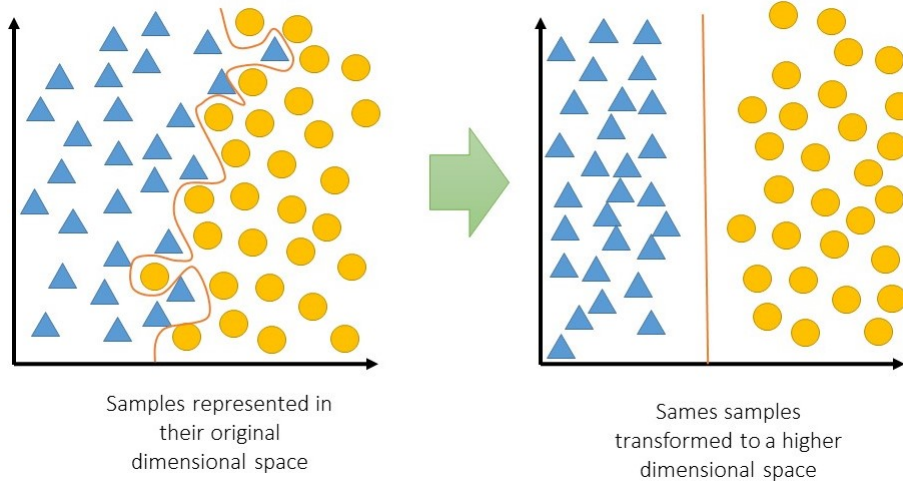


FIGURE 2.16: Example of the *kernel trick*. Since the data is not linearly separable, it is transformed to a higher space where it becomes separable.

The delivered model consists not of the hyperplane itself, but on the *support vectors* which come to be some specific training samples. The training of the SVM implies a maximization function responsible of finding the hyperplane with maximum margin making use of the support vectors, therefore the election of this *kernel function* impacts the computation time, since it may be linear, polynomial and radial based among some others.

The Kernel function generates the kernel matrix (a.k.a. the Gram matrix) of the low-dimensional space data, and are the milestone of this algorithm because they enable the hyperplane that generates the incoming data transformation into a high-dimensional space. The election of the Kernel function affects enormously the classifier's performance. Of course there is a wide offering of Kernel functions and they range from the simple linear transformations until very intricate mathematical models.

The Mercer's theorem [39] says that any symmetric function positive definite can be used as a kernel, as well as, it is possible to convolve kernels together in order to generate a resulting kernel. As the reader may deduce, the offer is very wide, however there are three different basis functions commonly used:

- **Polynomials** up to some degree s in the element x_k of the feature vector x and target vector t with a kernel of the Equation 2.8 type. For $s = 1$, a **linear kernel** is obtained.

$$\mathbf{K}(\mathbf{x}, \mathbf{t}) = (1 + \mathbf{x}^T \mathbf{t})^s \quad (2.8)$$

- **Sigmoid** functions of the x_k s with parameters κ, δ and a kernel as Equation 2.9.

$$\mathbf{K}(\mathbf{x}, \mathbf{t}) = \tanh(\kappa \mathbf{x}^T \mathbf{t} - \delta) \quad (2.9)$$

- **Radial basis** function expansions of the x_k s with parameter σ and kernel of Equation 2.10 type.

$$\mathbf{K}(\mathbf{x}, \mathbf{t}) = \exp\left(-\frac{(\mathbf{x} - \mathbf{t})^2}{2\sigma^2}\right) \quad (2.10)$$

There is a large discussion about which kernel and parameters work the best, their election is a very sensitive problem, and while there is a large theory towards this (based on a distance called the Vapnik-Chervonenkis dimension [40]), most researchers just experiment with different values and find what works best for their application using the regular validation techniques mentioned in Section 2.5.4.

The SVM are known because in comparison with other algorithms, they require longer time to train for large-scale one-versus-many mode of recognition, and besides, they require a big number of training samples per target class in order to ensure reasonable accuracy.

2.6.6 Hidden Markov Model

The HMM is a very popular classifier for temporal recognition applications since it executes classification through feature matching at each moment. It provides time scale invariance and keeps the gestures in time sequence. It is considered as a doubly stochastic process because is made up of a not observable underlying stochastic process (hidden) and can only be observed through a set of stochastic processes which produce the *observations*. This model may be represented by a set of finite states connected by transitions, where each state is characterized by the state transition probabilities and the probability distribution of the observations it may reach, as Figure 2.17 shows.

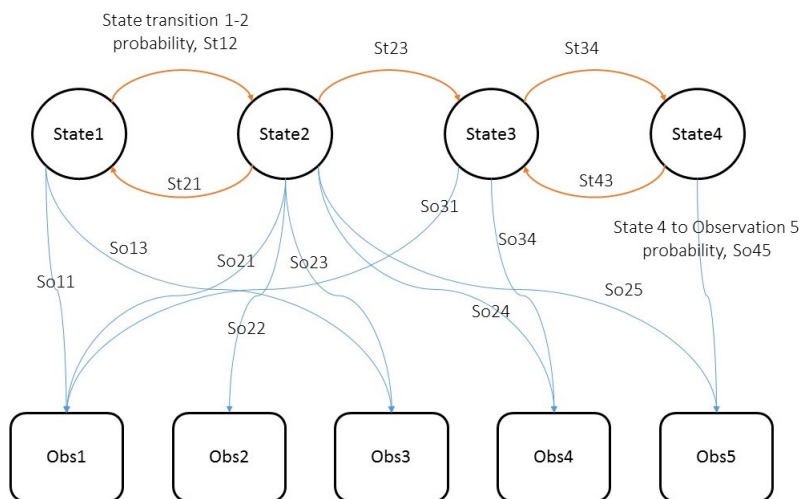


FIGURE 2.17: A representation of the HMM.

A disadvantage of this method is that the training process is time consuming and the selection of its topology structure is determined by the expert experience, i.e., trial and error method used for number of invisible states and transfer states determination.

2.6.7 Dynamic Time Warping

DTW, as HMM, is another very popular algorithm for motional classifications and is used in simple tracking recognition through the difference between the dealt gestures and standard gestures for feature matching at each moment. This algorithm generates the dynamic distance between an input sample and the set of target classes in a computationally efficient manner while coping with different speeds of motion hand gestures. Its best feature is that it calculates the similarity between two time series, even if their lengths do not match, as Figure 2.18 illustrates.

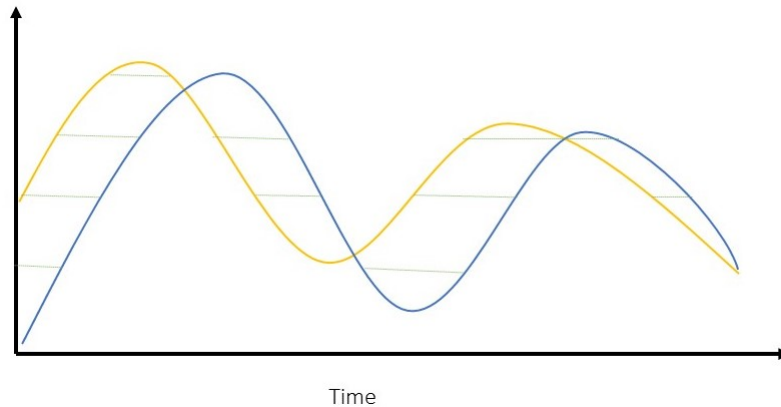


FIGURE 2.18: A graphical representation of how DTW obtains the distance in time of two out-of-phase signals.

Even when it is a reliable algorithm, the computational costs are highly considerable to use for real-time recognition, as every value in the cost matrix must be filled specially if a large database of classes is used. It may be said that DTW is the simplified version of HMM.

2.6.8 Artificial Neural Networks

Developed around the 1990's, the ANN are mathematical models inspired by the biological brain. The most popular ANN is the multilayer perceptron which consists of a series of layers, each layer is a computational node that produces an output according to the nonlinear activation function that is fed with the input of each node. Each computational node represents a *neuron* and they are interconnected in such a way that across each layer, the information travels along. The first layer receives the input features and the last layer delivers the output selection. Such networks are greedily learned through each layer and data abstraction increases with level of layers.

Its power resides in the fact that they are able to model *complex* nonlinear relationships between the input and outputs through a proper training, with this, a trained neural network provides a compact model, allowing efficient computation of new samples. They are founded to perform better than most conventional classifiers when the number of features increases or when the classification becomes very complex or nonlinear, while on the counterpart, the ANNs produce a black box model and hence cannot be mathematically interpreted as in other

approaches like the statistical one. Some others examples of ANN are Hopfield and Back-propagation.

2.6.9 Algorithms Comparison Chart

All these algorithms are considered traditional since they have been among the programmers for certain decades and therefore have been widely studied. So, a qualitative comparison is exposed in Tables [2.2](#) and [2.3](#).

Algorithm	Approach	Advantages	Limitations	Accuracy	Complexity (with the test set)	Regular application
<i>Bayesian Classifier</i>	Statistical. Assumes an underlying probability model.	Relatively immune to over-fitting due to simplicity of the model. Simple, efficient, effective and robust to noisy data.	Assumps features independence which is often violated in the real world, not feasible for many features datasets.	Good.	$O(\text{Number of attributes})$	Static classifier.
<i>Decision Tree</i>	Nonmetric, rules.	Easy to understand, robust to noises, low computational cost even for very large training datasets.	Over-fitting problem: when the tree has many levels and rules become difficult to understand.	Medium.	$O(\text{Number of Rules})$	Static classifier.
<i>k-Nearest Neighbors</i>	Instance based.	Easy to implement and to understand, does not require training.	Large storage requirements, very susceptible to high-dimensional data.	Good	$O(\text{Nr of examples} * \text{Nr of attributes})$	Static classifier.
<i>Minimum Distances</i>	Instance based.	Easy to implement and understand. Quick response.	Susceptible to high-dimensional data and close features data.	Good.	$O(\text{Nr of examples} * \text{Nr of attributes})$.	Static classifier.
SVM	Kernel method.	Provides a unique solution, solves nonlinear problem, works even when training samples are biased, flexible N-Dimensional input.	Depends on the choice of the kernel, difficult to design multi-class SVM classifiers, requires big number of training samples per class.	Very good.	$O(\text{Nr of support vectors} * \text{Nr of attributes})$, up to the kernel constant.	Static. Possible temporal implementations although difficult.

TABLE 2.2: First part of a qualitative comparison of some Supervised Learning algorithms

Algorithm	Approach	Advantages	Limitations	Accuracy	Complexity (with the test set)	Regular application
<i>HMM</i>	Statistical. Finite State Machine with double stochastic process.	Flexible N-Dimensional input when appropriate feature extraction is used such as k-means, moderated training size, provides time scale invariance and keep gestures in time sequence.	Requires a considerable training time, topology of the machine is up to the expertise of implementer, low applicability for unsegmented recognition.	Very Good.	$O(\text{Number of hidden states}^2 * \text{Length of sequence of states})$	Temporal Dynamic Classifier.
<i>DTW</i>	Based on distance comparison of two time series.	Flexible N-Dimensional input, low training size, high applicability for pre-segmented and unsegmented recognition, provides a distance measure insensitive to local compression and stretches, and a warping which optimally deforms one input serie onto the other.	Carefully specification of the approximation levels used in the alignment, problems when indexing large time series databases, not suitable for long time series.	Very Good.	$O(\text{Number of examples}^2)$	Temporal Dynamic Classifier.
<i>Neural Network</i>	Biologic inspired cells.	Well-generalizing capability, can solve dynamic or nonlinear problem.	The learned model is unable to be interpreted (black box), high complexity.	Very Good.	Depends on the architecture and number of layers.	Static and temporal dynamic classifier.

TABLE 2.3: Second part of a qualitative comparison of some Supervised Learning algorithms

2.6.10 Ensemble Methods

Ensemble learning takes a different approach in comparison with the aforementioned algorithms. Rather than finding *the* best hypothesis to classify the incoming data, they construct a set of hypotheses (called a *committee* or *ensemble*) Afterwards, those hypotheses are voted in some fashion to predict the class of new data points. Experimental evidence has shown that ensemble methods are often much more accurate than any single hypothesis since the ensemble of approaches can reduce both the bias and the variance of the compounding algorithms [41] and work pretty well even when there is very little data as well when there is too much [36].

It is said that the ensemble methods are in reality a *meta-algorithm* that under a certain heuristic, joins classification algorithms in the pursuit of variety. These meta-algorithms are then made up by multiple classifiers (known as **base learners**) to obtain a better predictive performance than could otherwise be obtained from any of the constituent classifiers [42]. It is desirable that the compounding algorithms perform their classification under different approaches and techniques in order to assure predictions from the **diverse perspectives** that the base learners provide.

2.7 Tested Algorithms

As result of the analysis and comparisons depicted in this Chapter, six supervised learning algorithms were tested in total: two dynamic and four static classifiers (final election of the static type above the dynamic approach is better explained in Section 4.1). Since this project makes use of an Ensemble approach, two algorithms were selected to become the **base learners**. This election and testing phase was required, because as expressed in Section 2.4.1, the Pattern Recognition system development requires an iterative process in order to assure the best performance and approach that each algorithm's perspective offers.

The tested algorithms were selected due to the *different* vision they employ, along with their diverse operative guidelines and characteristics displayed in Table 2.2, for the static type, and Table 2.3 for the temporal classifiers.

The gestures to be recognized are both **static** and **dynamic**, hence an initial proposal included the implementation of **temporal** classifiers as base learners. Based on the computational complexity that **HMM** and **DTW** count with, over the Neural Networks, along with the fact that the generated model is *readable*, these algorithms were first trained and tested. This implementation was discouraged due to reasons explained in Section 4.1 and followed by the insertion of **static** classifiers instead.

From Table 2.2, four static classifiers were elected:

- Adaptative Naive Bayes Classifier (ANBC).
- Minimim Distances (MinDist).
- Support Vector Machines (SVM), linear kernel.

- K-Nearest Neighbors (KNN).

Although these classifiers were originally designed for **static** classification purposes, this project makes use of them for the recognition of **temporal** gestures too. This is enabled by the generation of a specific Feature Vector and a **temporal** approach (to be depicted in Section 3.2.3).

Despite the perks that the Decision Tree algorithms show in regards of the current low number of target classes, they were not tested due to the nature of the feature vector: the *temporal approach* discouraged the generation of fixed rules.

Section 3.2.4 talks about the elected algorithms, whereas Section 4 backs up their election.

Chapter 3

Hand Gesture Recognition System for UAV Control

This Chapter contains the explanation of the implemented system. As start point, the architecture of the project is presented, the hardware components are mentioned and the system is subdivided in two main subsystems. Following section contains the description of the subsystem responsible of the static and dynamic hand gestures recognition and its translation into suitable tags; this subsystem is called *Hand Gesture Recognition* and holds the ensembled supervised learning meta-algorithm: It is the core mechanism of the system. Upcoming section holds the description of the Unmanned Aerial Vehicle (UAV) control subsystem and contains an explanation of an Unmanned Aerial System (UAS) requirements. Last but not least, section 3.4 contains the implementation details of this system, by introducing the Robotic Operating System (ROS) as the platform where this project is developed, the initial architecture is now translated to ROS terminology and their elements are displayed in a graphical and understandable way. The Gazebo software is the simulator where the UAV behaviour is emulated, and therefore, a screenshot of its environment is presented.

3.1 Architecture of the System

A first approximation to the complete system is contained in Figure 3.1. A person is responsible of generating hand gestures having in mind the UAV movement. Retrieval from the gestures information is made through the Leap Motion controller, which delivers a Skeleton Model to the Laptop computer. The Leap Motion data gets processed, the gesture is recognized and afterwards, the recognized class is sent to an UAV Control module; this one sends the appropriate information to the UAV's computer(Erle Brain 2) through WiFi in the MavLink protocol. The Erle Brain board internally process the incoming information with support of the ArduPilot-Mega(APM) software (the autopilot) and modifies the UAV behaviour. In a parallel process, the UAV Control module delivers the same information to a software executed in the same Laptop which acts as a Ground Control Station.

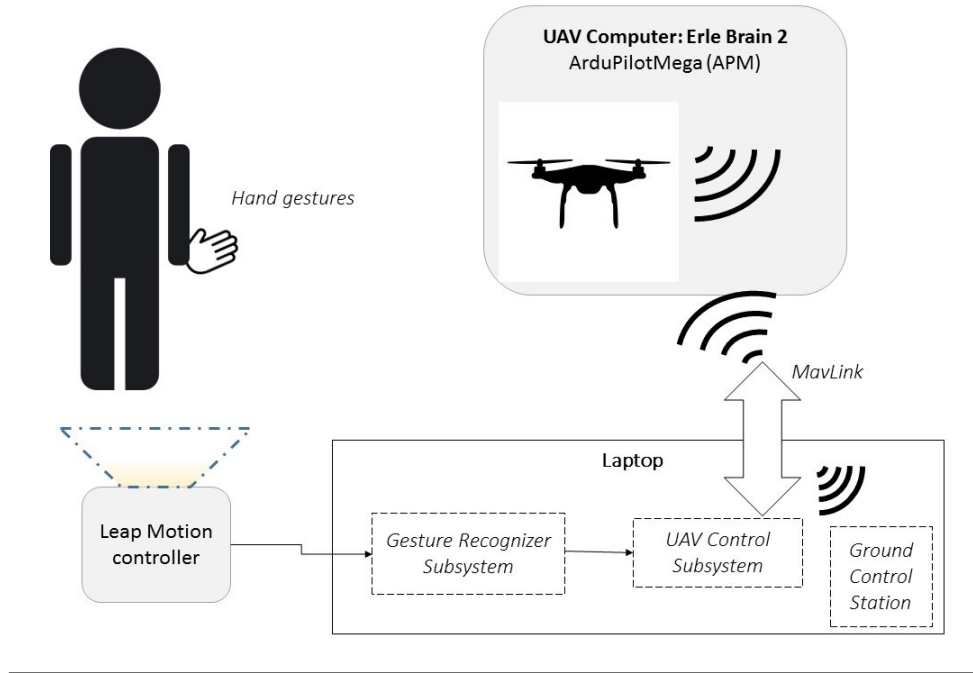


FIGURE 3.1: A logic view of the system.

Taking the Figure 3.1 as starting point, two subsystems are extracted: the Hand Gesture Recognizer and the UAV Control subsystems, both of them explained as follows.

3.2 Gesture Recognition subsystem

The Gesture Recognizer subsystem is displayed in Figure 3.2 and is the responsible of turning the hand gestures into *target classes* that are processed by the UAV Control stage. This subsystem is made up by two stages: The Leap Motion and the Hand Gesture Recognition. The first one is in charge of receiving the Hand gestures and with the transducer function provided by the Leap Motion controller, a Skeleton Model is obtained in form of *Frames*. This data is supplied to a following stage where the data is filtered and processed in order to retrieve an appropriated Features Vector. Following stage receives this Vector supplying the data to the **Base Learners (BL)** which execute their recognition in a parallel form (in threads) and deliver their recognized hypothesis to the *Hypothesis Vector*. This Vector is fed to the *selection* algorithm which plays a stabilization and selection function and is the last responsible of delivering a final *Recognized Class* to the following phase (the UAV Control subsystem). The Hand Gesture Recognition stage is where the Ensemble learning approach happens, due to the integration of the Base Learners, which use their own approach to emit their hypothesis; Section 3.2.4 explains in detail the individual components.

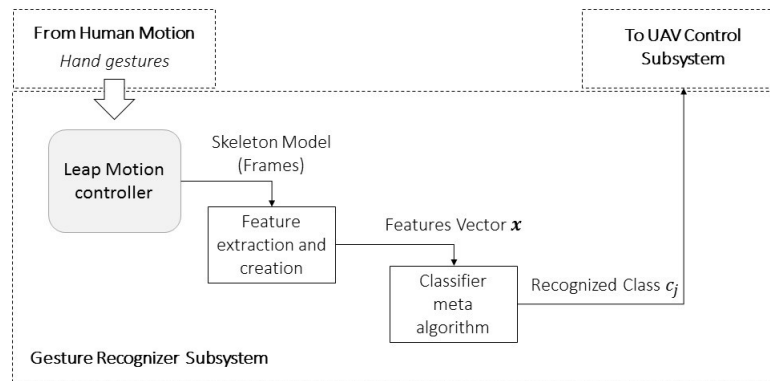


FIGURE 3.2: Gesture recognizer subsystem architecture.

3.2.1 The Leap Motion Controller

With the development of new hardware alternatives for data retrieval (a.k.a. *sensors*) the offer of optical 3D sensors has widened [15]. This brings the opportunity to experiment with devices such as the *Leap Motion controller*, shown in Figure 3.3.

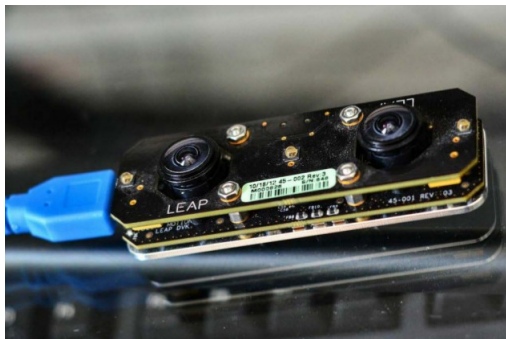


FIGURE 3.3: Leap Motion controller, cover removed

This device (released back in 2013 [43]) has gained popularity since it promises an accurate interaction between the user and a regular computer, however with the release of its Software Development Kit (SDK) and the effort of excited developers, the Leap Motion controller may be used not as a *controller* itself, but as a *transducer* or sensor that feeds an application, which performs something. As a matter of fact, the device shown in Figure 3.3 executes only a very small amount of data computing, leaving the real expensive image processing to the computer.

The Leap Motion controller consists of a pair of Infrared Light (IR) cameras and three IR leds which project an IR pattern in order to generate an image of predefined objects such as hands and pointy tools with depth information. Hence, the device may be categorized as a tracking system based on the *Stereo Vision* principle. This is one of the three principles of measurement for optical 3D sensors: structured light, time of flight and stereo vision; and consists of two optical 2D cameras with known extrinsic parameters[16]. It determines the depth of the objects in the scene based on the search of the points correspondence between captures in both images.

This information is then used in the optical tracking system in such a way that the position of predefined markers (such as fingers or tools) in the Cartesian space of the viewed scene are detected.

The information that Leap Motion delivers is the hand *Skeleton model* it sees in form of *Frames*. The **Skeleton model** consists of objects (like fingers or tools) **positions** relative to the Leap Motion controller's origin point [17]. The origin point is located at the center of the top surface device; Figure 3.4 shows the coordinate system, the axes and directions as well as the correspondence it generates with an XY plane (for a computer screen mapping).

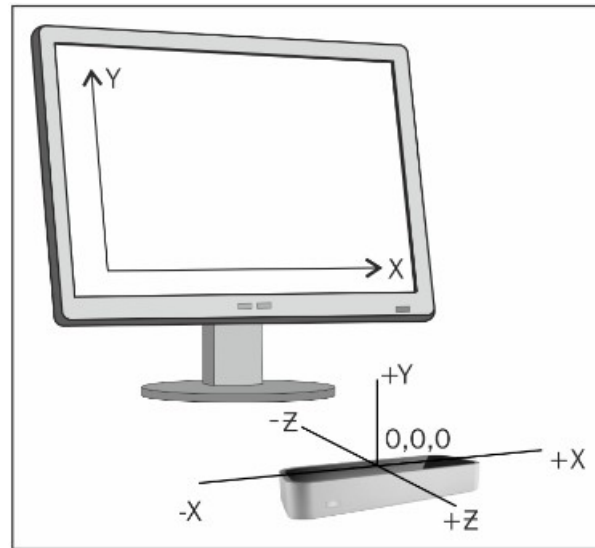


FIGURE 3.4: Leap Motion coordinate system and its correspondence with a screen.

The **Frames** are data structures that hold the hand skeleton model information and are generated accordingly to the current frame rate. They may be understood as snapshots of the hand skeleton at the time the frame was *captured*. The frame rate fluctuates depending on the available computing resources, activity within the device field of view, software tracking settings, and other factors, and therefore can not be precisely determined or adjusted. In *Object Oriented Programming* terminology, the frames are *objects* containing all the tracked information regarding hands, fingers and tools within the Leap Motion field of view, therefore a posterior data processing is required to provide the algorithms with a suitable Feature Vector.

3.2.2 Data Filters

It is necessary to implement *filters* for the data across the frames because Leap Motion loses, from time to time, certain frames or even displays completely incorrect information between consequent frames. This led to data that show an facing-down hand in one frame, followed by a facing-up hand or even worst.

This undesired behavior was corrected by implementing filters with the pattern of Equation 3.1:

$$filteredValue = filteredValue + decay * (value - filteredValue) \quad (3.1)$$

Where *decay* is a factor that sets the permission to modify the current *filtered value* with the incoming data (*value*).

This filter was implemented for all the features, although it was specially necessary for the **Z position data**. It is known in the Leap Motion developers community that the Z direction is not very accurate and therefore, the emphasis of the filter for this coordinate direction [44].

Although Leap Motion is able to provide much more information than the first six shown features, their use was tested and finally neglected due to the risk of falling into the peaking phenomenon discussed earlier (Section 2.5.3).

3.2.3 The Feature Vector

The Frame contains the information required to constitute the **feature vector** x (Equation 2.1, Section 2.5.1) to be used by the base learners.

As mentioned earlier, the conformation of the feature vector x is highly important due to its impact in the algorithm performance, therefore, the right choice of the compounding features x_i was determined after a series of exhausting tests (to be explained in Chapter 4). The full feature vector x is

$$\mathbf{x} = [x_1, x_2, \dots, x_i, \dots, x_{14}] \quad (3.2)$$

where the features x_i are:

- x_1 : Palm center X coordinate $Palm_x$.
- x_2 : Palm center Y coordinate $Palm_y$.
- x_3 : Palm center Z coordinate $Palm_z$.
- x_4 : Thumb tip X coordinate $Thumb_x$.
- x_5 : Thumb tip Y coordinate $Thumb_y$.
- x_6 : Thumb tip Z coordinate $Thumb_z$.
- x_7 : Vector of Change.
- x_8 : Palm center Delta X coordinate $Palm_{\Delta x}$.
- x_9 : Palm center Delta Y coordinate $Palm_{\Delta y}$.
- x_{10} : Palm center Delta Z coordinate $Palm_{\Delta z}$.
- x_{11} : Thumb tip Delta X coordinate $Thumb_{\Delta x}$.

- x_{12} : Thumb tip Delta Y coordinate $Thumb_{\Delta y}$.
- x_{13} : Thumb tip Delta Z coordinate $Thumb_{\Delta z}$.
- x_{14} : Delta yaw angle ΔYaw .

For auxiliary purposes in the selection algorithm, the *fist presence* and *number of fingers* were calculated as well.

Among the fourteen features x_i , only $x_1 : x_6$ were retrieved directly from each frame, while features $x_7 : x_{14}$ were generated by programming calculations using a so-called *temporal approach*. The **temporal approach** consists of the determination of the *Delta* values for each point of interest (the palm center and thumb tip in this case) and resembles a *footprint* across a certain number of k frames.

Figure 3.5 expresses this idea, when the hand is located in an initial position, a Frame 0 is collected (containing the x, y, z position data for $Palm_0$ and $Thumb_0$, and the rotation angle for Yaw_0). When the hand is displaced from this initial configuration to a posterior position (i.e. the hand performed a dynamic gesture), an array of size k gathers the same raw information across frames, resembling a buffer.

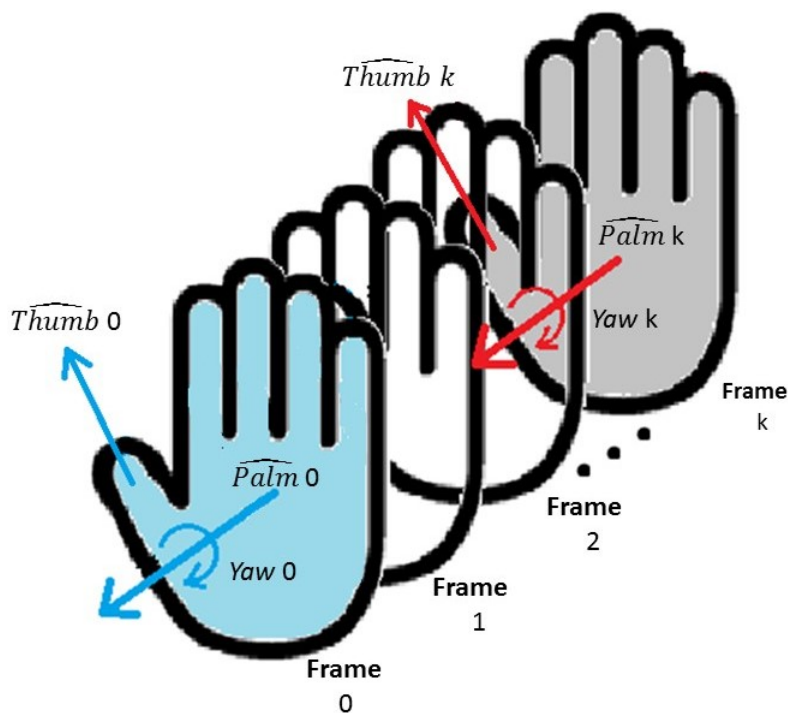


FIGURE 3.5: Difference across Frames

The **delta values** (Δ) are obtained as their difference across k frames. This principle is used for position and angle values. In the case of the palm delta position values ($x_8 : x_{10}$), the stored array allows to calculate the difference between the initial and the K th position vectors (Palm 0 to Palm K), which is displayed in Figure 3.6.

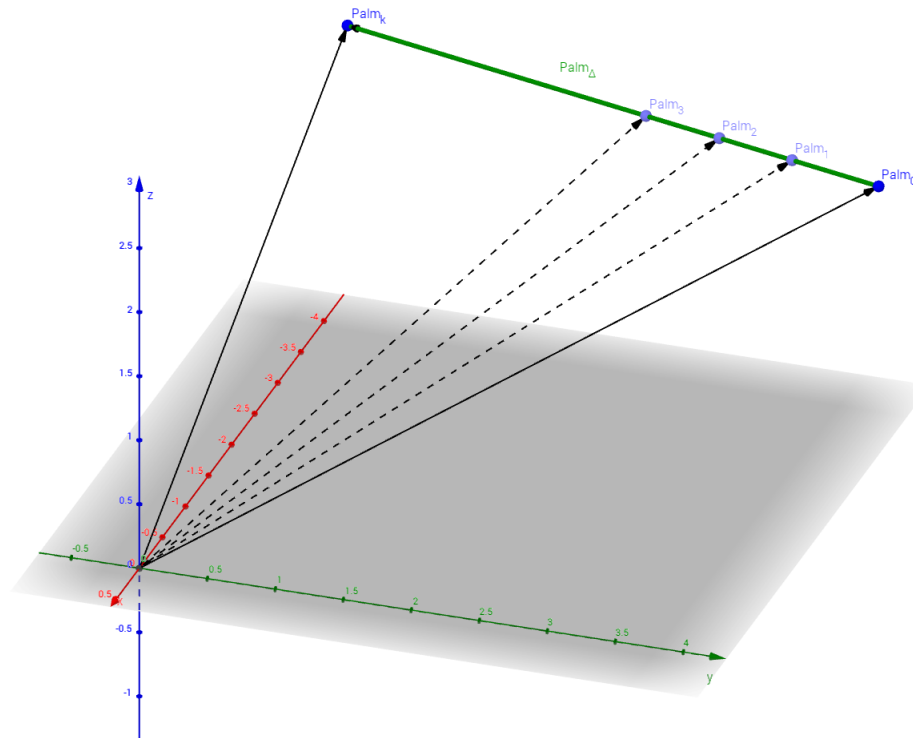


 FIGURE 3.6: Palm Delta vector (ΔPalm).

The Equation 3.3 shows this principle with the Palm Δx coordinate calculation (feature x_8).

$$\Delta \mathbf{x} = x_0 - x_k \quad (3.3)$$

The **previous frame number** k is set to 10 in this system, although it may be changed to some other number. This means that the delta values are calculated from the difference between the current position and the position across the last ten frames.

Back to the full feature vector x , and for clarification purposes, each feature x_i is described as follows:

- $x_1 : x : 3$: **Palm X, Y, Z coordinates.** Correspond to the location of the palm center calculated by the Leap Motion driver. It is displayed in Figure 3.7.

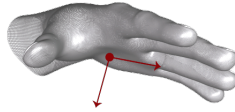


FIGURE 3.7: Leap Motion Palm Vectors

- $x_4 : x : 6$: **Thumb X, Y, Z coordinates.** Likewise, these coordinates correspond to the location of the thumb finger tip, shown in Figure 3.8.



FIGURE 3.8: Leap Motion Finger Vectors

- x_7 : **Vector of Change.** This feature is a *composed* attribute and reflects the coordinate where the major difference was observed. It assigns a label that corresponds to the following rule:
 - 1 if *Major Change* is in the +X axis.
 - 2 if *Major Change* is in the -X axis.
 - 3 if *Major Change* is in the +Y axis.
 - 4 if *Major Change* is in the -Y axis.
 - 5 if *Major Change* is in the +Z axis.
 - 6 if *Major Change* is in the -Z axis.
 - 7 if *Major Change* is in the positive Yaw angle direction.
 - 8 if *Major Change* is in the negative Yaw angle direction.

where *Major Change* is the greatest position (or angle) difference between the initial and the k th value.

- $x_8 : x : 10$: **Palm Delta X, Y, Z position.** These Features are calculated based on the X, Y or Z position difference between the current and the k th previous frame .
- $x_{11} : x : 13$: **Thumb Delta X, Y, Z position.** Similarly to the Palm Delta X, Y, Z position, these differences are calculated based on the Thumb tip positions across k th frames.
- x_{14} : **Delta Yaw Angle.** Likewise Palm and Thumb Delta, the hand yaw angle difference is calculated from the current to the last k th frame. The regular yaw angle of the hand movement is the one around the palm center as Figure 3.9 shows.

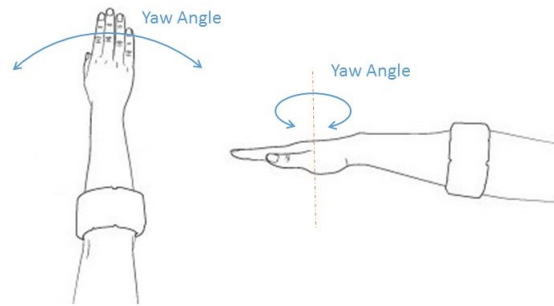


FIGURE 3.9: Hand angular movements.

The previously explained **temporal approach** is the mechanism that allows the usage of **Static** classifiers into the current **Dynamic** gestures classification. Without it, the implementation of the static algorithms would be impossible and it would be required to make use of the temporal algorithms (such as HMM, DTW or Neural Networks), which as mentioned in the comparison charts 2.2 and 2.3, require much more computation time, bigger training data, among some other special considerations. The implementation of the composed features enabled the option to work with these humble tools.

3.2.4 Classification

This stage is made up by two base and parallel classifiers which deliver their hypothesis to a *stabilizer* module that according to a predefined logic, delivers a recognized command. Taking as starting point the Figure 3.2, the diagram in Figure 3.10 becomes a more explicit resource.

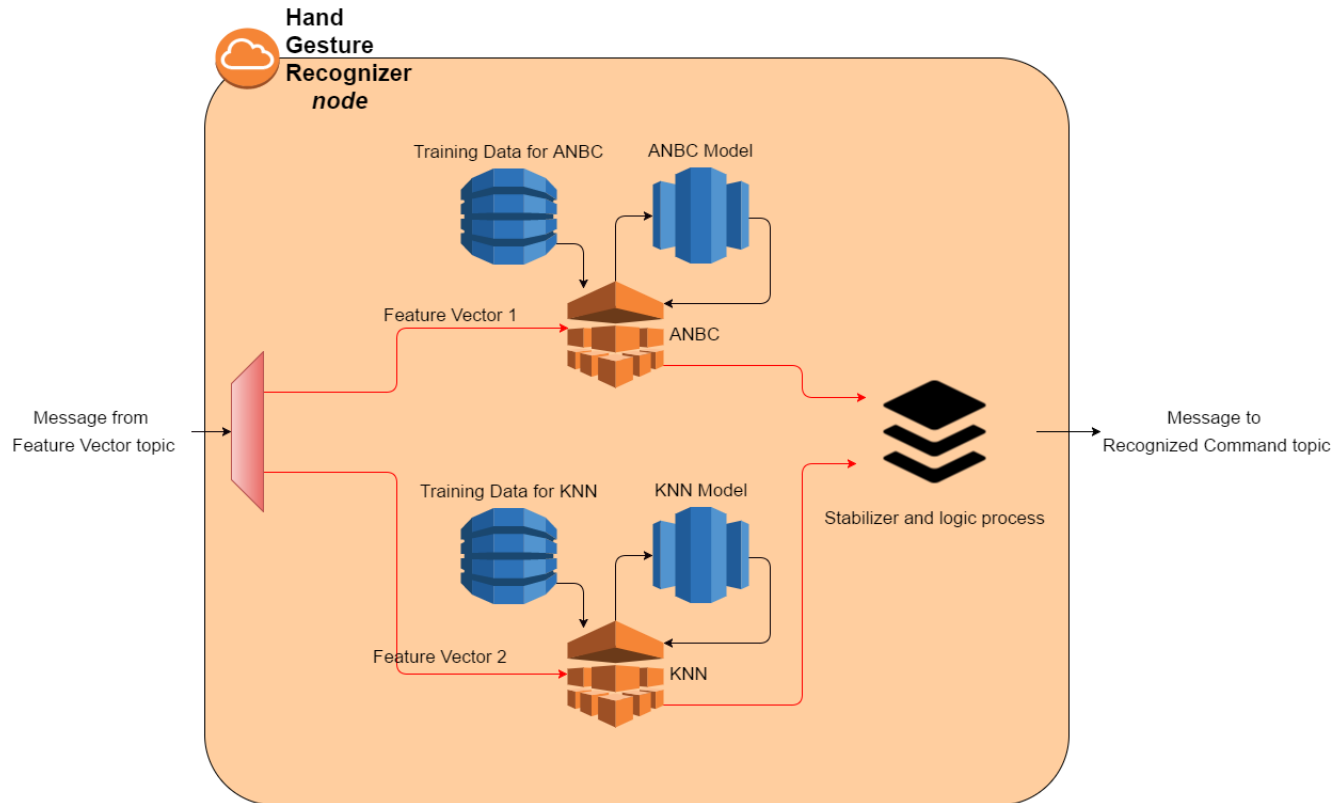


FIGURE 3.10: Hand Gesture Recognizer node.

It all begins with the feature vector retrieved by the previous stage. As mentioned earlier (Section 2.5.3), every classification problem and tool require specific treatment and details attention. In this case, each classifier requires its own *sub* feature vector, each one called *Feature Vector 1* and 2. In the same way, each classifier counts with its own *Training Data* set which generates an independent knowledge *model*.

Everytime the system is started it may, either load the knowledge model, or load the training data and train itself according to its own parameters. When the classifiers are trained, they are able to receive their own feature vector, which is specially splitted by a previous divider. Since the classifiers run in independent threads, they receive their features vectors and provide their hypothesis target classes to the stabilizer.

3.2.5 Base Learners

The classifiers used as base learners are the **Adaptive Naive Bayes Classifier** (ANBC) and the **Support Vectors Machine** (SVM) based on the results displayed in Section 4. Since they have been already explained in Section 2.6, their working principles are not covered here. Once the hypothesis are provided, the stabilizer ponders the recognized command across ten votes and through the media calculation delivers the final command, which comes to be one of the predefined target classes c_j (equation 2.2). Therefore, the full classes vector c is

$$\mathbf{c} = [c_0, c_1, \dots, c_j, \dots, c_{13}] \quad (3.4)$$

where each target class c_j is shown in Table 3.1 accordingly to the hand movement described.














Target class c_j	Gesture description	Illustration	Target class c_j	Gesture description	Illustration
1	Fist		8	Front movement	
2	Number one		9	Back movement	
4	Number two		10	Right movement	
3	Number three		11	Left movement	
5	Number four		12	Hand right rotation	
6	Up movement		13	Hand left rotation	
7	Down movement				

TABLE 3.1: Relation between target class c_j and the redognized hand gestures.

While $c_1 : c_5$ are made up by **static** and fixed hand positions, $c_6 : c_{13}$ require an extended and separated fingers hand to generate the described **movements** (dynamic gestures).

Although it is not expressed in the Table 3.1, the classifiers provide a zero as hypothesis if the gesture was not recognized. If several gestures are not recognized, the stabilizer algorithm delivers 0 as the recognized command.

3.3 UAV Control Subsystem

This subsystem is responsible for turning the target classes (recognized from the previous stage) into commands that modify the UAV behaviour. Figure 3.11 provides an overall scheme of the elements that play a role.

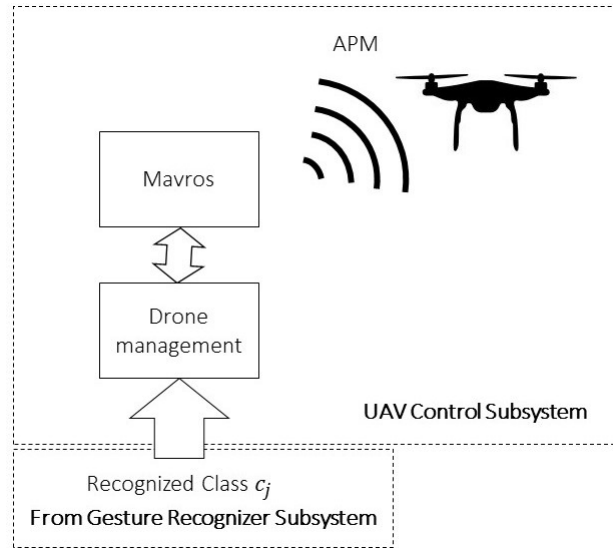


FIGURE 3.11: UAV Control Architecture.

In this stage, the recognized target class is assigned to a certain UAV behaviour according to Table 3.2. The recognized class (or *recognized command*) is retrieved by the *Drone management node*. This software is responsible for establishing communication with the *mavros node* (through topic or services depending on the command), checking the UAV status, requesting services to *mavros* and translating the *recognized commands* into understandable orders for the UAV movement.

Target class c_j	Gesture description	UAV behavior
1	Fist	Set to Brake mode
2	Number one	Turn on (Arm UAV)
4	Number two	Take off
3	Number three	Land
5	Number four	Return to launch
6	Up movement	Increase throttle
7	Down movement	Decrease throttle
8	Front movement	Increase Pitch
9	Back movement	Decrease Pitch
10	Right movement	Increase Roll
11	Left movement	Decrease Roll
12	Hand right rotation	Increase Yaw
13	Hand left rotation	Decrease Yaw

TABLE 3.2: Relation between target class c_j , hand movements and UAV behaviour.

On its side, the *mavros node* is the direct gate to the APM manipulation. Since APM handles

the MAVLink protocol (Micro Air Vehicle Communication Protocol, a specialized communication protocol for micro vehicles [45]), mavros receives the *messages* and *services requests* from the management node and translates them into MAVLink for its interpretation in the APM software.

Due to safety reasons, it was established that if the previous stage does not provide a recognized target class (delivers a class 0), the command is set to Brake mode.

3.3.1 UAS Elements

As explained in Section 2.1.2, an Unmanned Aerial System (UAS) is comprised of not just the UAV. However, it is important to talk about these elements. The UAV's computer is the board named **Erle Brain 2**[46] and is mounted directly on the back of the copter, shown in Figure 3.12. It is an open hardware and Linux based controller interface and consists of a *BeagleBone Black Board* and a *PixHawk Fire Cape (PXF)*. It holds a Linux Operating System, Debian image, which enables the ROS interface and holds the execution of the ArduPilotMega (APM) software: the UAV controller. The APM [47] is an open-source project focused in the mobile robots control and is the platform for independent robot enthusiasts. APM is at the same time, a *Dronecode Software Platform*[48] member, a Linux Foundation project.



FIGURE 3.12: Erle Brain 2 and Erle Copter.

The same Figure shows on the right the elected UAV: the Erle Copter, a DIY quadcopter that requires a full parameter tuning and calibration, which is very time consuming for a novice.

The Ground Control Station (GCS) establishes communication with the UAV through MAVLink and is the platform that provides a general status of the UAV. It is able to retrieve and modify parameters as well as setting some isolated features. For the UAV tuning and calibration, Mission Planner[49] is the used GCS, while for the system execution, APM Planner[47] and MavProxy[50] are the tools to keep track of the UAV and monitor its status.

3.4 Implementation

Section 3.3 contains the architecture of the UAV Control subsystem and a presentation of the elements that conform an UAS. Taking these resources as springboard, current section contains a brief introduction to the *Robotic Operating System*, presents the software particles (called *nodes*) developed under this framework to enable the UAV Control, and finally, renders the simulation details.

3.4.1 The Robotic Operating System (ROS)

The system is developed on top of the Robotic Operating System[51] (ROS), a member of the Open Source Robotics Foundation[52] (OSRF). Being strict, ROS is not an Operating System itself, but a framework made up by tools, libraries and conventions that aim to unify the creation of software robotics related. It is an open source project born in the Stanford University in 2008 and has gained researchers and industry attentions since then, becoming nowadays the *de facto* robotics standard. As its wiki page[53] mentions, “ROS is an open-source, meta-operating system for your robot” that works alongside a Linux Operating System. This project runs ROS over Ubuntu 14.04 LTS.

One of ROS’s main capabilities lies in the *Distributed Computation* that it supports. ROS software components are called **nodes** and its guiding design rule is that each one works independently and holds a single but general capability. There are two mechanisms that the nodes may use when they need to communicate: **messages** and **services**. Messages are enabled through a **topic** channel, while services are performed through direct communication between the nodes with the server/client model. The election of whether to use messages or services between nodes, resides on the feedback: while the service request always waits until a service response is received, the messages publication to the topic never receives a notification or feedback.

3.4.2 ROS Nodes

The project runs on top of the ROS platform and as expected, the subsystems presented in Figure 3.1 are translated into ROS terms (nodes, topics and services) and displayed in Figure 3.13.

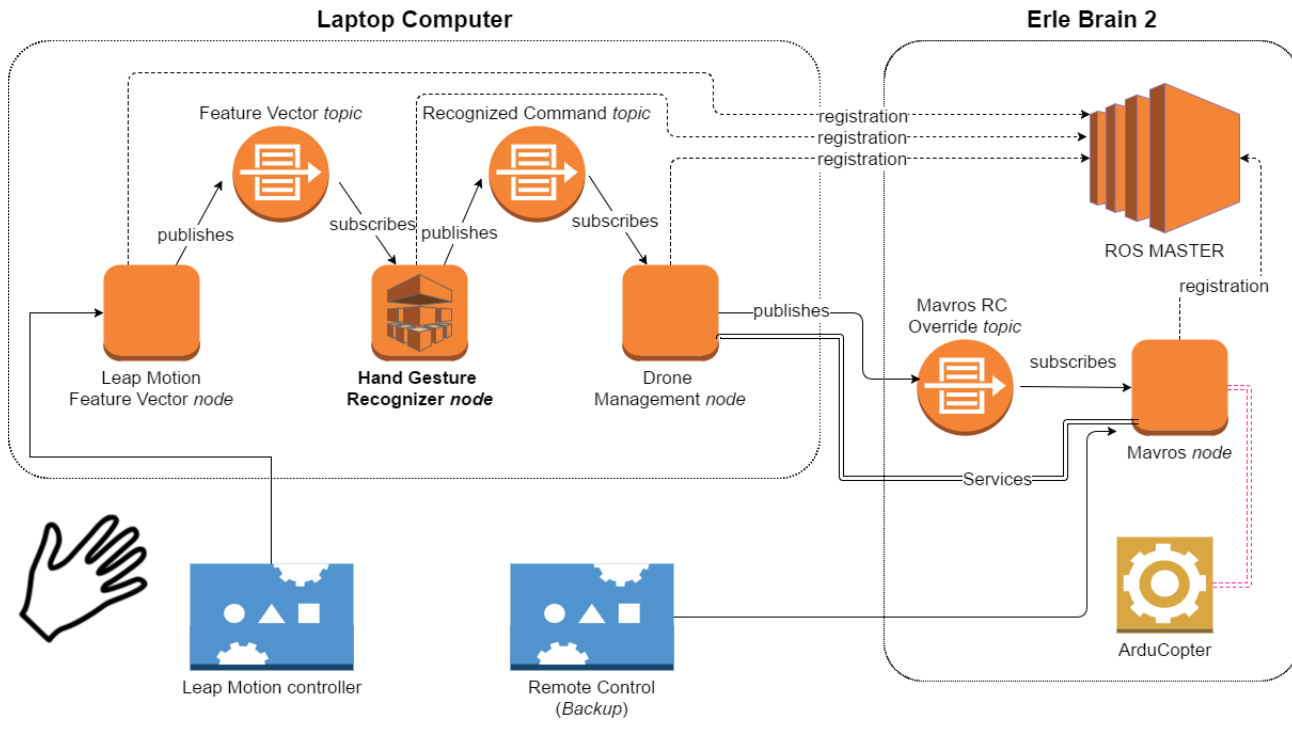


FIGURE 3.13: Interaction between ROS nodes.

The system runs in two platforms: the laptop where the gesture recognition is achieved, and the board over which the APM software runs; each one holds nodes with specific functions. Since it is a distributed system, all the nodes need to register with the *ROS MASTER*, which holds the complete system communication and integration. This is executed on the Erle Brain due to safety reasons, being the fact that it holds the node (mavros) that establish the direct communication with the APM.

Besides the platforms, the system counts with two external hardware components, the Leap Motion controller and the Remote Controller (for the UAV), which establishes communication to the UAV through the mavros node as well.

The complete system is then made up by three topics and services requests among the nodes:

1. **Leap Motion Feature Vector:** Receives the Leap Motion skeleton model data and generates the feature vector *message* to be published in the *Feature Vector topic*.
2. **Hand Gesture Recognizer:** Contains the Machine Learning algorithms, is subscribed to the *Feature Vector topic*, and everytime it process an incoming message, it publishes the recognized class in the *Recognized Command topic*.
3. **Drone Management:** Provides the UAV management and translation from classes into APM commands. It is subscribed to the *Recognized Command topic* and everytime it receives an incoming message, it verifies the current status and if it is safe and valid, publishes the correspondent command in the “Mavros RC Override” *topic*. It is also a client to the services offered by the Mavros node when it is necessary.

4. **Mavros**: This node owes its name to its main function. It translates into the MAVLink protocol the incoming messages it receives and sends them directly to ArduCopter (APM).

As mentioned, there are occasions where Drone Management requests services to Mavros. These services are:

- cmd ~arming
- cmd ~takeoff
- set mode

Therefore, the interaction between Drone Management and Mavros is as follows: Services are required in the event of commands 1 to 5, and messages are generated for commands 6 to 13 according to Table 3.2.

The Hand Gesture Recognizer node and Drone Management node are configured as *critical*, whereas Leap Motion Feature Vector node is only *required*. This means that if the Leap Motion node fails for whatever reason, it will reinitialize without further problem. But if a critical node is killed, then the laptops executing nodes close and leaves the UAV command to the physical Remote Control. This is one of the perks of having a decoupled system such as ROS.

Figure 3.14 displays the all the nodes and topics. It consists of two *namespaces* which are mechanisms to group nodes belonging to a same category; in this case, the namespaces are *hgr4dc_ns* and *mavros*. *hgr4dc_ns* contains the nodes for the hand gesture recognition and the drone management, while the *mavros* namespace gathers the *mavros* node and the topics it regulates.

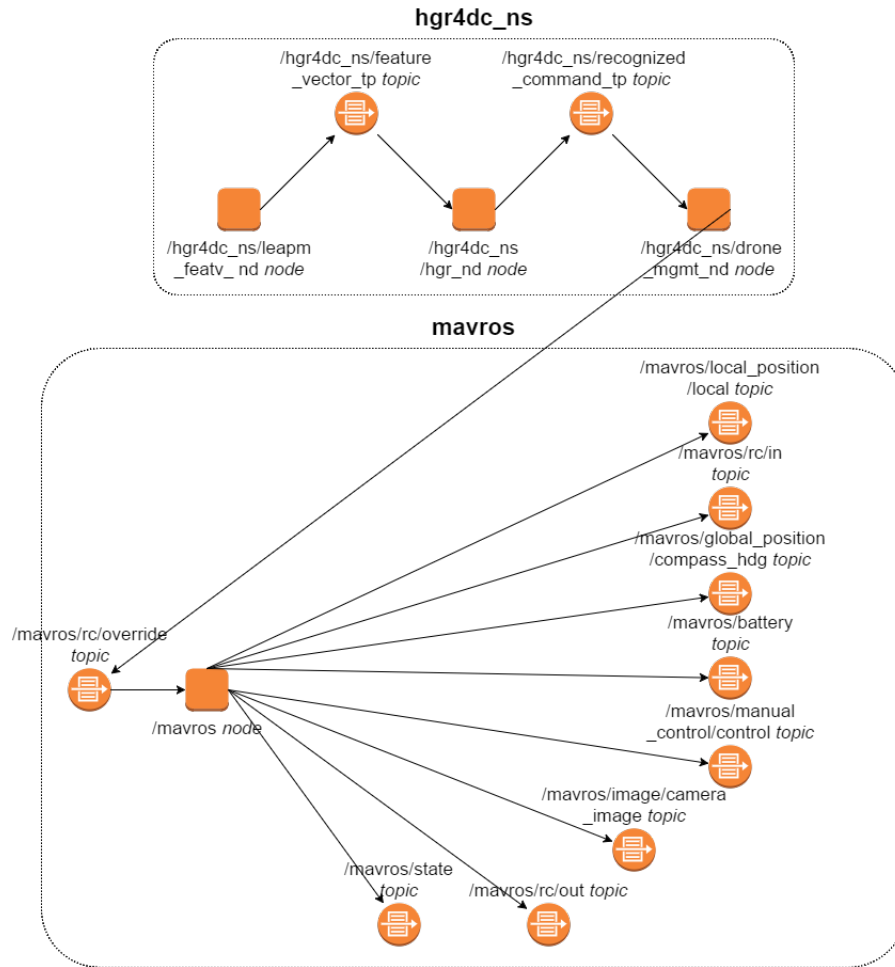


FIGURE 3.14: Graph with nodes and topics used in the recognition phase and the consequent UAV control.

3.4.3 System Pseudocodes

Once the two compounding subsystems (Hand Gesture Recognition and UAV Control) have been explained and the ROS terminology introduced, a pseudocode of them may be expressed. Algorithm 1 contains the Feature Vector node working principle, Algorithm 2 shows the Gesture Recognition subsystem and Algorithm 5 manifest the UAV control in pseudocode terminology.

As mentioned in Section 3.4.2, the interaction between the nodes and subsystems is regulated by the ROS Master, who is also responsible for bringing them to life and/or killing them.

```

input : Leap Motion Frames
output: Feature vectors  $\mathbf{x} = [x_1, x_2, \dots, x_n]$ 
initialization ;
Leap Motion controller creation;
while This node is alive do
  if controller is connected to the service then
    read current Frame;
    refresh the latest 10 Frames buffer;
    filter noise;
    update hand confidence, strength, radius and finger positions with the filtered
      data;
    generate the clean hand data;
    if updated hand information is valid then
      calculate deltaValues across the latest 10 Frames;
      generate FVectorMessage;
      if deltaValues are within the range then
        generate VectorOfChange;
        detect fist or extendedHand;
        FVectorMessage.yaw = hand.yawAngle;
        FVectorMessage.palmX = hand.palmX;
        FVectorMessage.palmY = hand.palmY;
        FVectorMessage.palmZ = hand.palmZ;
        FVectorMessage.thumbX = hand.thumbX;
        FVectorMessage.thumbY = hand.thumbY;
        FVectorMessage.thumbZ = hand.thumbZ;
        FVectorMessage.vectorOfChange = VectorOfChange;
        FVectorMessage.palmDeltaX = deltaValues.palm.x;
        FVectorMessage.palmDeltaY = deltaValues.palm.y;
        FVectorMessage.palmDeltaZ = deltaValues.palm.z;
        FVectorMessage.thumbDeltaX = deltaValues.thumb.x;
        FVectorMessage.thumbDeltaY = deltaValues.thumb.y;
        FVectorMessage.thumbDeltaZ = deltaValues.thumb.z;
        FVectorMessage.isFist = fist or extendedHand;
      else
        | FVectorMessage = latest FVectorMessage sent;
      end
      publish the FVectorMessage on topic feature_vector_tp;
    end
  else
    | print message No LM found;
    | kill this node;
  end
end

```

Algorithm 1: Feature extraction node pseudocode.

input : Training data $X = [[\mathbf{x}_1, \mathbf{t}_1], [\mathbf{x}_2, \mathbf{t}_2], \dots, [\mathbf{x}_M, \mathbf{t}_M]]^T$ and feature vectors
 $\mathbf{x} = [x_1, x_2, \dots, x_n]$

output: Target class c_j

initialize baseLearner1 and baseLearner2 objects ;
assign baseLearner1 and baseLearner2 objects to threads;
load Training Data for baseLearner1 and baseLearner2;
train baseLearner1 and baseLearner2;

while *This node is alive* **do**

if *baseLearner1.trained and baseLearner2.trained* **then**

generate RecognizedCommandMessage;

if *new FVectorMessage is published on topic feature_vector_tp* **then**

create featureVector1 and featureVector2;

hypothesis1=baseLearner1.recognize(featureVector1);

hypothesis2=baseLearner2.recognize(featureVector2);

if *hypothesis1.type=dynamic and hypothesis2.type=dynamic* **then**

recognizedGesture=stabilize(filter(hypothesis1, hypothesis2));

recognizedGesture.type=dynamic;

else

recognizedGesture=staticRecognition(featureVector1,featureVector2);

recognizedGesture.type=static;

end

RecognizedCommandMessage=recognizedGesture;

end

publish the RecognizedCommandMessage on topic recognized_command_tp;

end

Algorithm 2: Hand gesture recognition subsystem pseudocode.

input : Training data $X = [[\mathbf{x}_1, \mathbf{t}_1], [\mathbf{x}_2, \mathbf{t}_2], \dots, [\mathbf{x}_M, \mathbf{t}_M]]^T$ and feature vectors
 $\mathbf{x} = [x_1, x_2, \dots, x_n]$

output: Target class c_j

recover the training data set and complete initialization;
turn 20% of the data set into Test set and 80% in Training data;
convert the training data into a frequency table;
create a likelihood chart by finding the probabilities of the feature vectors and the corresponding classes;
Training: use Bayesian equation to calculate the posterior probability for each class in the training data;

$$P(\mathbf{x}|c_j)P(c_j) = \prod_{i=1}^n P(x_i|c_j)P(c_j);$$

while *This object is alive* **do**

Calculate the posterior probability of the incoming feature vector;

Classify: the class c_j with the highest posterior probability is the outcome of prediction;

end

Algorithm 3: Pseudocode of base learner 1: ANBC Algorithm.

input : Training data $X = [[\mathbf{x}_1, \mathbf{t}_1], [\mathbf{x}_2, \mathbf{t}_2], \dots, [\mathbf{x}_M, \mathbf{t}_M]]^T$ and feature vectors
 $\mathbf{x} = [x_1, x_2, \dots, x_n]$
output: Target class c_j

recover the training data set and complete initialization;
turn 20% of the data set into Test set and 80% in Training data;
for the linear kernel, compute the kernel of distances between the datapoints (x, c) of the Training Data;
 $K = XX^T$;
Training: identify the support vectors as those that are within some specified distance of the closest point and dispose of the rest of the training data;
while *This object is alive* **do**
 | Classify: for the given feature vector x , use the support vectors to classify the data for the relevant kernel;
end

Algorithm 4: Pseudocode of base learner 2: SVM Algorithm.

3.4.4 Simulation on Gazebo

A very useful advantage of using independent nodes is the fact that they are not looking for specific *nodes*, but for **topics** (in case they use only messages). This may look as a slight feature, however it enables one of the greatest ROS capabilities, which is the ability of running a system in whatever other interface that communicates through the same topics. And it finally enables the integration of a virtual UAV, executed through a simulator. This transition happens seamlessly.

Gazebo[54] is an open-source project robot simulator, member of the OSRF as well as ROS, which enables an environment for safe testing of a robotic project. For this implementation, given the nodes independence, it was possible to visualize the UAV movement in the Gazebo environment by replacing the real Mavros node with one that gets attached to a virtualization of the ArduCopter firmware.

This simulation counts with the virtualization of elements that together, mimic the behaviour of an UAV. The core of the ArduPilot software lies in a launch script (*simvehicle.sh*), well known of being an APM virtualization, which generates an instance of the MavProxy console and optionally a map; the console works as a very simple GCS (mentioned in Section 3.3.1). This script is part of the project called *Software in the Loop* (SITL), generated by the same community that leads the ArduCopter framework [55]. The APM virtualization counts with an Inertial measurement unit (IMU) which provides linear acceleration, angular velocity, atmospheric pressure and altitude, while the virtual GPS integration is provided by the plugin developed by the Technische University of Darmstadt [56] (*ardupilot_sitl_gazebo_plugin*) and delivers longitude, latitude and altitude data to allow the UAV self control.

Through the usage of common MAVROS topics, a MAVLink bridge may be generated for another GCS connection. Using the UDP protocol our ROS nodes send specific navigation commands to the virtual UAV and the simulated behaviour is calculated in the SITL and displayed in Gazebo. The step lock mechanism enforces a pause of the Gazebo simulation until it receives

```

input : Target class of recognized gestures as commands
output: Commands for the Mavros/RC topic
initialization, reset(throttle);
while This node is alive do
  reset(roll,pitch,yaw);
  if new RecognizedCommandMessage is published on topic recognized_command_tp then
    command=RecognizedCommandMessage;
    if command.type=dynamic then
      | set flight mode AltHold;
    end
    switch command do
      Case 0: print(Unrecognized command);
      reset(roll,pitch,yaw,throttle);
      Case 1: print(Freeze command);
      set flight mode Brake;
      reset(roll,pitch,yaw,throttle);
      Case 2: print(Turn on command);
      set flight mode Guided;
      arm_throttle(true);
      Case 3: print(Land command);
      set flight mode Land;
      Case 4: print(Take off command);
      set flight mode Guided;
      arm_throttle(true);
      take_off();
      Case 5: print(Return to Launch command);
      set flight mode RTL;
      arm_throttle(false);
      Case 6: print(Up command);
      verify throttle limits and increase;
      Case 7: print(Down command);
      verify throttle limits and decrease;
      Case 8: print(Front command);
      verify pitch limits and increase;
      Case 9: print(Back command);
      verify pitch limits and decrease;
      Case 10: print(Right command);
      verify roll limits and increase;
      Case 11: print(Left command);
      verify roll limits and decrease;
      Case 12: print(Right turn command);
      verify yaw limits and increase;
      Case 13: print(Left turn command);
      verify yaw limits and decrease;
    end
    generate RemoteControlMessage;
    RemoteControlMessage.set(roll,pitch,yaw,throttle);
    publish the RemoteControlMessage on topic mavros/rc/override;
  end
end

```

Algorithm 5: UAV Control subsystem pseudocode

the next motor command from the APM. Afterwards, Gazebo steps forward the simulation by 2.5 ms (for a 400 Hz update rate) and sends back new sensor measurements to the virtual APM. In this specific simulation, the virtual APM is the master of the simulation clock. The interaction between these components is better explained in Figure 3.15.

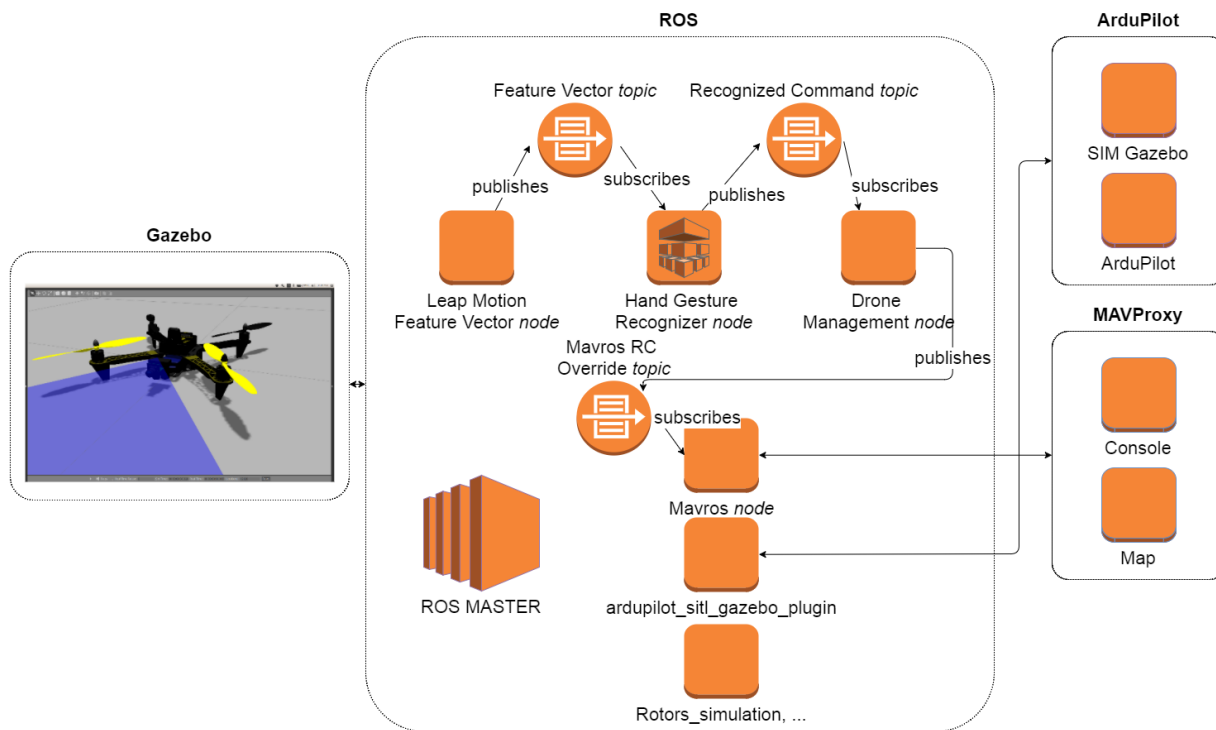


FIGURE 3.15: Communication between virtualized components and nodes in the simulation.

The node graph of the simulated environment and the recognition system is displayed in the Figure 3.16, as expected it holds the same nodes as Figure 3.14 adding the corresponding Gazebo nodes for the simulated UAV.

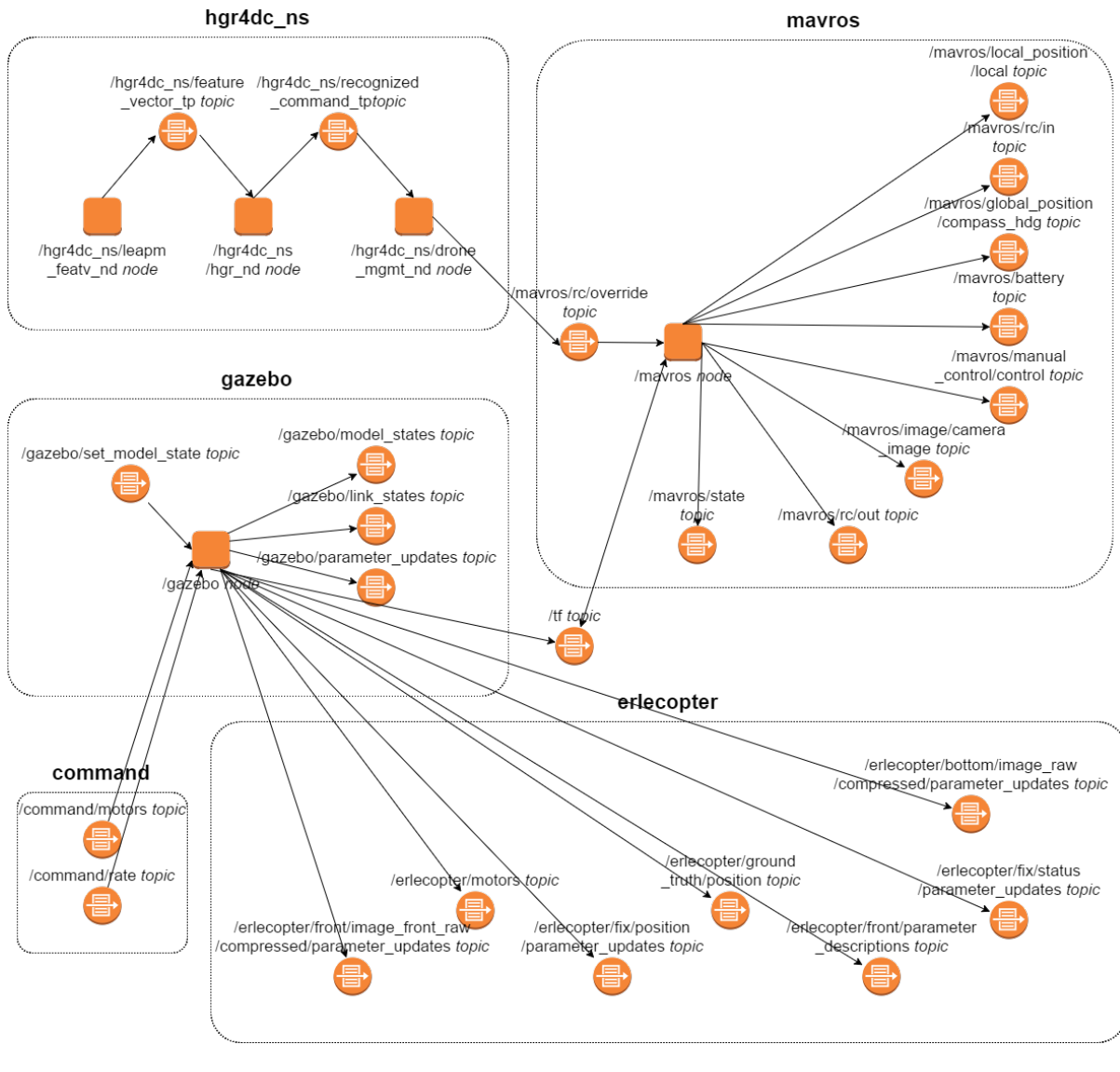


FIGURE 3.16: Node graph of simulated system.

Finally, a screenshot of the simulation environment is displayed in Figure 3.17. An orange square frames the Gazebo visualizer displays how the simulated UAV looks (the blue screen is the UAV field of view), the green frame shows how the Ground Control Station interacts with the simulation (MavProxy is the name of the GCS), whereas the purple and blue frames show the output of the *drone_mgmt* and *leapm_featv* nodes respectively, and last but not least, the yellow frame shows the output that the simulated APM provides.

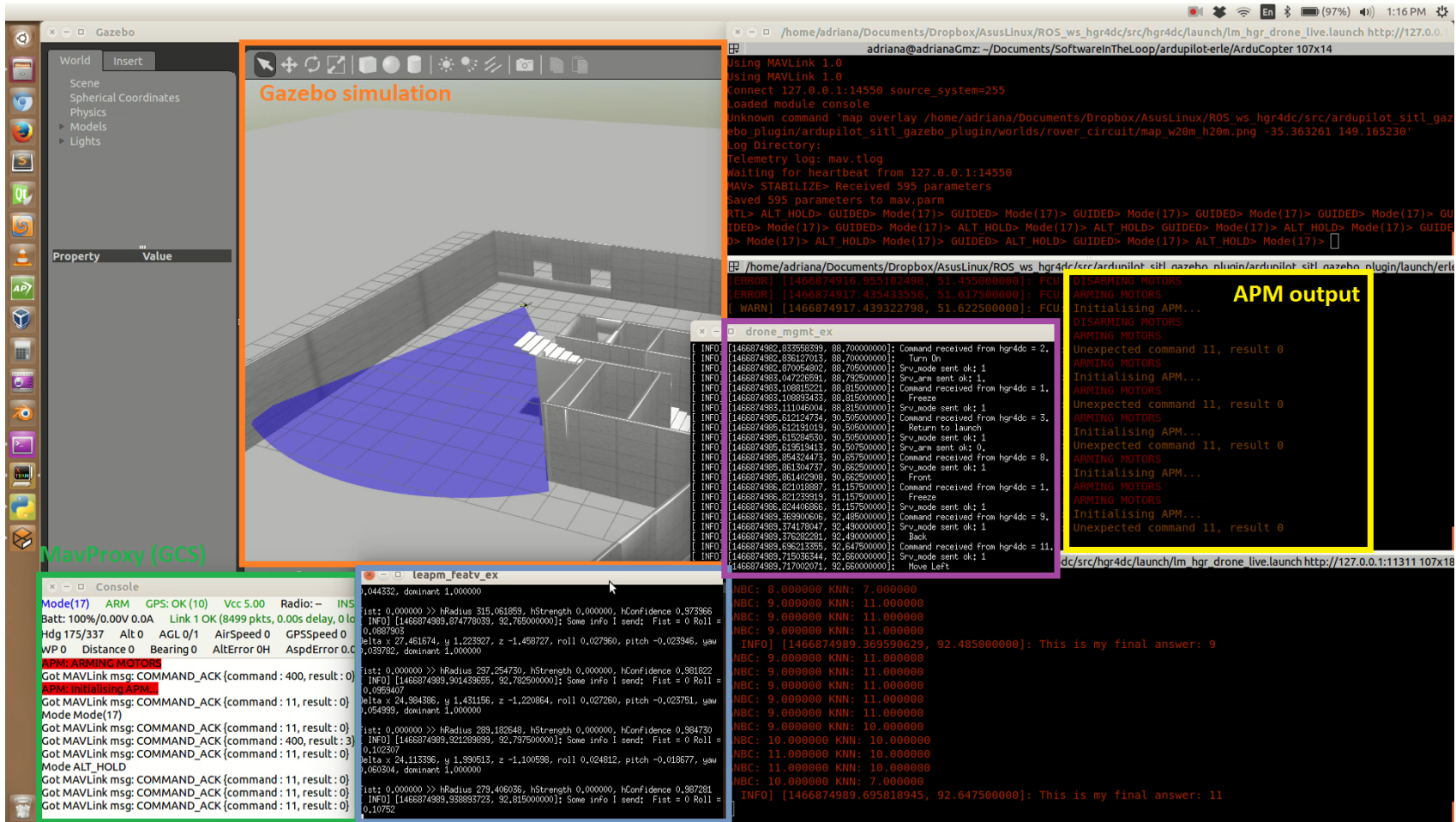


FIGURE 3.17: Simulation of the system in Gazebo.

Chapter 4

Experiment

The present Chapter contains the explanation of the driven experiments in order to measure the recognition accuracy response and analyze the time behaviour. Section 4.1 holds the specifications of the experiment, hardware and software tools and some other considerations. Section 4.2 contains the results from the gestures recognition evaluation. This was executed in two ways: first in a *static* form, where just the recognition accuracy was measured, followed by a *dynamic* test using live data. In this last evaluation, the accuracy and time response were measured and compared in order to elect the best base learner for the ensembled design.

4.1 Experiment Design

As explained earlier in Section 2.7, given the fact that both static and dynamic gestures are meant to be recognized, the first proposal for base learners was composed by the **temporal classifiers** HMM and DTW. However, the combination of gestures nature led to malfunctions in the algorithms principles, due to the lack of temporal variation in the static gestures. Although the temporal classifiers displayed their advantages such as the flexibility in the incoming data moderated training time and quick response, the inclusion of the **static gestures** generated unexpected recognized classes and therefore, the usage of temporal classifiers was discouraged.

The inclusion of the **temporal approach** (Section 3.2.3) enabled the usage of **static classifiers** for the recognition of both gestures type: static and dynamic, setting up as next challenge, the election of the two *best* base learners along with their *best* training data and *best* feature vector.

As mentioned in Figure 2.9, a traditional supervised learning implementation considers an iterative process. In the case of the present project, this workflow differed due to the addition of an extra test: a **dynamic evaluation** (or live validation).

Figure 4.1 shows the followed process in order to determine the best feature vector for the best base learners along with their best training data for *this implementation*. The election of these three best components is tied up among them, since there is no warranty that the feature vector that works best for a certain algorithm, works as good with some other algorithm, and the same happens with the training data, there is no warranty that some training data generalizes the problem perfectly for every classifier. This, and the dynamic evaluation, leaded to the situation of turning the regular process into an iterative one with more stages, explained as

follows.

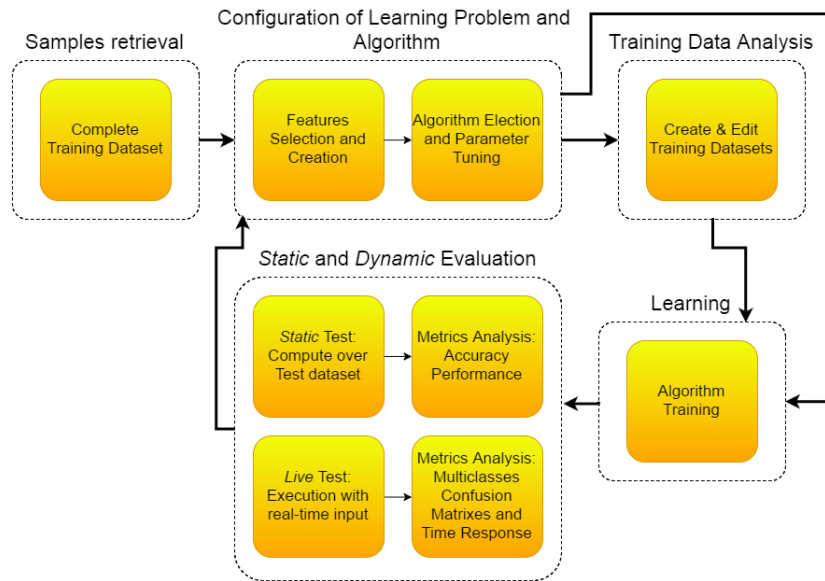


FIGURE 4.1: Workflow of current supervised learning algorithm implementation.

First step was to generate a *complete training dataset*. In order to generate these files, a program was developed to record samples with the information of interest from the Leap Motion frames, these files and data are explained in Section 4.2.2. As mentioned in Section 3.2.1, the Frames contain a whole set of information from which the next task is to *select* the most descriptive features and/or *create* the new required *features* (which led us to the *temporal approach*). This features election is followed by their test in an algorithm along with its *parameter tuning*. As a natural consequence of the supervised learning algorithms implementation, the algorithm gets *trained*, and as a supporting measure, a *training data analysis* was driven. This analysis was highly required in order to determine if the classifier accuracy was motivated by the algorithm itself, or induced by some tendency in the retrieved data (This analysis is also detailed in Section 4.2.2). After these tied phases, an *evaluation* stage was generated. This was composed by two parallel modes called *Static* and *Dynamic* evaluations, and owe their names to the data nature. In the **static evaluation**, the data belonged to the test set (retrieved from the 20% of the complete Training Dataset), whereas in the **dynamic evaluation** (or live validation), the data was provided by the real-time input from Leap Motion. In both cases, the accuracy performance was measured, and only in the live test, time response was included as well.

Due to the fact that nowadays, there exists a huge offer of machine learning algorithms implementation and libraries, it was decided to employ the **Gesture Recognition Toolkit** (GRT) as libraries to manipulate the classifiers. The GRT [57] is an open source C++ project which has been specifically designed for real-time gesture recognition and provides with many machine learning algorithms implementations and their corresponding parameters access.

The results shown in following sections were obtained in a computer with the following characteristics:

- PROCESSOR / CHIPSET
 - CPU: Intel Core i7 (4th Gen) 4500U / 1.8 GHz
 - Max Turbo Speed: 3 GHz
 - Number of Cores: Dual-Core
 - Cache: 4 MB (Installed Size)
 - Architecture: 64-bit Computing
- RAM
 - Size: 8GB
 - Technology: DDR3 SDRAM
- Operating System: Ubuntu 14.04.4 LTS, Trusty Tahr, 64 bits

4.2 Evaluation Description

The recognition of the gesture is a critical step in the complete system, therefore, an exhaustive series of tests were performed in order to get the best performance possible. Although a large amount of different attempts and evaluations were conducted during the development of this project, following sections show *only* the final and most important tests and results.

Chapter 2.4 introduced the evaluation tools and mechanisms presented in this section, therefore it is worth to mention that although only the *classical* methods are explained over there, some variations to the tools were developed in pursuance of a better results understanding and analysis, and are briefly explained as follows:

- **Static evaluation:** This is the classical mechanism for determining the performance of a classifier and was briefly explained in Section 2.5.4, it is the accuracy response with data from the test set.
- **Dynamic evaluation:** While the test set evaluation shows the classifier performance for a previously known data, the nature of this project required to test the accuracy with unknown data. Therefore, Section 4.4 presents the **live data test**, where the individual accuracy and time response are compared when they receive *live data* from Leap Motion. It is the accuracy response with live data.
- **Confusion matrixes.** As explained in subsection 2.5.4, these matrixes were originally developed for *binary* classifiers. Since they present the information in a concise and useful way, Section 4.4 makes use of them in a version where they present percentage results for the *multi-classes* classifiers used in this project and enhance the data analysis.

As we know, the Peaking Phenomenon (Section 2.5.3) is generated by an incorrect election of either the feature vector or the training data. In order to avoid this undesired behaviour, several test executions were performed to achieve the response we needed.

Basically the tests were driven in four dimensions expressed in the following list:

- Machine learning static algorithms (classifiers) tested (based on the analysis and comparisons showed in Section 2.6):
 - Adaptive Naive Bayes Classifier (ANBC).
 - Minimum Distances (MD).
 - Support Vector Machines (SVM).
 - K-Nearest Neighbors (KNN).
- Size of the Training Data: Number of training samples provided for the classes to be recognized, which could lead to a under/over training.
- Feature Vector: Number and combination of features (**How many** and **which** features generalize the problem in the best way).
- Required time to deliver a recognized class.

4.2.1 Training Data

Due to the over/under training problem, it is necessary to test the classifiers with different training data **sizes**, this lets us know which data set generalizes *best enough* the recognition problem (Section 2.5.3). In this context, *size* refers to the *number of samples* that each target class counts with and consequently the total number of samples per set.

The training data was generated by us, through the usage of a developed piece of software that retrieved the data corresponding to a gesture. The program creates an array that holds the target class of the gesture to be recorded, retrieves the information from the frames, filters the data (Section 3.2.2), generates the complete feature vector (Section 3.2.3), stores it in the array and saves the information in a text file. A sample of a training data text files for the full feature vector is shown for the gesture labelled as six (Up movement):

```

DatasetName: TrainingData_80D.txt
NumDimensions: 14
TotalNumExamples: 2701
Data:
...
6 -3.891090 63.118100 17.611600 -43.636800 65.272600 -14.094800 0.000000
-0.055208 -0.157330 -0.057181 -0.065460 -0.122833 0.014272 0.001159
6 -3.932880 63.001500 17.667800 -43.663000 65.227000 -14.085900 0.000000
-0.079023 -0.222164 -0.252977 -0.069847 -0.125015 0.009707 -0.001639
...

```

The first element of each data row is the target class c_j that corresponds to the recorded gesture (Table 3.1). Afterwards, the fourteen following values correspond to the full feature vector x . Therefore, each sample (row) of the training data file may be expressed by the target class followed by the feature vector:

$$\text{sample of } \mathbf{X} = [t_{\text{recorded gesture}}, x_1, x_2, \dots, x_{14}] \quad (4.1)$$

Table 4.1 shows each training data set labeled as **TD1**, **TD2**, **TD3** or **TD4** in its increasing order, along with the number of samples that each class has. This is important to know due to the fact that the classes should ideally be equally trained (i.e. every class should have the same number of samples), however, due to the way the samples were recorded, this was impossible to assure. Taking as example the extreme datasets, TD1 is the set with less number of samples per class (around 300) while TD4 is the *biggest* set holding around 1000 training samples per class. Although these results display only four datasets, many more were previously recorded in order to count with an initial approximation to the training data size that started to show acceptable accuracies. Counting with these different set sizes, allowed us to test the effect of over/under training and the algorithm’s capability to deliver a model that generalizes the training data for the requested classes (Section 2.5.1).

T. data name	Sample nr for $t = 6$	Sample nr for $t = 7$	Sample nr for $t = 8$	Sample nr for $t = 9$	Sample nr for $t = 10$	Sample nr for $t = 11$	Sample nr for $t = 12$	Sample nr for $t = 13$
TD1	337	412	337	269	419	310	391	226
TD2	454	338	410	396	349	300	377	305
TD3	501	522	491	511	478	519	488	503
TD4	1054	984	1382	1523	954	926	1132	1161

TABLE 4.1: Sample number per training data set and per class c_j .

Table 4.2 summarizes the total samples number per each set, giving an idea of how much *complete* knowledge the tests were provided with.

Training data name	Total number of samples
TD1	2701
TD2	2929
TD3	4013
TD4	9116

TABLE 4.2: Total sample number per training data set.

4.2.2 Feature Combinations

The next dimension to test is regarding the Feature Selection problem (Section 2.5.1) and it is comprised by two questions:

- **How many** features should the feature vector contain? and
- **Which** features are the most descriptive and useful for the algorithm?

Both questions should be answered having in mind the same goal: To select the features that allow a good generalization of the data that represents the same class. In order to solve both questions at the same time, four feature **combinations** created four different feature vectors and

are explained in this section (although they were not the only proposals, the displayed here are the bests).

Table 4.3 summarizes the combinations, naming them as **A**, **B**, **C** and **D**. These combinations were proposed in order to test the algorithm generalization capability with single and double reference points to be tracked and the inclusion (or lack) of the feature x_7 : *Vector of Change*. Tested features were chosen upon the utility of the reference points coordinates and the information they provided according to the delta values presented in Section 3.2.3. As the Table displays, combination **A** keeps track of the single reference point (the palm center), while Combination **C** is its extension adding the feature x_7 . Combination **B** is the arrangement that keeps track of two reference points (the palm center and the thumb tip) and Combination **D** is the B, including the Vector of Change. As it may be easily deduced, combination D is the *full feature vector* and therefore its size corresponds to the fourteen features previously showed in Section 3.2.3.

Combination name	Included features	Feature vector x	Size of feature vector x
A	Palm center (position and Δ values), ΔY_{aw}	$x = [x_1, x_2, x_3, x_8, x_9, x_{10}, x_{14}]$	7
B	Palm center (position and Δ values), Thumb tip (position and Δ values), ΔY_{aw}	$x = [x_1, x_2, x_3, x_4, x_5, x_6, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}]$	13
C	Palm center (position and Δ values), Vector of Change, ΔY_{aw}	$x = [x_1, x_2, x_3, x_7, x_8, x_9, x_{10}, x_{14}]$	8
D	Palm center (position and Δ values), Thumb tip (position and Δ values), Vector of Change, ΔY_{aw}	$x = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}]$	14

TABLE 4.3: Feature vector combinations.

4.2.3 Predominant Features

To finalize this introduction to the metrics analysis, it is necessary to list the predominant features depending on the gesture to be recognized. As displayed in Table 3.1, each gesture was matched to a class c_j .

From the nature of the gestures, it can be seen that certain features may play a more important role for certain gestures than for others. Let us mention class 6 for instance (corresponding to the Up movement). This movement provides more useful information in the Y axis (see Figure 3.4) than in some other feature (unlike data from the Yaw rotation for instance). Figure 4.2 shows that for this gesture, the *predominant features* are the related to the Y axis, which according to the list shown in Section 3.2.3, are features x_2 , x_5 , x_9 and x_{12} (the palm and thumb Y position and delta values). The values obtained in these features show greater variation than others.

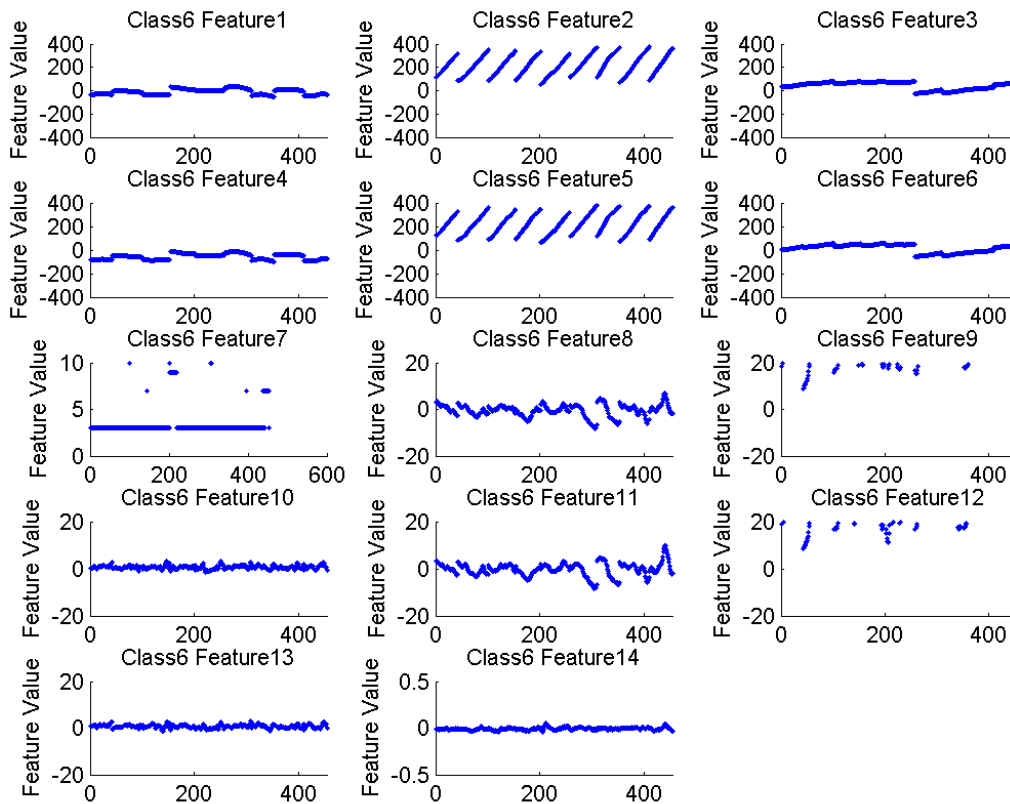


FIGURE 4.2: Data captured for Up gesture.

To understand the comparison, let us take now the gesture of left rotation, corresponding to class 13. With aid of Figure 4.3, we may compare against Figure 4.2 and verify that the features that played a important role for class 6, show no meaningful change for class 13, and moreover, since class 13 depends much on the Δ yaw angle (feature x_{14}) due to the movement nature, this

same feature x_{14} plays no important role for class 6.

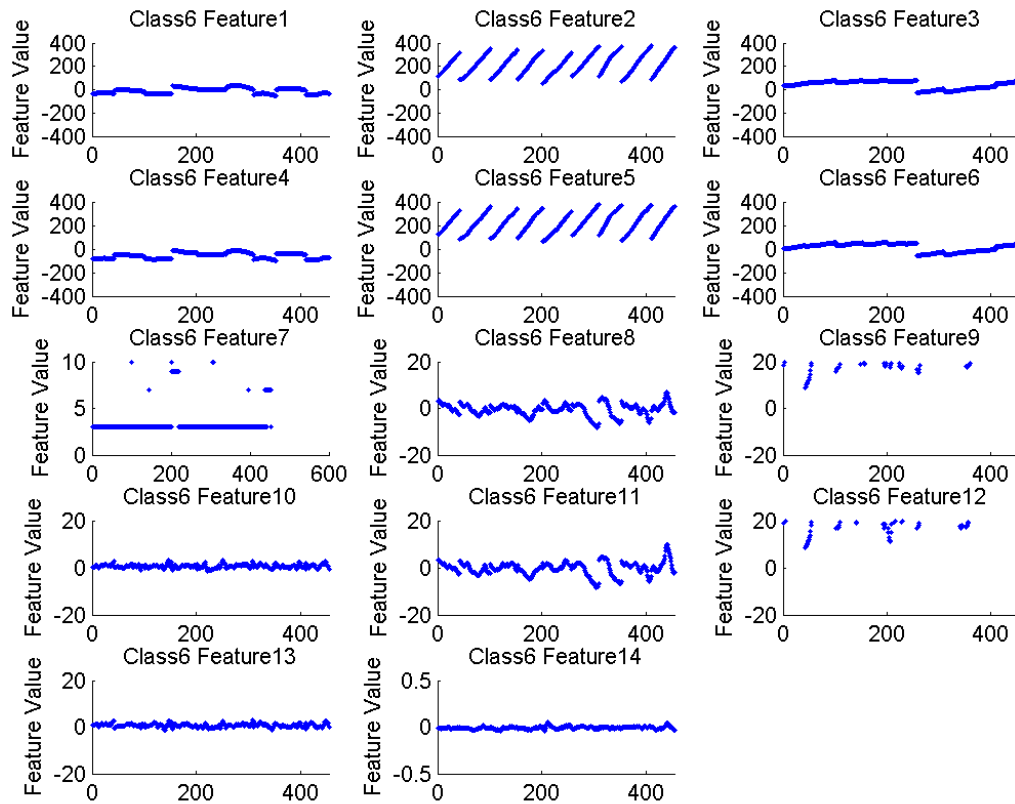


FIGURE 4.3: Data captured for left rotation gesture.

4.3 Static Tests: Test Subset Accuracy Response

As described in Section 2.5.3, the test set validation consists of the recognition evaluations executed over the test set: a subset previously retrieved from the original training data. The test subsets are composed by 20% randomly taken samples from the full training data set, leaving the rest for training purposes. In the current implementation, these methodology may be categorized as a **static** evaluation, due to the fact that the tested data is known in advance.

As explained in earlier sections, the tests were executed by training data set (1, 2, 3 and 4), by classifier algorithm (ANBC, MinDist, SVM and KNN) and by feature vector (features combination).

4.3.1 Class Accuracy

Let us show an example of one of these test results chart in order to understand their significance. For display purposes, accuracies for only classes $t_6 : t_{13}$ are shown, since they correspond

to the dynamic movements and are the angular stone of the project. Table 4.4 displays the results for the recognition accuracy, in the dynamic gestures, as output of the evaluation with the four training data sets for every classifier, with the combination D as the feature vector x .

Each column represents the target class c_j (to be recognized) and each row is the arrangement of training data and classifier used at that time. The cell obtained as intersection between the *Training Data* row and the *Class accuracy* column, contains the accuracy percentage of the class prediction (or classification). Last column, *Average accuracy*, holds the average of accuracies between the classes $t_6 : t_{13}$, i.e. the average of the row. **Accuracy** was defined in Section 2.5 as a *measure of how well the selected decision reflects the right decision*. For this analysis, accuracy for each class is measured as the percentage of **True Positive** classifications in each class. A True Positive (TP) is considered when the classifier *correctly* assigns the right class to the tested sample (Proper assignment of class c_j to the evaluated data). Hence, accuracy for each class c_j ($Accuracy_j$) is expressed by the sum of the number of TP divided by the total number of tested samples for class c_j (n_j), and represented by the Equation 4.2.

$$Accuracy_j = \frac{TP_j}{n_j} \quad (4.2)$$

Average accuracy (Equation 4.3) is the measure we used to qualify a classifier in general terms and is obtained as the mathematical mean of the classes accuracy ($Accuracy_j$) for the tested classes ($N = 6 : 13$).

$$\overline{Accuracy_j} = \frac{\sum_{j=6}^{13} Accuracy_j}{N} \quad (4.3)$$

Table 4.4 provides an overview of how well the static classifiers perform their job; the chart is composed by four subsections where each one belongs to the accuracy results of each tested algorithm (ANBC, Min Dist, SVM and KNN) and discussed in following lines. From the comparison of all of them and at first sight, KNN is the algorithm which presented the best accuracy to almost every training datasets. It does not create a model of the data, but classifies *on the fly*, hence, the training data is not altered and classifies only by comparing. This algorithm counts with just one parameter (K) which makes it a very simple approach for static purposes. In the same Table 4.4, although MinDist counts with a similar heuristic about not transforming the training data, this algorithm showed lower $Accuracy_j$. This was result of the similarity between the training samples, remembering the working principle of this algorithm, MinDist creates a chart containing the euclidean distances (in this case) from the tested sample to the training data. If the Training data are very similar among classes (low inter-class distance, Section 2.5.1), the algorithm will find more than one good candidate for the tested sample.

A more interesting response is the one obtained from the ANBC and SVM algorithms. Since both of them *do* generate a model from the training data, their prediction is completely influenced by the training samples nature. This means that for ANBC, the ability to generate a probability density function from the data, gets the model directly affected; hence, the data needs to comply with the independent probability condition for each event (which indeed happens for these dynamic gestures and their training data). In juxtaposition, SVM and its linear kernel transformation get rid of the probabilistic assumptions. Thanks to the margin maximization, this algorithm exhibits a better inter-class distance which compensates in great manner the inconsistent data retrieved by Leap Motion and delivers the high accuracy displayed in the Table.

Combination D										
Accuracy for Test Data										
ANBC										
		Class accuracy (%)								Average accuracy (%)
		6	7	8	9	10	11	12	13	
Training Data	1	80.00	100.00	97.50	100.00	100.00	96.77	95.71	100.00	96.25
	2	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
	3	99.00	100.00	97.92	98.97	100.00	100.00	98.92	99.07	99.23
	4	100.00	99.55	97.79	100.00	98.84	98.92	97.86	98.69	98.96

Min Dist										
		Class accuracy (%)								Average accuracy (%)
		6	7	8	9	10	11	12	13	
Training Data	1	100.00	98.98	100.00	97.87	100.00	98.53	100.00	100.00	99.42
	2	99.02	98.36	100.00	100.00	100.00	100.00	100.00	100.00	99.67
	3	92.39	98.26	96.74	100.00	76.29	71.00	97.92	94.23	90.85
	4	99.51	99.49	99.24	98.75	99.47	98.32	98.70	100.00	99.18

SVM										
		Class accuracy (%)								Average accuracy (%)
		6	7	8	9	10	11	12	13	
Training Data	1	95.95	100.00	100.00	100.00	98.68	98.08	97.40	100.00	98.76
	2	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
	3	100.00	100.00	99.03	100.00	100.00	99.11	100.00	100.00	99.77
	4	100.00	100.00	98.96	99.30	100.00	99.47	98.26	99.55	99.44

KNN										
		Class accuracy (%)								Average accuracy (%)
		6	7	8	9	10	11	12	13	
Training Data	1	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
	2	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
	3	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
	4	100.00	100.00	100.00	99.69	100.00	100.00	100.00	100.00	99.20

TABLE 4.4: A sample of the results in the static validation: Test set accuracy for dynamic gestures with the Combination D.

Previous Table is just a sample, indeed, the same exercise was made with the four feature combinations (A, B, C and D). Table 4.5 displays the summary of all the static evaluations using the four feature vectors, and for displaying purposes, the chart shows only the average accuracy $\overline{Accuracy}_j$ for each classifier in each combination with each training data set.

It becomes clear that when two reference points are tracked, the results are much better. This is verified by the outputs obtained in Combination B and D in comparison with the retrieved in Combination A and C. From this pair, Combination C behaved in general slightly better than A due to the inclusion of the extra feature, while Combination D showed the same advantage for the lazy classifiers (MinDist and KNN) but decreased slightly the performance for the algorithms that create a model (ANBC and SVM). The over/under training effect may be also compared between the rows of each classifier, and it can be seen that while the same data set may not be a peaking problem for one arrangement (classifier and feature combination), it may greatly affect another configuration. That is the case (for instance) of Training Data 3, which turned to be a perfect training set for the KNN-Combination B, but the worst for the

MD-Combination A.

		<i>Combination A</i>	<i>Combination B</i>	<i>Combination C</i>	<i>Combination D</i>
		ANBC average accuracy (%)			
Training Data	1	94.69	97.19	95.05	96.25
	2	88.20	99.86	99.48	100.00
	3	99.38	98.99	97.44	99.23
	4	88.24	99.15	88.39	98.96
		MD average accuracy (%)			
Training Data	1	97.533	97.743	97.564	99.422
	2	96.996	99.380	97.279	99.672
	3	77.169	89.856	76.328	90.853
	4	92.635	98.111	92.957	99.184
		SVM average accuracy (%)			
Training Data	1	99.00	100.00	98.09	98.76
	2	99.63	100.00	99.85	100.00
	3	98.64	99.89	98.50	99.77
	4	88.51	99.50	90.26	99.44
		KNN average accuracy (%)			
Training Data	1	99.84	100.00	98.92	100.00
	2	99.17	100.00	99.28	100.00
	3	99.54	100.00	98.96	100.00
	4	98.52	99.93	98.50	99.86

TABLE 4.5: Average of accuracy results in the static validation.

In order to provide a better comparison experience, Figure 4.4 holds the graphical comparison for these averages values, exactly in the same way Table 4.5 did. This gives an idea of how well the classifiers work by showing how close their accuracies are to a perfect recognition (100%).

It is absolutely important to mention that just by looking at these results, the classifiers performance seem highly promising; apparently all the tested classifiers deliver recognition accuracies almost perfect. However this is only the response to static tests, and since the classification is targeted to control a mobile device, its behaviour for live data must be tested as well. This is demonstrated in following Section.

AVERAGE ACCURACY IN *STATIC* TEST

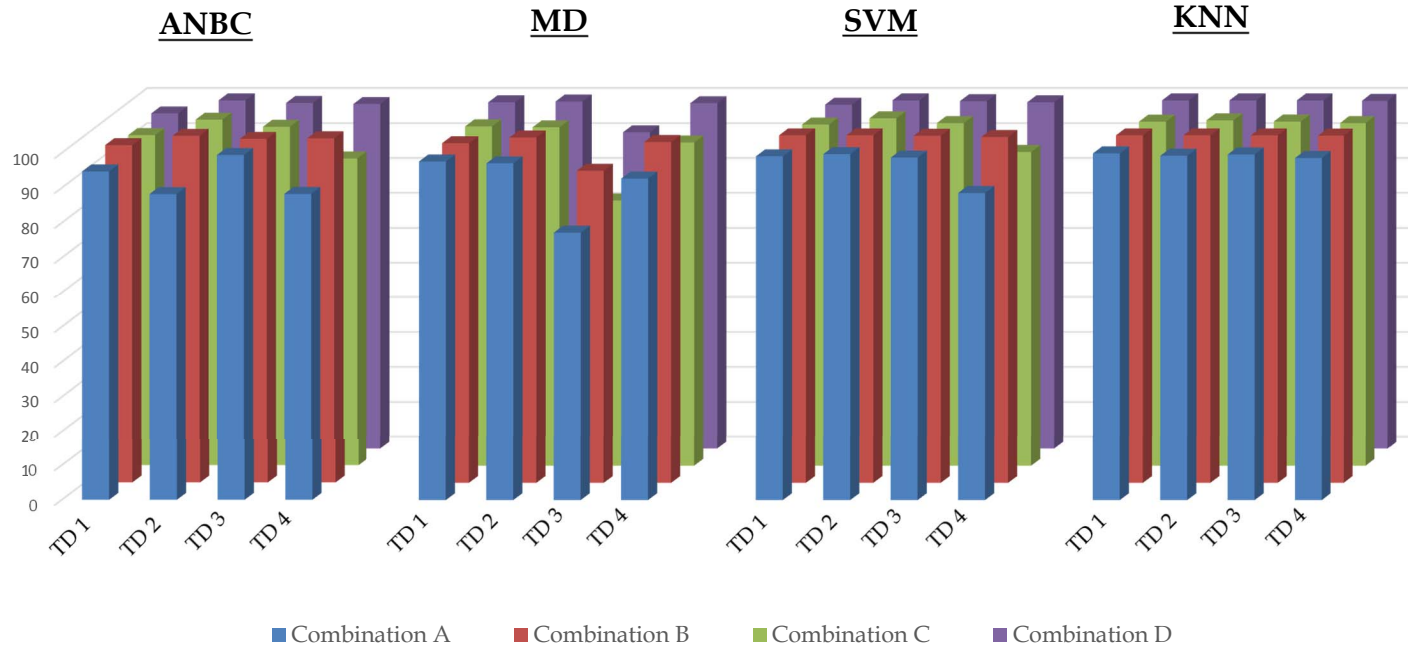


FIGURE 4.4: Comparison between average accuracy for each classifier in the test set validation.

4.4 Dynamic Tests: Live Data Accuracy Response

At first sight, accuracies presented in Table 4.5 seem highly promising. However, as mentioned earlier, these are tests performed with **pre-recorded** data (*ergo, static*) and when it comes to real-time systems, the weight of reality falls into place...

Truth is, when the classifiers were tested with *live* data, their predictions were highly affected and the optimistic expectations from the previous static tests served only as starting points and guidelines towards more realistic tests.

In fact, this is the reason why the temporal classifiers were discarded from this system implementation (Section 2.7) and the *temporal approach* proposed: although the temporal classifiers accuracy with static data was high, their behavior with live data was completely different, the accuracy fell down to 10% or below.

4.4.1 Multi-classes Confusion Matrixes

Taking the classical binary confusion matrixes as starting point, a program was designed in order to generate a modified version, called *multi-classes confusion matrixes*, used to measure the algorithms response to *live data*. This program was provided with the target class c_j of the hand movement to be performed, and recorded the classifiers response. Afterwards the **predicted** class (the outcome from the classifiers) was compared against the **expected** class (the real class corresponding to the hand movement) and the confusion matrix was generated in a similar way than the binary confusion matrixes (Section 2.5.4), explained as following.

The *multi-classes confusion matrixes* reveal the accuracy in percentage format (another difference from the classical binary) for each class. Its columns express the **predicted** class, its rows contain the **expected** class and the intersecting cell shows the percentage of their coincidence (if the predicted class was the same that the expected class). To fill the matrix, the pseudocode of the algorithm is displayed in Algorithm 6.

In order to better describe what they represent and how they were used, let us present a sample of a set. Table 4.6 shows four confusion matrixes belonging to the four training data sets, just for the ANBC algorithm with the feature combination B.

An ideal confusion matrix should contain 100% values across the diagonal (the predicted class matched the expected class), whereas a complete row should add up 100%. Class 0 acts as a wildcard and is obtained when the classifier can not recognize the gesture as one of the known classes (from six to thirteen). Let us analyze the case of the first confusion matrix, for the expected class 11. As wished, the correspondence to the column 11 shows a 94, it means that 94 % of the times, the live data corresponding to class 11 was rightly classified as class 11. However, 4% of the times it was missclassified as class 13 and 2% of the times, the gesture could not be recognized at all (Class 0). Since the rows hold the expected class, the missclassification analysis throws very useful information (explained in following paragraphs). Last column in each matrix contains the accuracy average retrieved from the main diagonal, which at the end

Combination B											
Accuracy for Live Data: Confusion Matrix											
ANBC											
		Output Class (classifier's response)									
		0	6	7	8	9	10	11	12	13	
Training data 1	Expected Class (Assigned in the Training Data)	0	6	7	8	9	10	11	12	13	Diagonal average accuracy (%)
	0	100	0	0	0	0	0	0	0	0	91
	6	2.00	98.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	7	0.00	0.00	100.00	0.00	0.00	0.00	0.00	0.00	0.00	
	8	6.00	52.00	0.00	40.00	0.00	0.00	0.00	0.00	2.00	
	9	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00	
	10	0.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	
	11	2.00	0.00	0.00	0.00	0.00	0.00	94.00	0.00	4.00	
	12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00	
	13	2.00	0.00	0.00	0.00	2.00	0.00	0.00	0.00	96.00	
Training data 2	Expected Class (Assigned in the Training Data)	0	6	7	8	9	10	11	12	13	
	0	100	0	0	0	0	0	0	0	0	95.25
	6	2.00	96.00	0.00	2.00	0.00	0.00	0.00	0.00	0.00	
	7	4.00	0.00	96.00	0.00	0.00	0.00	0.00	0.00	0.00	
	8	4.00	0.00	0.00	96.00	0.00	0.00	0.00	0.00	0.00	
	9	2.00	0.00	0.00	6.00	92.00	0.00	0.00	0.00	0.00	
	10	12.00	0.00	0.00	0.00	0.00	88.00	0.00	0.00	0.00	
	11	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	
	12	2.00	0.00	0.00	0.00	0.00	4.00	0.00	94.00	0.00	
	13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	
Training data 3	Expected Class (Assigned in the Training Data)	0	6	7	8	9	10	11	12	13	
	0	100	0	0	0	0	0	0	0	0	75.75
	6	34.00	66.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	7	2.00	0.00	98.00	0.00	0.00	0.00	0.00	0.00	0.00	
	8	70.00	0.00	0.00	30.00	0.00	0.00	0.00	0.00	0.00	
	9	34.00	0.00	0.00	18.00	48.00	0.00	0.00	0.00	0.00	
	10	20.00	0.00	0.00	0.00	0.00	80.00	0.00	0.00	0.00	
	11	2.00	0.00	0.00	0.00	0.00	0.00	98.00	0.00	0.00	
	12	10.00	0.00	0.00	0.00	0.00	0.00	0.00	88.00	2.00	
	13	2.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	98.00	
Training data 4	Expected Class (Assigned in the Training Data)	0	6	7	8	9	10	11	12	13	
	0	100	0	0	0	0	0	0	0	0	97.75
	6	0.00	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	7	2.00	2.00	96.00	0.00	0.00	0.00	0.00	0.00	0.00	
	8	2.00	0.00	0.00	88.00	10.00	0.00	0.00	0.00	0.00	
	9	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00	
	10	0.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	
	11	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	
	12	2.00	0.00	0.00	0.00	0.00	0.00	0.00	98.00	0.00	
	13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	

TABLE 4.6: A sample of the results in the dynamic validation: Live data accuracy for dynamic gestures with the Combination B for algorithm ANBC.

input : File with live data classifications

output: Multi-classes confusion matrixes

Classifications retrieval from the file;

Storage of classifications in the data structure TrainingSample;

Creation of square matrix *confusionM_rate* as zeros(numClasses,numClasses);

```

for class=1:numClasses do
    totalMembers = How many samples were recorded for all classes in total (right and
        wrong predictions);
    vectorConfM = Accumulates how many votes each Output class adds in each
        Expected class;
    nrElems = How many samples were recorded per class;
    for row = 1:nrElems do
        for col = 1:numClasses do
            retrieved = trainingSample(row, col,class);
            if retrieved = 0 then
                | vectorConfM(retrieved)++;
            end
        end
    end
    confusionM_rate(class,:) = (vectorConfM/totalMembers)*100;
end

```

Algorithm 6: Pseudocode for the multi-classes confusion matrixes creation.

may be summarized as the *integrated* accuracy.

Reasons why the classifier algorithm could not assign the live data to a known class are different. One of them is that the entered data to the algorithm were completely wrong values (feature vectors full of zeros) due to Leap Motion failures, therefore, the classifier was not able to categorize the incoming feature vector in neither class due to lack of information.

These matrixes helped to understand better why the classifiers failed, for instance, when two continuous classes are confused among each other, it may exist a correlation between their training data values and such. Retaking the Table 4.6, the matrixes belonging to the training data 2, 3 and 4 show this behaviour in the predicted classes 8 and 9. It can be seen that classes 8 and 9 are confused among them. The reason of this is because class 8 and 9 belong to the front and back hand gesture and the predominant features are x_3 , x_6 , x_{10} and x_{13} , i.e. the features related to the Z direction (Section 4.2.3). A brief discussion about the problem of the Leap Motion Z coordinates estimation has already been presented in Section 3.2.2. Therefore, the results of a misclassification may be studied to gain understanding about some data problems.

Likewise in the test set validation, the live data response was tested using the same feature vector combinations, training data sizes and the four algorithms, hence, for the complete analysis, sixty four confusion matrixes were generated and studied: the four algorithms with its four features combinations for each training data set. From their analysis, interesting things could be understood for the recognition problems, and just as a quantitative measure, the diagonal average accuracy of each matrix is displayed in Table 4.7.

		Combination A	Combination B	Combination C	Combination D
		ANBC average accuracy (%)			
Training Data	1	74.50	91.00	68.25	96.75
	2	97.75	95.25	96.75	94.25
	3	73.75	75.75	71.00	70.50
	4	65.00	97.75	69.00	94.75
		MD average accuracy (%)			
Training Data	1	65.250	80.000	64.750	92.250
	2	83.250	86.750	77.500	85.500
	3	54.500	67.500	58.250	64.000
	4	62.500	95.500	57.500	93.250
		SVM average accuracy (%)			
Training Data	1	96.75	95.00	95.50	94.00
	2	91.50	97.75	87.75	92.00
	3	73.75	79.00	78.75	78.25
	4	80.00	89.50	79.75	95.50
		KNN average accuracy (%)			
Training Data	1	77.50	90.25	81.25	95.00
	2	72.00	96.25	79.25	87.50
	3	67.25	74.00	63.00	76.75
	4	83.25	90.75	69.00	98.00

TABLE 4.7: Average of accuracy results for the main diagonal in the live validation.

Two more useful details were depicted from the matrixes and are explained as follows: Since they show if a gesture was mistakenly classified as another one, the matrixes display exactly which one was the missclassified gesture. This helps figuring out the reason why the gesture was not properly recognized. Let us say for instance, if similar movements with opposed directions were missclassified among them (like the up-down or left-right couple), the reason was that the necessary characteristics were properly saved, but the temporal features did not reflect correctly the movement *sense*. The second extra feature of these matrixes is that they allow to know the error generated by the hardware itself. This can be read in the matrixes row; it is easily recognizable that every row must add up 100% (the most desirable is to have 100% in the cell that matches the recognized class with the desired class), however, in the reality of our experiments this did not happen. The additions did not sum up the desired 100%, which meant that the classifier could not assign the incoming data to any of the available classes. This could make think that at that moment, Leap Motion was delivering some zeroed Frames that obligated the classifiers to deliver a non-recognized class.

From all this, it could be expressed that the inclusion of these modified confusion matrixes achieves three major and useful functions: they allow to know the classifier's accuracy, delivers information with clues about the missclassification reasons, and finally, gives an idea of how much anomalous information is delivered by the hardware. These functions were meticulously employed and analyzed in the results evaluation towards the following four goals: calibrate/tune the classifiers parameters, improve the Training Datasets everytime they were recorded, choose the most meaningful characteristics for the feature vector, and finally, design

the final recognizer.

In similar way as in the test set validation section, Figure 4.5 summarizes the comparison for **just** the accuracy averages across the main diagonal for the live validation tests. This time, the classifiers do not show a *quasi* perfect recognition because now they are facing real conditions. In this graph, it can be seen how well an algorithm generalizes the training data in order to provide a classification for unknown data (or live data). It is easy to see that now, ANBC, SVM and KNN are the classifiers that approach closer to the perfect recognizer (100%) with the Combinations B and D, and Training Data 2 and 4.

Now that the best classifiers start to arise, is time to include the last important component to the algorithm election: its *time* response.

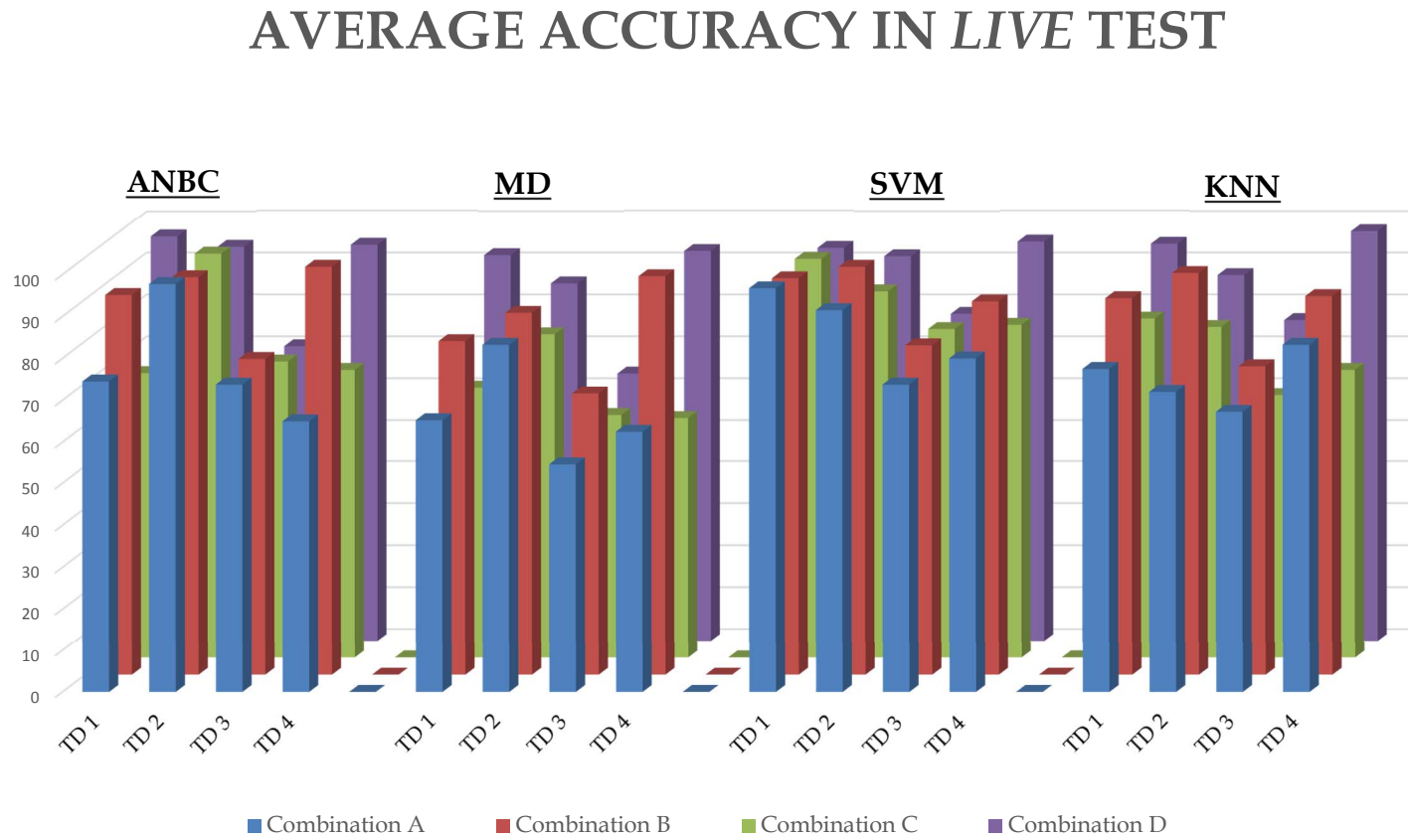


FIGURE 4.5: Comparison between average accuracy for each classifier in the live validation.

4.5 Dynamic Tests: Live Data Time Response

Last but not least, another highly important factor to take in account for the classifiers election is the time response. Table 4.8 holds the average time it takes to each classifier, under the different circumstances, to deliver the prediction value, expressed in milliseconds.

		<i>Combination A</i>	<i>Combination B</i>	<i>Combination C</i>	<i>Combination D</i>
		ANBC average response time (ms)			
Training Data	1	0.04	0.19	0.07	0.07
	2	0.05	0.13	0.05	0.09
	3	0.09	0.13	0.08	0.08
	4	0.06	0.11	0.09	0.06
		MD average response time (ms)			
Training Data	1	0.016	0.066	0.045	0.035
	2	0.024	0.041	0.024	0.045
	3	0.043	0.063	0.031	0.038
	4	0.059	0.066	0.044	0.038
		SVM average response time (ms)			
Training Data	1	0.24	0.67	0.29	0.25
	2	0.46	0.34	0.53	0.18
	3	0.66	0.25	0.46	0.23
	4	1.74	0.91	0.96	1.97
		KNN average response time (ms)			
Training Data	1	2.37	5.68	4.29	5.12
	2	5.74	5.18	5.76	6.02
	3	7.37	9.28	6.56	9.51
	4	14.43	19.59	19.17	22.49

TABLE 4.8: Average time response (in milliseconds) of classifiers across combinations in the live validation.

Although KNN had been presenting very good classification results, due to the fact that it does not properly create a model of the data (it is a lazy learner), it takes a long time in order to provide a classification. This discards completely its inclusion into the final base learners. On the other side, MinDist is the fastest classifier due to the simple euclidean distance it generates. After MinDist, ANBC follows as second fastest classifier, its success reason is because it does create a model from the training data and just turns the incoming data into a probability value, compare it against the already created probability functions and chooses the most likely class. Finally, SVM is the third fastest due to the usage of a linear kernel. This helps tremendously to the kernel creation by lightening the transformation function for the live data and compares with the already stored data model from the training set.

These response times were obtained in a laptop computer with the characteristics mentioned in Section 4.1.

Classifiers Election

Paraphrasing the *No free lunch theorem* (Section 2.5.3): There is no ideal classifier for all the classification tasks. In reality, it is a trade off between multiples factors, and for this work the factors were listed in subsection 4.2.

The results displayed in the previous sections showed some of this trade off, because although *Minimum Distances* and *ANBC* were the fastest classifiers, *Minimum Distances* was not able to even figure within an acceptable accuracy percentage. And although *KNN* showed the best accuracy from all the compared classifiers, its response time was more than 200 times slower (in comparison with *ANBC* for instance) due to the fact that *KNN* does not create a learning model per se, but compares the neighbors distances *on the fly*.

Therefore the elected classifiers that respond with the best accuracy possible in a timely manner, are:

- **ANBC**. Using TD4 and Combination B.
- **SVM**. Using TD2 and Combination B.

Both classifiers run in separate threads which leads to independent executions, as shown in Figure 3.10.

Chapter 5

Conclusions and further work

This document comprises the development of a Hand Gesture Recognizer towards the UAV direction movements control, implementing it over ROS. The core components of the recognizer are classifiers based on the Supervised Learning paradigm and run engaging the spirit of the Ensemble methodology. The manipulation of this system is provided by dynamic hand gestures, which are translated into a static approach, in order to be suitable to the base learners. The response of this system is visualized in the Gazebo environment, which is a ROS-based simulator.

The interest in developing such a system with the mentioned characteristics is explained in the introductory chapter, which is then followed by the theoretical frame required to understand the terminology and tools used throughout the current document.

Beginning with a small introduction to the Human Machine Interface Systems, the need to reach a consensus about what the *Real Time* term implies, obligated us to outline about the Real Time Control Systems, and in order to narrow down our goal study field, the Hand Gesture Recognition systems were reviewed along some current published applications. Since the *Leap Motion* controller was the elected hardware for this project implementation, some related works and proposals are mentioned in this same section as well.

Pattern Recognition is the great study field tackled in this project, therefore, a complete section is dedicated to a gentle introduction to the Gesture Recognition and Machine Learning tools. Since the nuclear components of the recognizer are Machine Learning algorithms, some Supervised Learning Algorithms are mentioned and compared along with some of their most important characteristics. This section contains a brief description about the terminology, approaches and mechanisms used in this work, which knowledge is essential for a full understanding of the algorithms evaluation, development of the recognizer and its implementation.

After the reader is well conditioned with the provided information, the System Design is presented, introducing the project idea with an overview of the complete system architecture, giving way to the detailed explanation of the two main subsystems: the Hand Gesture Recognition subsystem and the UAV Control subsystem. Each one is described in simple but direct terms, which then brings in the implementation details. As aforementioned, ROS is the selected developing platform, hence, a full explanation of the component nodes and topics is provided, accompanied with an overhaul of the system simulation in Gazebo.

Finally, the Evaluation chapter consists of all the tests explained earlier with an interesting proposal regarding the confusion matrixes. Being loyal to the traditional terms, the regular tests were driven, however, a slight different approach is proposed and implemented due to

our own real time system needs, therefore the live data tests for accuracy and time responses are executed. This led to the election of the best classifiers for the proposal, complete the recognizer development life cycle (presented in the Introduction), and therefore, to gain the best possible response from the system.

5.1 Conclusions

The Pattern Recognition skill, so natural and impeccable executed by the humans, turns out to be a true challenge when trying to implement it even in a basic level; it is nonetheless, another small attempt to imitate a cognitive process.

This document holds the project description of the proposal to employ the capabilities of a hardware device that has gained a lot of attention in recent years (the Leap Motion controller) and focus its usage in a very interesting task: the Hand Gesture Recognition through Supervised Learning mechanisms. To turn this proposal into an even more interesting one, it was decided to make use of static and dynamic gestures at the same time in a system that addresses the recognition of specific gestures towards another very exciting task, which is the UAV movement control. The management of a device of this nature implies the supervision of degrees of freedom in all directions (in comparison to a ground mobile robot, where not all directions are required), which leads to the generation of more recognized gestures and therefore, to more characteristics to process and evaluate. As mentioned in Section 2.4.1, the recognizer generation is an iterative work that requires an improvement procedure and demands to be exhausted until a decent output is achieved.

The system

It is widely known that when something works optimally in theory, its response in physical conditions may deliver erratic behaviors that obey to situations that were not either imagined, or simply uncontrollable. That is why it was decided to implement the recognizer in a simulated environment, where at least most of the conditions are able to be manipulated, and control a simulated UAV. The projection displays the UAV movement response to the system commands. The results were very favorable since the aircraft obeyed the commanded orders at its backend. A detail to considerate is the fact that there were some physical variables, owned by the simulator, that were not completely accurate and sometimes did not allow the robot to land 100% perfectly (sometimes it just hanged out, bouncing in small steps all over the floor and was not capable of remaining steady). This behaviour was aided too by the APM autopilot, the software installed in the simulated UAV itself, which tried to hold a position and owned to this simulators physics, capricious UAV movements were sometimes obtained. Since this work focuses on the gesture recognizer deployment and the recognized gestures transformation into UAV commands, we did not spend much time in the research of all the components that affect the simulated system, or the re-programming and tuning of the UAV autopilot software.

The transducer

Since the recognizer reads continuously the Leap Motion retrieved data, it constantly sends the commands to the UAV, hence, in order to manipulate the UAV it was necessary to generate the

desired gesture and afterwards either a fist or remove the hand from the Leap Motion field of view, otherwise, the recognizer would continue sending the most likely command to the hand's stance at that moment. This turned the system into a *discrete* one, meaning this, that the movements commands were ordered in steps towards the desired direction. This is a good behaviour because if the UAV could move unstoppable towards the required direction, it could even go outside of the remote control operation range and coverage area.

The simulation

Naturally, a simulated system helps beating the adversities that a physical system may count with. In our case, it was preferred to stick with the simulation because the physical UAV was never able to hold a steady and controlled position, not even with its own remote control. This would lead to an extremely unstable system if the recognizer system controls the real aircraft, therefore this option was avoided. Since the manipulation and tuning of the internal compass within the autopilot board was beyond the scope of this work, it was decided to bide the simulated system response, which in short words, achieved its goal.

The recognition

It is interesting to emphasize the accomplishment of enabling the dynamic gesture recognition making use of tools dedicated to static gestures recognition. This was possible due to the employed approach: generation and addition of temporal features to the vector delivered to the core classifiers. Worth to say is that this approach is not owned by the device, but may be implemented in whatever other device that, in similar way, delivers a skeletal hand model in some framerate. The frame number was set to ten in this project, however, as future work, a series of comparatives could be generated in order to determine an optimal number for this parameter.

Of course some other problems were encountered during the development of this project, and among them, some interesting ones related to Leap Motion are described as following. The Leap Motion controller does not work at a fixed, adjustable or constant frame rate [58], therefore the temporal features recorded in the Training Datasets may not fit if they are used in a computer with different features, or may even differ in the same computer if the conditions are very distinct (like running programs with high memory demands at the same time). Due to this, a very high amount of tests were driven, more than fifty Training Datasets recorded and of course, different recognizers implementation proposals. The most promising results are the ones displayed in the Evaluation chapter.

The Leap Motion usage brought another extra complication. As commented earlier, the developers community is aware of the following situation [44]: Leap Motion is a device optimized for the position tracking in the XY plane (the plane parallel to a computer's monitor) because its intended use is towards the hand interaction with objects in a screen. This feature brings along the fact that the tracking position for the Z direction is not as optimal as for the X or Y direction and therefore, the retrieved information for the Z direction may not be very accurate. Since the recognizer controls the UAV in six degrees of freedom, the tracking of the Z position turns out to be vital in at least a third part, which impacts directly the over-all system performance. Adding to this effect, the problem where Leap Motion sometimes loses certain Frames and delivers zeroed data, it was necessary to implement data filtering, which nevertheless helped, they did not

completely eliminated the errors due to these effects. The Evaluation chapter shows that the accuracy for the movements in the XY plane is higher than the ones where the Z direction plays an important role.

The election of the core classifiers was perhaps the most demanding and exhausting step of this work given its fundamental importance. This is the reason why it was necessary to study very detailedly their answers and yet, generate mechanisms that allowed us to compare their behavior in a more suitable way in regards of this project's approach. This is how the Live Data tests were generated through the incorporation of the modified confusion matrixes for the multiclassifiers. From these matrixes, maybe the most important number is the accuracy percentage for each recognized-received class pair, however, it is not the only interesting information that can be extracted from them.

5.2 Further Work

The limitations and difficulties explained in above paragraphs are some of the matters that may be managed by intelligent mechanisms (or mechanisms that are not explicitly programmed to do so) such as the ones that Machine Learning delivers. This transition from the rigid programming towards more flexible models, provides some clearance that allows the inclusion of problems out of our reach (as the ones exposed in previous paragraphs).

Of course it is possible that a single classifier would be employed in this project, however it is well known that error happens, and for a system with an application such as the one our system counts with, the errors must be reduced to the minimum. Therefore it was thought in countervailing their effect taking in account the spirit of the Ensembled methodology, which promotes the diversity of hypothesis making use of the abilities that every classifier's approach has. It would be tremendously interesting try with alternative ensemble methodologies such as boosting and including other machine learning algorithms, there exists an amazingly wide offer to explore.

Knowledge should be free and open, promoted by people with access to it in such a way that the community gets benefits of it. This is the reason why this project made use of open source tools such as Linux, ROS and the APM project. This last one is integrated in its majority by technology adepts and enthusiastic programmers, moved by the desire of developing open code for UAV autopilot needs. On its side, the ROS project was born with the set of mind of not *reinventing the wheel* every time a person wishes to put his hands on the robotics world in sake of an achievement of more complex proposals. With this said, this work aims to hold this philosophy and keep up this profile. Because, if we stop for a minute and think about the human evolution, isn't it true that through the diversity of opinions, ideas and knowledge of misfits, rebels and crazy thinkers, humanity and science have achieved one step ahead everytime?

Unity is variety and variety in unity is
the supreme law of the universe.

Isaac Newton

Bibliography

- [1] Association for computing machinery, *computing classification system, 2012 revision*. <http://delivery.acm.org/10.1145/2380000/2371137/ACMCCSTaxonomy.html>, Accessed: 2016-06-14.
- [2] J. Fiset, *Human-Machine Interface Design for Process Control Applications*. Instrumentation, Systems, and Automation Society, 2009, ISBN: 9781934394359. [Online]. Available: https://books.google.com/books?id=NE_TEJBmwi8C.
- [3] G. Salvendy, *Handbook of Human Factors and Ergonomics*. Wiley, 2012, ISBN: 9781118129081. [Online]. Available: <https://books.google.com/books?id=WxJVNLzvRVUC>.
- [4] *HMI product design and development*, <http://sites.ieee.org/tcrtts/education/terminology-and-notation/>, Accessed: 2016-03-01.
- [5] *Wikipedia: Real time system*, https://en.wikipedia.org/wiki/Real-time_computing, Accessed: 2016-03-01.
- [6] E. Ueda, Y. Matsumoto, M. Imai, and T. Ogasawara, "Hand pose estimation using multi-viewpoint silhouette images", in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2001: Expanding the Societal Role of Robotics in the the Next Millennium, Maui, HI, USA, October 29 - November 3, 2001*, 2001, pp. 1989–1996. DOI: 10.1109/IROS.2001.976365. [Online]. Available: <http://dx.doi.org/10.1109/IROS.2001.976365>.
- [7] S. Hoffberg and L. Hoffberg-Borghesani, *Ergonomic man-machine interface incorporating adaptive pattern recognition based control system*, US Patent 5,875,108, 1999. [Online]. Available: <https://www.google.com/patents/US5875108>.
- [8] R. Fiebrink, "Real-time human interaction with supervised learning algorithms for music composition and performance", PhD thesis, Princeton University, Princeton, NJ, USA, 2011.
- [9] S. Bilal, R. Akmeliawati, M. J. E. Salami, and A. A. Shafie, "Vision-based hand posture detection and recognition for sign language", in *Mechatronics (ICOM), 2011 4th International Conference On*, 2011, pp. 1–6. DOI: 10.1109/ICOM.2011.5937178.
- [10] K. Liu and N. Kehtarnavaz, "Real-time robust vision-based hand gesture recognition using stereo images", *Journal of Real-Time Image Processing*, vol. 11, no. 1, pp. 201–209, 2016, ISSN: 1861-8219. DOI: 10.1007/s11554-013-0333-6. [Online]. Available: <http://dx.doi.org/10.1007/s11554-013-0333-6>.
- [11] R. Z. Khan and N. A. Ibraheem, "Survey on gesture recognition for hand image postures", *Computer and Information Science*, vol. 5, pp. 110–121, 2012.
- [12] P. Mátételki, M. Pataki, S. Turbucz, and L. Kovács, "An assistive interpreter tool using glove-based hand gesture recognition", in *Humanitarian Technology Conference - (IHTC), 2014 IEEE Canada International*, 2014, pp. 1–5. DOI: 10.1109/IHTC.2014.7147529.

- [13] P. Pławiak, T. Sośnicki, M. Niedźwiecki, Z. Tabor, and K. Rzecki, "Hand body language gesture recognition based on signals from specialized glove and machine learning algorithms", *IEEE Transactions on Industrial Informatics*, vol. 12, no. 3, pp. 1104–1113, 2016, ISSN: 1551-3203. DOI: [10.1109/TII.2016.2550528](https://doi.org/10.1109/TII.2016.2550528).
- [14] S. S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: A survey", *Artif. Intell. Rev.*, vol. 43, no. 1, pp. 1–54, 2015. DOI: [10.1007/s10462-012-9356-9](https://doi.org/10.1007/s10462-012-9356-9). [Online]. Available: <http://dx.doi.org/10.1007/s10462-012-9356-9>.
- [15] G. Marin, F. Dominio, and P. Zanuttigh, "Hand gesture recognition with leap motion and kinect devices", in *Image Processing (ICIP), 2014 IEEE International Conference on*, 2014, pp. 1565–1569. DOI: [10.1109/ICIP.2014.7025313](https://doi.org/10.1109/ICIP.2014.7025313).
- [16] F. Weichert, D. Bachmann, B. Rudak, and D. Fisseler, "Analysis of the accuracy and robustness of the leap motion controller", *Sensors*, vol. 13, no. 5, p. 6380, 2013, ISSN: 1424-8220. DOI: [10.3390/s130506380](https://doi.org/10.3390/s130506380). [Online]. Available: <http://www.mdpi.com/1424-8220/13/5/6380>.
- [17] D. Bachmann, F. Weichert, and G. Rinkenauer, "Evaluation of the leap motion controller as a new contact-free pointing device", *Sensors*, vol. 15, no. 1, p. 214, 2014, ISSN: 1424-8220. DOI: [10.3390/s150100214](https://doi.org/10.3390/s150100214). [Online]. Available: <http://www.mdpi.com/1424-8220/15/1/214>.
- [18] M. Spiegelmock, *Leap Motion Development Essentials*. Packt Publishing, 2013.
- [19] *Leap Motion developers site*, <https://developer.leapmotion.com/>, Accessed: 2015-11-21.
- [20] *Parrot developer portal*, <http://developer.parrot.com/>, Accessed: 2015-11-21.
- [21] M. Hu, F. Shen, and J. Zhao, "Hidden markov models based dynamic hand gesture recognition with incremental learning method", in *2014 International Joint Conference on Neural Networks, IJCNN 2014, Beijing, China, July 6-11, 2014*, 2014, pp. 3108–3115.
- [22] P. Molchanov, S. Gupta, K. Kim, and J. Kautz, "Hand gesture recognition with 3d convolutional neural networks", in *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops, Boston, MA, USA, June 7-12, 2015*, 2015, pp. 1–7.
- [23] G.-W. Wang, C. Zhang, and J. Zhuang, "An application of classifier combination methods in hand gesture recognition", *Mathematical Problems in Engineering*, vol. 2012, 2012.
- [24] T. Kopinski, S. Magand, A. R. T. Gepperth, and U. Handmann, "A light-weight real-time applicable hand gesture recognition system for automotive applications", in *2015 IEEE Intelligent Vehicles Symposium, IV 2015, Seoul, South Korea, June 28 - July 1, 2015*, 2015, pp. 336–342.
- [25] P. Modler and T. Myatt, "Video based recognition of hand gestures by neural networks for the control of sound and music", in *8th International Conference on New Interfaces for Musical Expression, NIME 2008, Genova, Italy, June 5-7, 2008*, 2008, pp. 358–359.
- [26] *Intel real sense sdk*, <https://software.intel.com/en-us/intel-realsense-sdk>, Accessed: 2017-06-15.
- [27] R. C. Luo and Y. Wu, "Hand gesture recognition for human-robot interaction for service robot", in *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, MFI 2012, Hamburg, Germany, September 13-15, 2012*, 2012, pp. 318–323.

- [28] C. Li, H. Ma, C. Yang, and M. Fu, "Teleoperation of a virtual icub robot under framework of parallel system via hand gesture recognition", in *IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2014, Beijing, China, July 6-11, 2014*, 2014, pp. 1469–1474.
- [29] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. Wiley, 2012, ISBN: 9781118586006. [Online]. Available: <https://books.google.com.mx/books?id=Br33IRC3PkQC>.
- [30] G. Dougherty, *Pattern Recognition and Classification, An Introduction*. Springer, 2013.
- [31] M. S. D. Rose and C. C. Wagner, "Survey on classifying human actions through visual sensors", *Artificial Intelligence Rev*, vol. 37, pp. 301–311, 2012.
- [32] C. K. Mohan, "Artificial intelligence: Pattern recognition", *Salem Press Encyclopedia of Science*, 2015.
- [33] J. A. Fails and D. R. Olsen Jr., "Interactive machine learning", in *Proceedings of the 8th International Conference on Intelligent User Interfaces*, ser. IUI '03, Miami, Florida, USA: ACM, 2003, pp. 39–45, ISBN: 1-58113-586-6. DOI: 10.1145/604045.604056. [Online]. Available: <http://doi.acm.org/10.1145/604045.604056>.
- [34] A. L. Samuel, "Programming computers to play games", *Advances in Computers*, vol. 1, pp. 165–192, 1960. DOI: 10.1016/S0065-2458(08)60608-7. [Online]. Available: [http://dx.doi.org/10.1016/S0065-2458\(08\)60608-7](http://dx.doi.org/10.1016/S0065-2458(08)60608-7).
- [35] T. M. Mitchell, *Machine learning*, ser. McGraw Hill series in computer science. McGraw-Hill, 1997, ISBN: 978-0-07-042807-2.
- [36] S. Marsland, *Machine learning : An algorithmic perspective*, ser. Chapman and Hall CRC machine learning and pattern recognition series. Boca Raton: CRC Press, 2009, A Chapman & Hall book., ISBN: 978-1-4200-6718-7. [Online]. Available: <http://opac.inria.fr/record=b1129336>.
- [37] D. Wolpert and W. G. Macready, "No free lunch theorems for optimization", *IEEE Trans. Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997. DOI: 10.1109/4235.585893. [Online]. Available: <http://dx.doi.org/10.1109/4235.585893>.
- [38] N. Gillian and R. Benjamin, "An adaptive classification algorithm for semiotic musical gestures", in *In the 8th Sound and Music Computing Conference (SCM2011)*, 2011.
- [39] I. Steinwart and C. Scovel, "Mercer's theorem on general domains: On the interaction between measures, kernels, and rkhs", *Constructive Approximation*, vol. 35, no. 3, pp. 363–417, 2012, ISSN: 1432-0940. DOI: 10.1007/s00365-012-9153-3. [Online]. Available: <http://dx.doi.org/10.1007/s00365-012-9153-3>.
- [40] M. Warmuth, "Sample compression, learnability, and the vapnik-chervonenkis dimension", in *Computational Learning Theory: Third European Conference, EuroCOLT '97 Jerusalem, Israel, March 17–19, 1997 Proceedings*, S. Ben-David, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 1–2, ISBN: 978-3-540-68431-2. DOI: 10.1007/3-540-62685-9_1. [Online]. Available: http://dx.doi.org/10.1007/3-540-62685-9_1.
- [41] M. A. Arbib, *Handbook of brain theory and neural networks*, 2nd ed. The MIT Press, 2002, ISBN: 0262011972,9780262011976,9780585457406.
- [42] L. I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004, ISBN: 0471210781,9780471210788. [Online]. Available: <http://gen.lib.rus.ec/book/index.php?md5=D3E45AB795C1549CFC5A3374FEA50373>.

- [43] *Leap Motion public introduction*, <https://www.leapmotion.com/news/leap-motion-launches-world-s-most-accurate-3-d-motion-control-technology-for-computing>, Accessed: 2015-11-21.
- [44] *Leap Motion and the z axis accuracy*, <https://community.leapmotion.com/t/stabilized-palm-z-component-has-opposite-than-intended-behaviour/528/2>, Accessed: 2016-05-31.
- [45] *The MavLink protocol*, <http://qgroundcontrol.org/mavlink/start>, Accessed: 2016-05-31.
- [46] *Erle brain 2 git book*, <https://erlerobotics.gitbooks.io/erle-robotics-erle-brain-a-linux-brain-for-drones/content/en/>, Accessed: 2016-05-31.
- [47] *Arducopter project*, <http://ardupilot.org/copter/index.html>, Accessed: 2016-05-31.
- [48] *Dronocode software platform*, <https://www.dronocode.org/dronocode-software-platform>, Accessed: 2016-05-31.
- [49] *Mission planner*, <http://ardupilot.org/planner/index.html>, Accessed: 2016-05-31.
- [50] *Mavproxy*, http://qgroundcontrol.org/mavlink/mavproxy_startpage, Accessed: 2016-05-31.
- [51] *The robotic operating system (ros)*, <http://www.ros.org/>, Accessed: 2016-05-31.
- [52] *Open source robotics foundation*, <http://www.osrfoundation.org/>, Accessed: 2016-05-31.
- [53] *Ros wiki*, <http://wiki.ros.org/ROS/Introduction>, Accessed: 2016-05-31.
- [54] *Gazebo simulator*, <http://gazebo.org/>, Accessed: 2016-05-31.
- [55] *Ardupilot: software in the loop*, <http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html>, Accessed: 2016-06-31.
- [56] *Gazebo hector plugin*, https://github.com/tu-darmstadt-ros-pkg/hector_gazebo/, Accessed: 2016-06-31.
- [57] N. Gillian and J. A. Paradiso, "The gesture recognition toolkit", *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3483–3487, Jan. 2014, ISSN: 1532-4435. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2627435.2697076>.
- [58] *Leap motion frame rate*, <https://developer.leapmotion.com/documentation/cpp/api/Leap.Frame.html>, Accessed: 2016-05-31.